Università degli Studi di Ferrara

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Specialistica in Informatica

Development of a Soil Classification Web Service and integration onto the SSE portal

Relatore:
Dott. Mirco Andreotti

Laureando:
Alan Beccati

Corelatori:
Dott. Marco Folegani, *Dott.* Stefano Natali

Controrelatore:
Prof. Eleonora Luppi

Anno accademico 2005-2006

to my family
to my lovely girlfriend Erminia

Preface (Italian)

L'Agenzia Spaziale Europea (ESA), con l'obiettivo di espandere il segmento Earth Observation del mercato Europeo, ha promosso lo sviluppo del sistema Service Support Environment (SSE); questo sistema avvicina clienti ed aziende del settore, fornendo all'utente finale gli strumenti per facilitare la ricerca e l'uso dei prodotti integrati nel sistema ed alle aziende un sistema standardizzato per la commercializzazione dei loro servizi. L'attività principale del mercato di riferimento è la trasformazione di immagini satellitari in informazione. L'utilizzo di metodi standardizzati di comunicazione ed accesso a dati e servizi, facilita inoltre la cooperazione tra le aziende. Un punto centrale di pubblicazione ed accesso, quale il portale SSE, realizza di fatto un mercato virtuale dove ogni cliente può facilmente ricercare un servizio adatto alle sue necessità.

Oltre che per applicazioni di mercato, il portale SSE viene utilizzato anche per progetti di ricerca interni ad ESA: in quest'ambito il progetto KIM Extensions and Installations (KEI) prevede lo sviluppo di servizi web in ambiente SSE per essere utilizzati in ambito di classificazione automatica e semi-automatica del suolo ed applicazioni semantiche di ricerca su ampi database.

MEEO, un'azienda Italiana operante nel settore EO, ha realizzato un sistema automatizzato di classificazione del suolo basato sull'analisi di immagini satellitari multispettrali. L'integrazione di questo servizio sul portale SSE consentirebbe ai ricercatori l'accesso alle sue innovative funzionalità, contribuendo all'obiettivo del sistema SSE. Il sistema SSE, in un ottica orientata ai servizi, implementa un architettura di tipo Service Oriented Architecture (SOA) che prevede la pubblicazione di ogni servizio da integrare come Web Service.

Obiettivo di questa tesi è l'integrazione del classificatore SOIL MAPPER (SM), sviluppato da MEEO, sul portale SSE secondo le specifiche del sistema, attraverso l'utilizzo degli strumenti di supporto resi disponibili per SSE.

Questo elaborato è strutturato nel seguente modo:

- Il primo capitolo è una presentazione introduttiva di SM, del quale viene fornita una panoramica di funzionamento, dei principi di base e del tipo di dati utilizzati nell'elaborazione. Il capitolo si chiude con una breve introduzione al sistema SSE.

- Il secondo capitolo descrive brevemente la maggior parte delle tecnologie coinvolte nello sviluppo di un servizio integrato nel sistema SSE. Sono introdotti gli standard di base atti a garantire l'interoperabilità del sistema quali: EXtensible Markup Language (XML), per la definizione dei dati ed EXtensible Stylesheet Language (XSL) per la loro trasformazione. Nel capitolo sono inoltre descritte le due principali tecnologie impiegate nel progetto: i Web Services ed il linguaggio Business Process Execution Language (BPEL) per la loro organizzazione in processi eseguibili.

- Il terzo capitolo descrive il processo di integrazione del servizio SOIL MAPPER nel sistema SSE; il capitolo si apre con una panoramica completa sul sistema integrato seguita dall'analisi dei singoli componenti realizzati.

- Il quarto capitolo contiene le conclusioni, alcuni possibili sviluppi futuri del sistema ed alcuni suggerimenti per il possibile miglioramento degli strumenti di supporto all'integrazione dei servizi.

- Nelle due appendici di questo elaborato sono descritti rispettivamente la procedura completa di installazione e configurazione dell'elaboratore che ospita il servizio SM ed i listati principali relativi a tale servizio.

L'algoritmo di classificazione del suolo SM analizza immagini satellitari multispettrali, basandosi sul riconoscimento di particolari modelli di emissione e riflessione delle radiazioni detti *firme spettrali*. I sensori Thematic Mapper ed Enhanced Thematic Mapper installati sui satelliti Landsat 5 e Landsat 7 captano diverse bande di emissione comprese tra lo spettro visibile ed l'infrarosso; questi dati vengono calibrati in corrispondenti valori di riflettanza e temperatura per formare la base di analisi per la classificazione. Il risultato dell'analisi consiste in uno strato aggiuntivo dell'immagine di partenza, contenente la classe di appartenenza di ogni pixel in essa contenuto. La classificazione può essere fatta secondo tre livelli con precisione crescente: small, medium e large.

Il sistema SSE facilita all'utente l'accesso alle risorse in esso registrate presentando un'interfaccia uniforme, accessibile tramite un comune Web browser, che rende disponibile un catalogo, organizzato in categorie, di tutti i servizi disponibili attraverso il sistema. Sfruttando il Process Manager integrato nel sistema, inoltre, è possibile fornire all'utente servizi complessi realizzati attraverso la composizione di più servizi di base registrati nel sistema SSE.

Mediante il portale SSE, un utente può effettuare la ricerca di un servizio che fornisca l'elaborazione desiderata, quindi effettuare ricerche su cataloghi di immagini collegati al servizio scelto ed, infine, richiedere l'elaborazione delle immagini selezionate dai risultati della precedente ricerca. Se l'elaborazione richiesta dall'utente dovesse necessitare dell'utilizzo di più servizi, supponendo che questi siano concatenati in un singolo servizio complesso, egli potrebbe ottenere il risultato finale dell'elaborazione mediante una singola richiesta al servizio complesso.

Il sistema SSE garantisce l'interoperabilità fra i vari sistemi coinvolti mediante l'utilizzo di standard aperti ed estensibili quali:

XML un linguaggio per la descrizione delle informazioni estensibile basato unicamente su file di testo semplice, utilizzabili quindi su qualsiasi piattaforma tramite un interprete di questo linguaggio. Un documento XML non utilizza tag standardizzati ma si basa su uno schema, anch'esso scritto in XML, che ne consente la corretta interpretazione;

XSL un insieme di tre standard progettati per definire come l'informazione contenuta nei documenti XML possa essere presentata, selezionata o trasformata;

Web Services la tecnologia utilizzata per la comunicazione tra il sistema SSE ed i servizi integrati. L'architettura Web Services si basa sull'estensione del linguaggio XML Simple Object Access Protocol (SOAP), orientata alla veicolazione di messaggi XML tra sistemi eterogenei.

BPEL un linguaggio standardizzato, basato sull'XML, progettato per orchestrare diversi Web Services in un unico processo. Un processo BPEL può essere astratto oppure eseguibile; SSE utilizza per l'esecuzione dei processi BPEL *Oracle Process Manager*.

Il sistema SSE mette a disposizione uno strumento, chiamato TOOLBOX, per facilitare ai service provider l'integrazione dei loro servizi sollevandoli dalla necessità di realizzare da zero l'interfaccia richiesta dai Web Service per ricevere ed interpretare i messaggi scambiati con l'SSE. La TOOLBOX è uno strumento configurabile tramite uno specifico linguaggio di scripting XML; al service provider è richiesta la realizzazione degli script di programmazione e di un sistema di comunicazione fra la TOOLBOX ed il proprio sistema informativo dove risiede il servizio da integrare.

Il sistema SSE fornisce una specifica formale delle operazioni che un servizio integrato può fornire, queste operazioni sono:

Search utilizzabile per effettuare ricerche di immagini tramite servizi di catalogo;

Present utilizzabile per richiedere informazioni più dettagliate su un risultato di una precedente operazione di Search;

Request For Quotation (RFQ) utilizzata per richiedere informazioni riguardanti il servizio quali, ad esempio, prezzi praticati secondo l'elaborazione richiesta o presentazione di risultati dimostrativi sul servizio fornito;

Order utilizzabile per effettuare un'effettiva richiesta di elaborazione ad un servizio.

Nessuna di queste operazioni è obbligatoria per un servizio che potrebbe, ad esempio, fornire solo le prime due operazioni (fornendo di fatto un servizio di consultazione per un catalogo) od anche la sola operazione di Order.

L'interfaccia che un servizio integrato in SSE mette a disposizione dell'utente è definita nel suo foglio di stile: un documento XSL che contiene dei modelli di trasformazione, selezionati dal Portale SSE secondo l'operazione richiesta dall'utente. Il foglio di stile contiene inoltre le direttive per la composizione dei messaggi XML scambiati tra il sistema SSE ed il servizio stesso.

Per gli scopi di questo progetto, il servizio SM deve supportare il catalogo ufficiale ESA, Earthnet OnLine Interactive (EOLI), e consentire inoltre l'elaborazione di immagini direttamente fornite dall'utente; per soddisfare questi requisiti stata realizzato un unico servizio di base, integrato sul sistema SSE in due versioni differenti: una versione FTP, per consentire l'elaborazione di immagini accessibili tramite un indirizzo FTP fornito dall'utente ed una versione EOLI che fornisce il supporto dell'omonimo catalogo. L'integrazione del servizio SOIL MAPPER sul sistema SSE è ha comportato le attività

descritte nel capitolo 3, di seguito brevemente riportate:

- Installazione e configurazione di un server di calcolo dedicato al servizio SM, descritta in appendice A.
- Analisi del modello d'utilizzo del classificatore SM e realizzazione del collegamento (basato su script di shell Linux) fra TOOLBOX e SM.
- Realizzazione dello schema del servizio e dei necessari script di configurazione per la TOOLBOX
- Realizzazione dei processi BPEL per la gestione delle operazioni fornite dal servizio SM.
- Realizzazione dei fogli di stile per la gestione dell'interazione con l'utente.
- Registrazione sul portale SSE delle due versioni del servizio realizzate.
- Verifica del corretto funzionamento delle due versioni del servizio tramite il *test center* incluso nella TOOLBOX ed il portale SSE.

Il servizio SOIL MAPPER sarà consegnato ad ESA per l'installazione nei suoi sistemi di calcolo, rendendo così disponibili, ai ricercatori associati all'Agenzia, le innovative funzionalità del classificatore SM. Questo servizio sarà inoltre utilizzato nell'ambito del progetto KEI, andando a costituire parte del sistema di estrazione di informazioni previsto dal progetto.

L'utilizzo del servizio sui sistemi di ESA e nell'ambito del progetto KEI è riservato a scopi non commerciali; una versione commerciale del servizio, con le opportune integrazioni proposte nella sezione 4.2, sarà integrata nel sistema SSE e gestita direttamente da MEEO.

Contents

List of Figures

List of Tables

Listings

Introduction

Today's European Earth Observation (EO) market services (e.g. the transformation from satellite images into information) are performed by a small number of specialised companies each operating independently in its specific application domain. This separation prevents the optimization of allocated resources, limits the visibility of the companies to potential customers and provides no standard way for customers to access the company's services or for companies to cooperate in synergy. Aiming to solve these problems and in an effort to broaden the European EO market, the Service Support Environment (SSE) has been developed for the Ground Segment Research and Technology Development (RTD) Department at European Space Agency (ESA) - ESRIN (the ESA Centre for Earth Observation, one of the five ESA specialised European centers).

SSE implements an open service-oriented and distributed environment for users, service providers and data providers integration into a coherent supply chain thus facilitating service provision and companies collaboration.

Adhering to a Service Oriented Architecture (SOA) point of view, the SSE system leverages Web Services technology for its communication infrastructure so a service provider, to integrate its value added service, must implement it as a Web Service which receives the SSE requests and reacts according to their content. Intecs S.p.A., an italian software company participating in the SSE framework development, had developed a server side application, called TOOLBOX, to aid service providers in their service development.

The SSE system interface has been developed to be used also in the framework of the ESA - ESRIN Ground Segment Projects, these projects involve development of several EO services focused on the Image Information Mining. This initiative is amied at extracting useful information from the European satellite image databases.

Meteorological and Environmental Earth Observation (MEEO), an italian company located near Ferrara, has developed SOIL MAPPER (SM), a soil classification system based on spectral signals remotely detected by a satellite's sensor. MEEO has been involved in the KIM Extension and Installation (KEI) project with the aim to integrate its SM services into the Knowledge based Information Mining (KIM) system. KIM is one of the Image Information Mining (IIM) projects sponsored by ESA that focuses on the implementation of a new information mining technique, differing from traditional feature extraction methods which analyse pixels looking for predefined patterns. KIM extracts and stores basic image pixel and area characteristics (Primitive Features), which are then selected by users as representative of the searched high-level feature. The weighted combination of one or more Primitive Features, resulting from such training, can be associated by the user to a specific semantic meaning, closely linked to his domain and knowledge[1].

Following ESA SOA oriented intetgration system, scope of this thesis is the integration of the SM classification algorithm onto the SSE system. This writing describes the SSE system, focusing on the service integration process and details the integration of the SM classification system.

This thesis describes some essential concepts of the SSE portal, focusing on the service integration system, and the MEEO classifier. Furthermore it details the integration process performed in order to publish SM as a web service integrated on the SSE portal using the Intecs TOOLBOX. The contents of this writing is structured in the following chapters:

Chapter 1 This introductory chapter provides an overview of the SM classification algorithm, its principles and SSE: the target system for the SM integration;

Chapter 2 Provides a brief description of the technologies and standards upon which the SSE system is built, description of the Intecs TOOLBOX is also included since it is the main integration supporting tool of the SSE framework;

Chapter 3 Details the developed classification service and its integration onto the SSE Portal;

Chapter 4 Holds conclusions and possible further work.

Chapter 1

Automated soil classification system and SSE

To define the context of the project upon which this writing is based, this chapter describes the SOIL MAPPER (SM) classification algorithm (along with its base principles of spectral categorization and the kind of images used as its input) and the SSE portal and its founding SOA philosophy.

1.1 Principles of spectral categorization

Water, sand, bare soil, plants, urban/artificial areas, etc. differ in the way they reflect or emits electromagnetic radiation at different wavelengths. The human eye perceives these differences in the optical (visible) spectrum as color therefore the color of the observed portion of earth surface, differs according to its material composition or coverage.

Besides visible spectrum, the most effective electromagnetic bands for earth surface characterization are infrared (IR) and microwave (MW) spectra. Each material has a specific electromagnetic response to direct solar illumination or emission in the visible and IR spectrum: this is called spectral signature. Any portion of the earth surface can thus be characterized by means of its spectral signature but when dealing with discrete units of measurement like pixels of a satellite acquired image covering different mixed terrain types some fuzziness will be introduced into the signature definition depending on the image resolution and this must also be considered.

1.2 Landsat imagery system

The Landsat Program is a series of Earth-observing satellite missions jointly managed by the National Aeronautics and Space Administration (NASA) and the United States

Geological Survey (USGS). Landsat satellites collect information about Earth from space (a science known as Remote Sensing (RS)). The Landsat satellites we consider in this work are Landsat5 and Landsat7 both flying at a 705 km high sun synchronous orbit collecting multi-band earth data with different sensors[2]. The acquired data is collected into scenes concerning a specific stripe segment of the continuous satellite's 185Km wide data acquisition swath. Each satellite has a different set of sensors on board; the classification algorithm considered in this work is focused on the utilization of the following sensors:

- The Thematic Mapper (TM) sensor mounted on the Landsat5 satellite which provides a multi-band acquisition divided into six 30mt resolution bands (ranging from the visible blue band to the mid wave infrared band) and a single 120m resolution thermal infrared band, see Table 1.1[2];

- The Enhanced Thematic Mapper (ETM) sensor mounted on the Landsat7 satellite which provides a multi-band acquisition divided into six 30mt resolution bands (ranging from the visible blue band to the mid wave infrared band), a single 60m resolution thermal infrared band and a panchromatic 15mt resolution band (covering green, red, and near infrared wavelengths), see Table 1.2[2].

Band	Wavelength (μm)	Resolution (m)	Swath Width (km)	Revisit time(days)
Band 1 (VIS)	0.45 to 0.52	30	185	16
Band 2 (VIS)	0.52 to 0.6	30	185	16
Band 3 (VIS)	0.63 to 0.69	30	185	16
Band 4 (NIR)	0.76 to 0.9	30	185	16
Band 5 (SWIR)	1.55 to 1.75	30	185	16
Band 6 (TIR)	10.4 to 12.5	120	185	16
Band 7 (SWIR)	2.08 to 2.35	30	185	16

Table 1.1: Thematic Mapper bands technical data

Band	Wavelength (μm)	Resolution (m)	Swath Width (km)	Revisit time(days)
Band 1 (VIS)	0.45 to 0.515	30	185	16
Band 2 (VIS)	0.525 to 0.605	30	185	16
Band 3 (VIS)	0.63 to 0.69	30	185	16
Band 4 (NIR)	0.75 to 0.9	30	185	16
Band 5 (SWIR)	1.55 to 1.75	30	185	16
Band 6 (TIR)	10.4 to 12.5	60	185	16
Band 7 (MWIR)	2.08 to 2.35	30	185	16
Band PAN (VIS)	0.52 to 0.9	15	185	16

Table 1.2: Enhanced Thematic Mapper bands technical data

1.3 SOIL MAPPER overview

SM is an innovative RS image rule-based classifier based on spectral prior knowledge exclusively[3]. SM takes inspiration from a paper published in RS literature[4]. The algorithm input must be an RS image calibrated into planetary reflectance and absolute temperature values and its generated output is a preliminary (baseline) classification map with each pixel of the input image classified into a discrete set of spectral categories. These spectral categories feature a semantic meaning and are suitable for driving further image analysis on a stratified basis. A key feature of SM is full automation since its execution requires neither user supervision nor ground truth data sample.

1.3.1 Architecture

The SM classification system is based on a set (dictionary) of spectral rules built from *a priori* spectral knowledge consisting of spectral signatures extracted from existing RS literature. A two stage architecture allows to handle the inherent variability (fuzziness) of land cover class-specific spectral signatures.

The first stage classifies each pixel-based input data against a logical (and, or) combination of inter-band relative relationships (e.g. `band 1 > band 2`) provided with tolerance intervals providing a possibly multiple classification by the spectral rules.

The second stage classifies each pixel-based input data by an irregular but complete grid-partition of the input feature space given by a logical combination of fuzzy sets (e.g. `band 1 is high`).

Described in its general logic schema, SM consists of the three processing blocks depicted in Figure 1.1

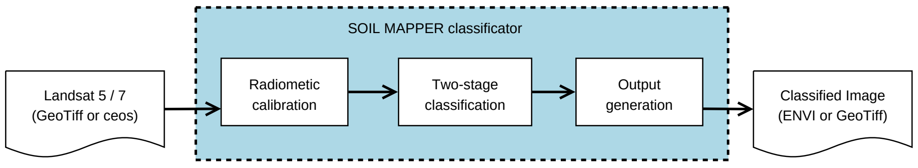

SOIL MAPPER classificator
Landsat 5 / 7 (GeoTiff or ceos)
Radiometic calibration
Two-stage classification
Output generation
Classified Image (ENVI or GeoTiff)

Figure 1.1: SOIL MAPPER logical processing blocks

The SM implementation used for this work accepts either the GeoTiff or the Committee on Earth Observation Satellites (CEOS) data formats and can write its classified output image files in either ENVI or GeoTiff data formats. The three processing blocks functionalities can be summarized as follows:

Radiometric calibration performs a conversion from the raw multi-band image data to the needed reflectance and absolute temperature values of the corresponding bands;

Two-stage classification is the core SM rule based classification algorithm implementation;

Output generation although SM supports three primary types of output products[1], only the classification map of the input image is produced as elaboration result for this service integration.

1.3.2 Classification levels

In this service integrated version SOIL MAPPER generates, at the user choice, one of three output classification maps featuring different levels of informational granularity (number of discrete classes): the Large, Intermediate and Small sets of output categories consisting of 72, 38 and 15 spectral categories respectively. An example of each classification level output obtained from the same source image (extracted from a larger Landsat 7 scene to focus on Lipari island) is depicted in Fig.1.2 where it can be noticed how the coarseness of the classification results grows as the number of classes is reduced.

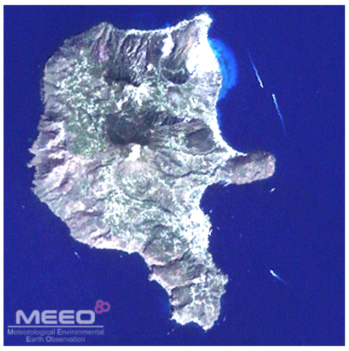
1.4 Services Support Environment

Driven by the need to follow a market-pulled user-oriented approach to earth observation implementation strategy, as stated in the "Oxygen project" [5], the European Space Agency had sponsored and is running the eoPortal[2] which aims to improve market expansion of EO products and services providing a content updated user-friendly website as a single access point for those EO products and services thus enhancing people's ways to access EO resources.

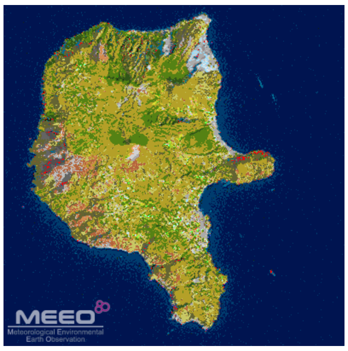
Among the eoPortal provided features (like satellite imagery catalogues, map services and cartographic resources), the SSE portal is such a centralized access point that coordinates access to a wide range of value added EO services. A key feature of the SSE portal is its Interface Control Document (ICD) which holds the formal rules to publish those value added services in a standard way, providing full interoperability between all the adhering service provider systems. A service provider is any organization publishing a value added

[1]Classification maps featuring different aggregation levels of spectral layers, header files summarizing image-wide classification statistics and continuous spectral indexes (Greenness, Canopy chlorophyll content, Canopy water content and Water index).

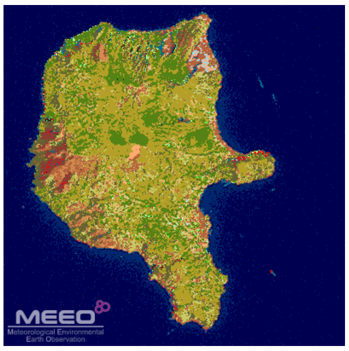
[2]The *eoPortal: Sharing Earth Observation Resources* is located at `http://www.eoportal.org`



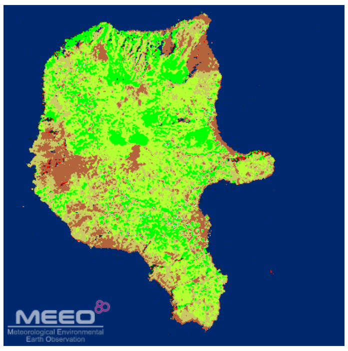
(a) Input



(b) Large



(c) Intermediate



(d) Small

Figure 1.2: An example of the different classification outputs from the input Landsat 7 image of Lipari island acquired on Sep-26-1999. (a) is the input image depicted using the three visible bands (red, green and blue); (b) is the Large output set classification (72 classes); (c) and (d) are the Intermediate (38 classes) and Small (15 classes) output sets respectively.

EO service on the SSE portal. Here follows an example based overview of its working model; a more detailed description can be found in the SSE whitepaper[6] and in section 2.4.

To get an overview of how the service integration is obtained through the use of the SSE system lets suppose that a user wants to find an image of interest on a data provisioning catalogue and then he needs some value added processing of the image. Suppose also that the required processing system is provided by the sequential application of three elaboration services registered on the SSE portal by two different providers; this sequential application can also be published as a single chained EO service, even if the composing services are provided by different service providers. Figure 1.3 shows this chained service scenario and the key interactions among the involved entities.

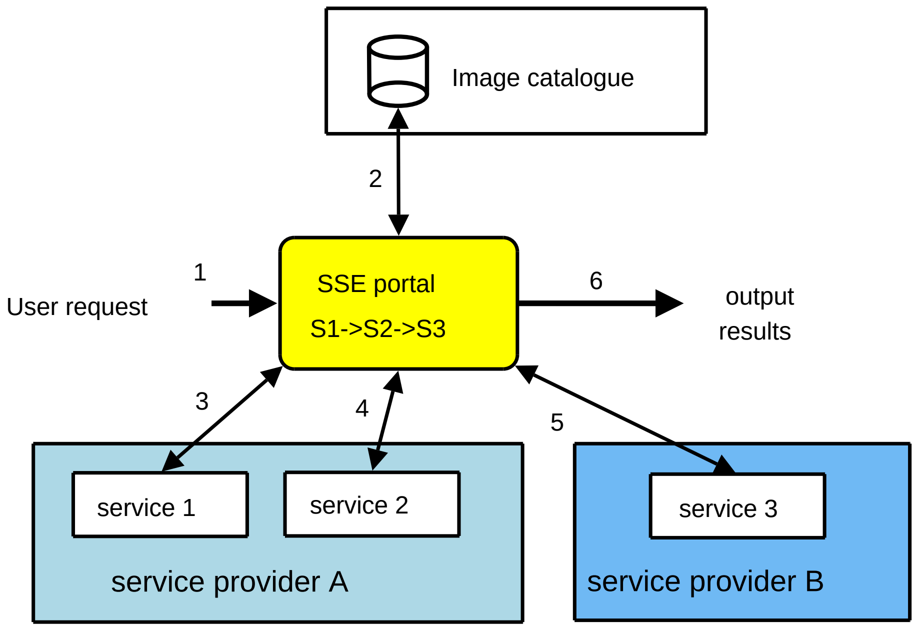


Figure 1.3: An overview of the SSE system

The user connects to the SSE portal and searches for a service matching its needs. Once a suitable service is found, the user can search through the appropriate image catalogue by means of a graphical interface. Once he have chosen an image, the elaborations on the chosen images can be ordered.

The SSE portal's workflow manager then handles all the interactions with each involved service provider system by means of the Simple Object Access Protocol (SOAP) open communication standard, invoking the services and retrieving the results of the elaborations; then it returns the relevant information on the elaboration outcome and result retrieval to the user.

This automatic workflow management system, the interactive graphical interfaces and the integration of users, service providers and data providers in a coherent supply chain are key factors to the user-friendliness of the SSE portal bringing transparency and ease of use to its complex integration mechanism.

The catalogue search can be done on any catalogue service integrated onto the SSE portal: the official ESA catalogue is the Earthnet OnLine Interactive (EOLI) catalogue providing access to multiple collections of image metadata from different satellites. This catalogue has an interface for portal integration and data provisioning and is run by ESRIN.

The SOA principles upon which ESA built the SSE system are clearly seen on its characteristics:

- a service provider builds and manages its services using its own fully controlled resources;
- the SSE acts as service broker registering published services into appropriate searchable categories;
- a service can be used directly or orchestrated into a complex workflow involving many services.

Registered Services on the SSE portal are classified into searchable categories providing to the user a virtual market place that relates them to value adding businesses. The services are also reusable blocks of arbitrary complexity ranging from small *basic services* which can be orchestrated into a workflow to provide large chained *complex services*.

Chapter 2

Technology, standards and tools

This chapter is an introductory description of the standards and technologies upon which the integration project relies and of the tools supporting them.

2.1 Base standards

This section provides an introduction to the founding standards adopted in the integration framework of the SSE portal. These standards have been developed or approved mainly by the Word Wide Web Consortium (W3C) and are largely adopted whenever interoperability is a key success factor.

2.1.1 XML

EXtensible Markup Language (XML) is a W3C recommendation which describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them[7]. An XML document has a tree-like structure consisting in a root element containing storage units called entities. An entity contains either parsed data (character data and markup tags) or unparsed data. The markup present in the parsed data is used to describe the document's storage layout and logical structure.

XML is designed to be straightforwardly usable over the Internet and to be easily used in a wide range of applications as well as to be clear and human-legible. Terseness is also of minimal importance in an XML document which shall also be easy to create. Its main focus is data description thus its not about doing something but only to structure, store and send information.

One key concept of XML is its extensibility, in fact, no markup tags are predefined by its specification and a Document Type Definition (DTD) or an XML Schema are used to define new tags for data description; an XML document must be well formed according to its specification and it can also be valid if it complies with the definitions in its associated schema.

An example of a valid XML document and its associated schema are presented in Listing 2.1 and 2.2 respectively.

Listing 2.1: An example of XML document

```xml
<?xml version="1.0"?>
<message
    xmlns="http://message.namespace"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="message.xsd">
<to>Alice</to>
<from>Bob</from>
<text>Here is my message</text>
</message>
```

In an XML document each open tag must either have a corresponding closing tag or be an empty tag and the elements must be properly nested. With an associated DTD or schema an XML document is completely self-descriptive.

Listing 2.2: Associated example of XML schema document

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://message.namespace"
xmlns="http://message.namespace"
elementFormDefault="qualified">

<xs:element name="message">
  <xs:complexType>
    <xs:sequence>
          <xs:element name="to" type="xs:string"/>
          <xs:element name="from" type="xs:string"/>
          <xs:element name="text" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

An XML schema, known as XML Schema Definition (XSD) is the XML version of the DTD which takes advantage of all the XML features and also adds support for data types. It constrains the contents of any associated XML document defining which elements are valid in the document and how to use them. Many basic date types are built-in in the XSD (like: string, integer, boolean and time), more on XSD can be found in its specification[8].

Being made of plain characters XML provides a way to exchange data between incompatible systems (by converting their data formats in a standardized XML document) since it is not dependent on those system's software nor hardware. Data storage can also take advantage of this independent behaviour by using a plain XML file as a fully portable data store.

2.1.2 XSL, XPath and XSLT

Since XML is a language focused on information holding, another language is needed to define how to present the information held by XML documents and to operate rule-based manipulation of these documents (namely, to transform them). EXtensible Stylesheet Language (XSL) is a family of W3C recommendations defining a language for the aforementioned purposes which plays a fundamental role in the SSE framework.

XSL was developed as an XML-based Stylesheet Language. Being a stylesheet language its main purpose was to define how information contained in the XML documents can be displayed. Since XML does not use predefined and well understood tags the stylesheet assumes fundamental importance in the document presentation process. XSL has grown to be more than a stylesheet language and it now consists of three main parts for defining not only XML presentation but also its transformation[9].

The three parts of XSL are the following:

XSL Transformations (XSLT) is a language for transforming an XML document into another, XML or other structured language, document.

XML Path Language (XPath) is an expression language used to access or refer to elements and attributes of an XML document.

XSL Formatting Objects is an XML vocabulary for specifying formatting semantics and is formally named XSL since it solves the former XSL purposes.

The parts mostly used in the SSE framework are the first two while the third has less relevance to this work and is not be presented here.

Since XPath is the language used by XSLT to access or refer to the desired informations in an XML document its knowledge is mandatory to understand XSLT. Here follows a brief description of this language and its structure.

XPath uses path expressions to address elements and attributes in the tree structure of an XML document. An XPath expression is evaluated to yield an object having one of four basic types (node-set, boolean, number or string). A location path is an important kind of expression which selects a set of nodes that is returned as its result. A general location path is in the form:

```
[/] step [/ step]*
```

Where the presence of an initial slash character indicates an absolute location path while its absence indicates a path relative to the current context node. The optional slash is followed by a mandatory path step which can be followed by zero or more additional steps divided by slashes. The form of each location step is as follows:

```
axis nodeTest [predicate]*
```

The initial node set, selected by a location step, consists of every node which is associated to the current context node by the relation specified by axis (i.e child, ancestor, self, ...) and has type and name specified by the nodeTest (i.e. book). The initial node set is then filtered by the optional predicates, in sequential order of appearance. The exact semantic of the predicates is axis-dependant, the default axis for location steps is *child*.

As an example consider the following location path:

```
/bookstore/book[price<25 and price>10]/title[@lang="it"]
```

This expression, when processed, initially selects the root bookstore element then all its child book element nodes filtering them to keep only those having a price element with a value between 10 and 25, then it selects all the title elements of these book elements and finally filters them returning as result a set of title elements having the attribute named lang with value "it".

Besides location paths, other expressions are defined in XPath and there are also built-in functions that can be used in expressions; more information can be found in the XPath specification document[10].

XSLT is a language for describing how a source XML document is transformed into another result document. The XML documents are modeled as node trees so, as foretold, the XPath expressions are used to address their parts. Picture 2.1 shows an overview of a transformation executed by a component called XSLT processor.

The processor takes an input XML document and builds its corresponding source tree then, for the transformation engine, XPath expressions defined in the input stylesheet are tested for matching parts of the source document to one or more templates. When a match is found, the engine transforms the matching part of the source tree into the result tree. The processor's output is then a document which can be an XML document or another type of structured document represented by the result tree.

Being itself an XML document it is a particular root element that declares it as an XSL style sheet, such a root element can be either *stylesheet* or *transform* qualified, as any of

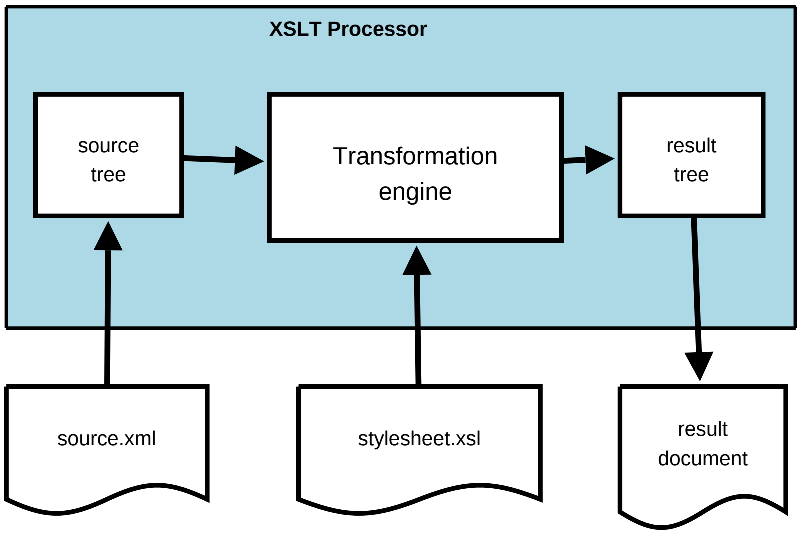


Figure 2.1: XSLT Processing

the stylesheet elements, with the XSLT namespace `http://www.w3.org/1999/XSL/Transform`[1]. The most relevant types of elements that can appear as top-level elements[2] in a stylesheet relating to this writing subject are:

xsl:import is used to add other stylesheets to the importing one ensuring a precedence policy in the template order.

xsl:param is used to pass parameters into a template or stylesheet being invoked.

xsl:template is used to define template rules.

The transformation processing model is defined as follows:

> A node is processed by finding all the template rules with patterns that match the node, and choosing the best among them; the chosen rule's template is then instantiated with the node as the current node and with the list of source nodes as the current node list. A template typically contains instructions that select an additional list of source nodes for processing. The process of matching, instantiation and selection is continued recursively until no new source nodes are selected for processing[9].

Node identification for template matching is done using Patterns which are a set of restricted location paths separated by the, or operator-equivalent, | character. Template

[1]In this subsection the *xsl* prefix is associated with this namespace but any XSL stylesheet can bound it to any prefix

[2]Top-level elements are child elements of the root element

rules application is invoked by means of the *xsl:apply-templates* element. An example template rule is provided in listing 2.3, the listed rule matches the *emph* elements and produces a bold HTML formatter then processes its immediate children.

Listing 2.3: An XSL template rule example

```
<xsl:template match="emph">
  <b>
    <xsl:apply-templates/>
  </b>
</xsl:template>
```

XSLT also provides elements to operate flow control selection in a transformation (*xsl:foreach, xsl:if, xsl:choose*), value selection (*xsl:value-of*) as well as some built-in functions; complete information on the aforementioned XSLT elements can be found in the language specification[9].

2.2 Technology

The main thechnologies adopted in the SSE systems are Web Services and process workflows for service orchestration, this section provides an introductory description to these technologies.

2.2.1 Web Services Architecture

XML web services are an important evolution of web-based technology; they are distributed server-side software components designed to interface to other software components enabling application development in the internet environment. Web services technology is based upon SOAP which in turn is an XML extension thus Web services are interoperable in a wide range of heterogeneous systems provided they support the SOAP protocol.

The Web Service Architecture (WSA)[11] is a W3C note aimed at providing a conceptual model and context to understand Web Services and the relations between the conceptual model's elements. It promotes the interoperability defining standardized compatible protocols neither restricting Web Services to a particular implementation nor to a specific use case. The WSA definition of a Web Service is:

> Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using

SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards[11].

In the WSA model Web Services are implemented by software components called agents designed by a service provider to provide some functionality to a requester agent. The message exchange between these agents are documented in a machine-processable "agreement" which provides description of the expected message contents and exchange patterns; this document is called Web Service Description and is written in the XML based Web Services Description Language (WSDL). The WSDL description defines only the service's interface, the semantics (meaning of information, usage conditions and consequences) are outside the description document scope and must be defined by other means.

The main structure of a WSDL document is presented in Listing 2.4 while a complete description can be found in its specification[12].

Listing 2.4: WSDL service description example

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://www.example.ns"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
   <!-- This section defines, using XML schema syntax, the
        data types used by the described service --> ...
</types>
<message>
   <!-- A message defines the data elements of an operation
         and consist of one or more message parts --> ...
</message>
<portType>
   <!-- A prtType describes a web service specifying which operations are available
       and the messages used by those operations --> ...
</portType>
<binding>
   <!-- A binding associate message format and
        transport protocol details to a defined port--> ...
</binding>
</definitions>
```

Four type of operations can be defined in the portType elements: one-way, request-response, solicit-response and notification each representing its respective interaction model.

Web service's messages are conveniently conveyed in a platform- and application-independent way by using the SOAP protocol. This XML-based protocol is used to encapsulate the message in a standardized way; the SOAP message structure is shown in listing 2.5, for a complete SOAP description refer to its specification[13].

Listing 2.5: SOAP message structure

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
     ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
       ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

In a SOAP message the mandatory *soap:Envelope* element encapsulates optional application specific message information in its *soap:Header* element, the message itself in its *soap:Body* element and fault related information like error code and description, if needed, in the *soap:Fault* element.

Both SOAP and WSDL can be extended to include other functionality like WS-Addressing for message delivery and correlation and PartnerLinks for service operation roles specification in a business process.

Service Oriented Architecture

A SOA is a specific kind of distributed system which can be viewed as a mesh of collaborating services each executing a well defined operation set which can be invoked over the network in a standard and interoperable way. Key components of a SOA are:

- Exchanged messages;
- Requester and provider agents;
- Open and standardized transport protocols;
- Component description and access policies.

The focus on messages instead of actions in a SOA and their standardization brings transparency in the architecture, a message intermediary like a firewall can easily inspect transit messages and act appropriately in a predictable way while a human trusted inspector can examine the messages and find out the involved services and requested actions.

The WSA is indeed a SOA since its focused on services and exchanged messages, based on shared, open standards and protocols and can be extended to provide means for requester and provider identification and interaction policy. Figure 2.2 shows the correlation between the technologies and standard involved in the WSA.

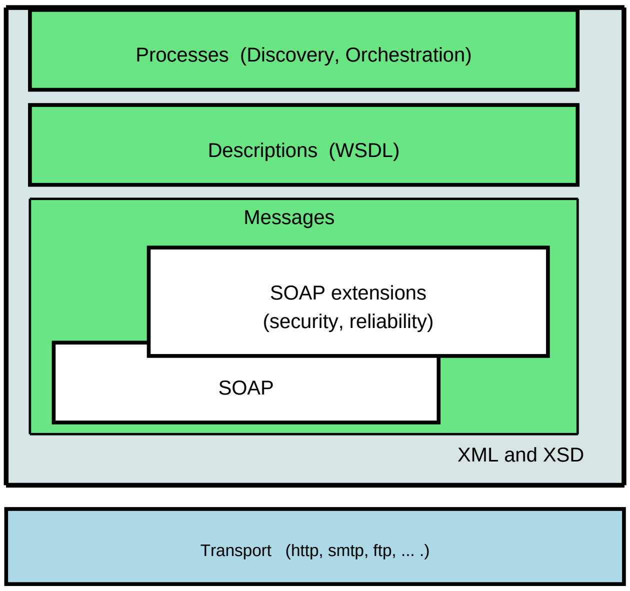


Figure 2.2: WSA related technologies

2.2.2 Services orchestration: BPEL

Web Services interactions can be described as an orchestration or as a choreography. The scopes of the two models are different since orchestration encompasses all parties and interactions in a global view of the system while choreography is focused on the point of view of a particular service and the interactions are based on events outside that service. Business Process Execution Language (BPEL) is an XML-based orchestration language used to model a business process; such a model can be either abstract (describing only the publicly observable behaviours) or executable (describing the detailed internal behaviour) by processing engine. A BPEL document is an XML document with a root *process* element qualified with the BPEL 1.0 namespace `http://schemas.xmlsoap.org/ws/2003/03/business-process/`.

The BPEL process orchestrates Web Services provided by partners, the role of each partner is specified by a *partnerLink* element in its WSDL service definition. The orchestration itself is a workflow built using various BPEL elements each providing specific functionality like flow control (while, flow, sequence) service communication (receive, reply, invoke), data variables (assign, variable), error control (throw, rethrow, fault handlers) and more. An exhaustive description of those elements is outside the introductory scope of this section and can be found in the BPEL specification[14]. The typical BPEL process used by the processes defined in this work follows the general structure shown in Listing

2.6, where some of the main BPEL elements can be observed.

Listing 2.6: Sample BPEL document structure

```
<process name="..."
  targetNamespace="..."
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/" ...>

  <partnerLinks>
    <partnerLink name="..." ... />
  </partnerLinks>

  <variables>
    <variable name="..." ... />
  </variables>

  <faultHandlers>
    <catch faultName="..." ... />
  </faultHandlers>

  <sequence>
    <receive partnerLink="..."  portType="..."
        operation="..."  variable="..."/>
    </receive>
    <flow> ... </flow>
    <reply partnerLink="..." portType="..."
       operation="..."  variable="..."/>
  </sequence>
</process>
```

The root *process* element defines the document as a BPEL process and contains all the other building elements:

partnerLinks contains all the definitions of the services participating in the described business process, these partners are named and assigned a role in the process;

variables contains the definitions of all the typed global variables identifiable by their assigned names;

faultHandlers defines the fault handling mechanics and can contain many fault specific *catch* elements and a general *catchAll* element. Other event handlers, catching non fault events, can also be defined in the similar unlisted *eventHandlers* element;

sequence this final element defines the main process control sequence and can contain any appropriate flow-control, assignment, service communication an like elements which composes the process orchestration logic.

The final sequence element of the listed process starts with a blocking receive element which puts the workflow in a message waiting state, when the specified message is received it is assigned to the specified variable and the process execution continues into the flow element[3]; when the flow is completed the reply element causes the response contained

[3]The flow element specifies a parallel execution of its contained activities and can also be replaced by a sequence element to specify a sequential execution model.

in the specified variable to be sent back to the requester thus defining the process to be synchronous; an asynchronous process is characterized by a final invoke element instead.

2.3 The Intecs TOOLBOX

Since the SSE system requires that all communications with service providers must be encapsulated in SOAP messages, there must be an endpoint at each service provider to receive the messages and correctly interpret them, this component (a SOAP interface) can be developed with any SOAP enabled programming language as well as built from scratch by the service provider. Intecs, partecipating in an ESA SSE development project, to facilitate the service provider in the Web Service development, has developed a JAVA servlet, which can be run on the Apache Tomcat servlet engine, that provides such an endpoint for the standard SSE operations: this servlet is called TOOLBOX. The TOOLBOX is built entirely with open source component and is released under the GPL licence.

Using a web interface, the service provider can configure the TOOLBOX to provide endpoints for his Web services, moreover, other utilities are provided by the TOOLBOX for the service provided to use for services debugging and managing, along with an integrated FTP server for data publishing. Figure 2.3 shows an overview of the TOOLBOX components and its placement as the interface between the SSE system and the service provider's system in the SSE framework.

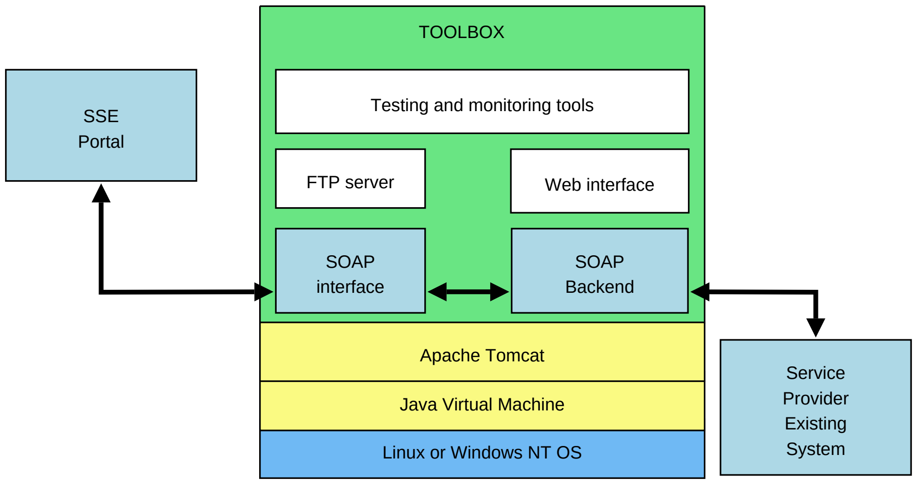


Figure 2.3: An overview of the Intecs TOOLBOX

The SOAP backend component must be developed and integrated with the TOOLBOX by the service provider as the actual connection between his existing system and the TOOLBOX. This backend is not standardized and must be integrated with the TOOLBOX using its scripting language.

The TOOLBOX scripting language is an XML based language interpreted and executed by the TOOLBOX as a programming language. This scripting language provides the most common programming language features (like flow control, variables, error handling) and adds its own elements to aid the service provider with the service integration (like XML manipulations, XPath expressions, and SSE based structures).

2.4 ESA interfaces

The SSE provides an open environment for Earth Observation and GIS services integration and its ICD defines the external interfaces between the SSE Portal and remote services hosted by the service providers[15].

the ICD provides an overview of the service integration process which can be resumed into the following points:

- an XML schema of the service's interface data model must be defined to provide a formal definition of the data exchanged in every service operation invocation;
- For basic services only: Install and configure the toolbox with the proper integrating mechanism. This step needs the development of the operational toolbox scripts and an eventual "glue" layer to enable the toolbox to manage the legacy service;
- Become a service provider making the proper request from the user profile on the SSE portal;
- For chained services only: Design and deploy the service workflow (implemented as a BPEL process). This can be accomplished using the Oracle BPEL designer Eclipse plugin to develop and the BPEL console provided on the SSE portal to deploy the workflow;
- Meet the service registration requirements by producing *the WSDL description* of the service and *the stylesheet* that defines how to present and process the input and output service data;
- Register and test the service using the service provider interface provided by the portal and once the service has passed all the desired operational tests it can be enabled setting its status to "enable".

The messages exchanged between SSE and the integrated services are treated as data (constrained to the ICD rules) and transferred according to the SOAP over HTTP or HTTPS protocol while the larger image data is transferred with File Transfer Protocol (FTP).

2.4.1 SSE Operations

Four operations that the service provider can expose are defined and will be accessible from the SSE portal pages and workflows once the service is published. The service operations are:

Request for Quotation (RFQ) this operation is meant to provide the user with service related data such as service availability, pricing or sample output data;

Order is the operation that allows sending an actual service order request; its interface must handle all the required service inputs and output results, either embedded in the messages or linked via FTP.

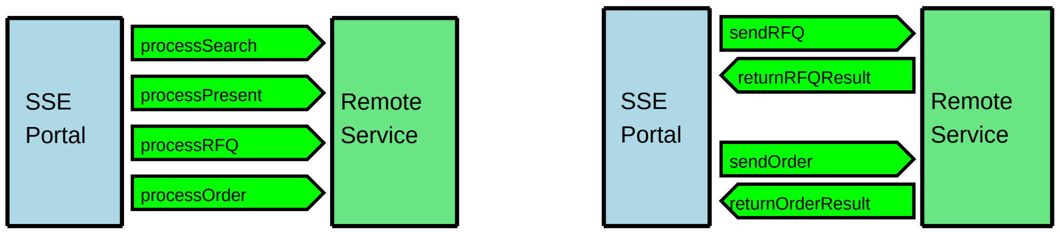
Search is the operation used to query image metadata from a remote catalogue matching some search criteria.

Present is based on a single *Search* result and allows requesting different levels of detailed information about the result from the corresponding catalogue.

None of the defined operations is mandatory for a given service but the invocation order is well defined and, supposed the availability of all operations, a Search will always come first and enables both the Present of one of its results or the Quotation request for a subsequent service Order. Since there is no mandatory operation the sequence can be reduced to a single Order request or to a catalogue consulting service providing only Search and Present operations.

The integrated services can implement operations taking variable amounts of time for completion thus two interaction modes are supported: synchronous and asynchronous operations.

The synchronous mode is available for each aforementioned operation and is suitable for operations taking a reasonably short amount of time to complete (approximately the waiting time expected for a web browser request) and consists of a single request/response message pair exchange while the asynchronous mode is available only for RFQ and Order operations and in this mode no assumption is made for the expected completion time which can be in the scale of days, weeks or longer; the latter model requires a caller callback endpoint for the service to communicate the elaboration results when it completes thus requiring two pairs of request/response messages, the first initiated by the SSE an the second by the elaboration service. The two operation modes and the message naming conventions are shown in Figure 2.4



(a) synchronous mode (b) asynchronous mode

Figure 2.4: SSE Operation Modes

2.4.2 Interface specifications

The SSE ICD states which message elements must be used by the service interface and provides recommendations for the relative payload types. A key feature of the message types is their extensibility: all of the RFQ and Order related request message types have in fact an extensible type for the service provider to add its service specific data types for both input and output. The service schema must import the schemas defining the types that it requires (some schemas, like the Geography Markup Language (GML) schema, are already imported by other schemas and need not to be imported again). Table 2.1 shows a list of the the available schemas and the relative files along with a brief description of their purpose[4].

The rest of this subsection describes the main elements in the SSE schema related to the Order operation, for an exhaustive description of all the types defined by the schemas presented in table 2.1 refer to the ICD[15].

The *commonInput* element is included in each request message and in the return messages for asynchronous operations. This element contains the *orderId* used to trace RFQ and Orders.

The *statusInfo* element is part of every response message and contains status infor-

[4]The schema files themselves are available in the ICD's Annex A[15] as well as online at: `http://services.eoportal.org/schemas/1.4/`

Schema	Files	Purpose
SSE Schema	mass.xsd or sse_common.xsd	Defines the types needed to support the SSE operations. The message types are named by the operation name and the InputMsg or OutputMsg postfix
AOI Schema	aoi.xsd	Defines the types to support the Area Of Interest (AOI) definition. This XML format is used on the SSE portal by the AOI tool to export the area of interest selected by the user (if needed)
Opengis GML Schemas	feature.xsd geometry.xsd xlinks.xsd	Defines GML types. It is imported by the AOI schema to define geometric areas, coordinates and box extents in a standard way
ESRIN EOLI ICD	eoli.xsd	Defines the types needed for the interface to the catalogues (Search and Present operations)
ESRIN Ordering	oi.xsd	Defines the ESRIN Ordering types which are used to obtain data from a catalogued data provider. Those are information concerning user identification, prices, payment and delivery method
GML Web Map Viewer Schema	serviceresult.xsd	Defines the format to display service results on a map
Regions Definition Schema	regions.xsd	Defines the expected format to import regions definitions specified by the service provider during the AOI configuration of a service. Those AOIs will be proposed to the user during the service order

Table 2.1: SSE message schemas

mation divided in its two contained elements: *statusId* containing a non negative integer value representing the "exit code"[5] of the requested operation and the optional *statusMsg* which should contain a user readable message explaining the operation status.

The Order-related types for synchronous mode are shown in Fig.2.5[15] and in Fig.2.6[15] for asynchronous mode; in these figures the thick border boxes highlights the aforementioned extensible types.

The synchronous Order operation *processOrder* (Fig.2.5) allows the submission of an order and the immediate reception of the relative results.

processOrder input message (see Fig.2.5(a)) contains the mandatory elements *commonInput* and the extensible *sendOrderInput*. Optional elements relative to the specific order are added by the SSE system if applicable; among those elements the last previous Search and RFQ related inputs and outputs can be found. The output message (Fig.2.5(b)) contains the extensible *getOrderOutput* with status information and optional result display enabling information.

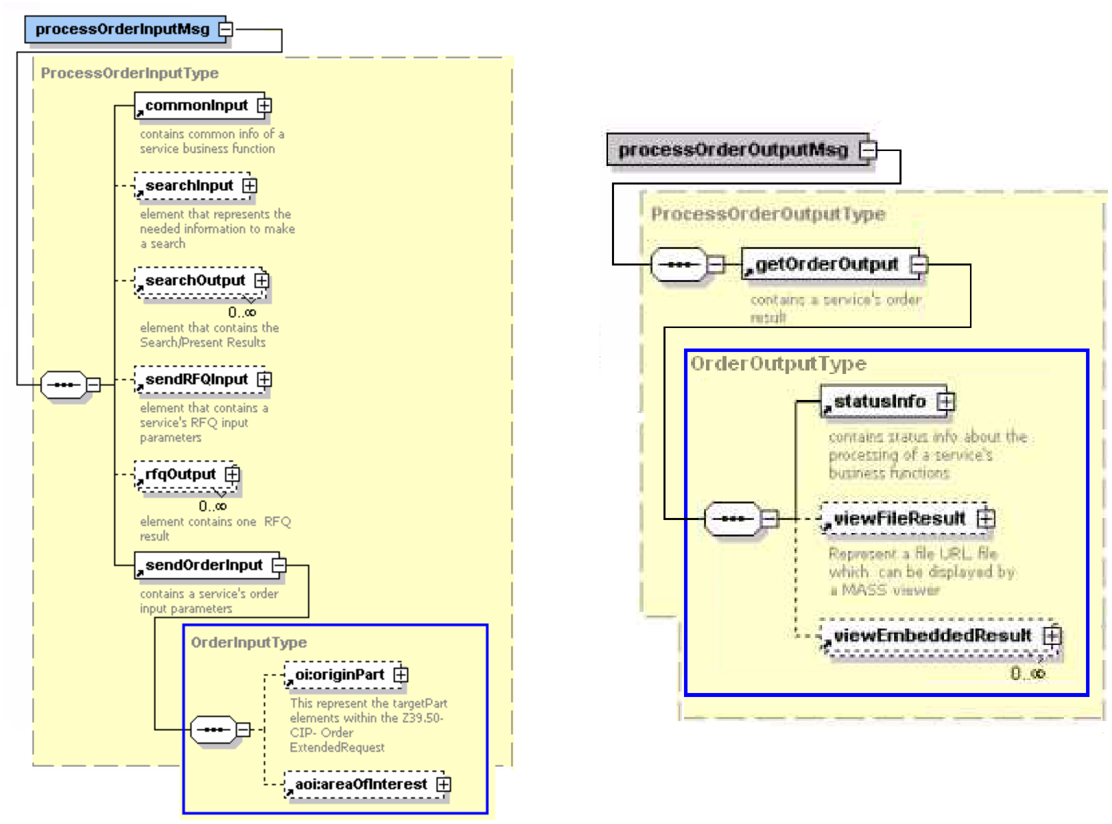
The asynchronous Order operation *sendOrder* (Fig.2.6) is used to submit an order allowing the service provider to differ the response which, once ready, will be sent to a specified callback endpoint.

Since this is the "two pair of messages" asynchronous mode of the Order operation the send (Fig.2.6(a)) and return (Fig.2.6(b)) input messages are almost the same as the input and output messages of the synchronous operation respectively. One noticeable difference is the presence in the *returnOrderResultInputMsg* message of the *commonInput* element which allows to trace this message back to its related *sendOrder* message. Both the send and return outputs contain the single mandatory *statusInfo* element used to report the status of the received message processing (e.g. accepted, rejected or faulted).

The presence of two different schema files for SSE schema (see table 2.1) is due to the two approaches to service schema definition, enabling the type extension, available for a service provider. These approaches are described below, the first imports the mass.xsl schema while the second imports the sse_common.xsd schema.

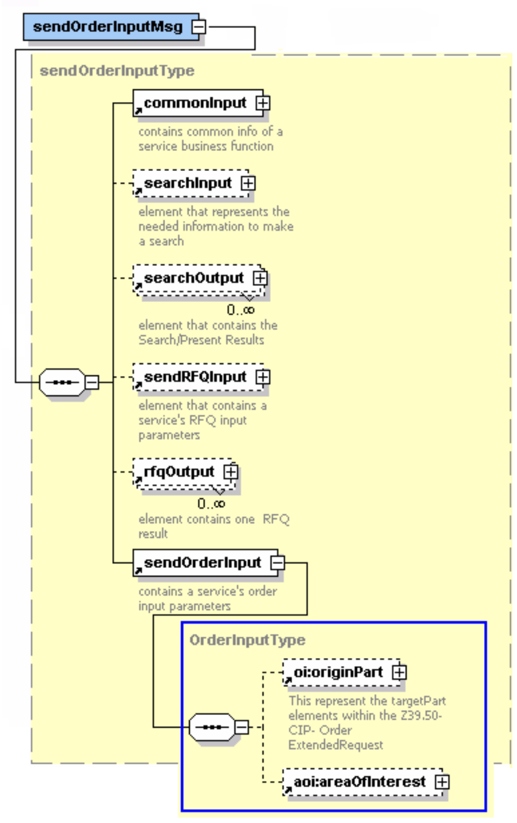
redefine this was the only approach available prior to ICD version 1.5. To define a schema by this approach a service provider imports the needed schemas and extends types using the *redefine* XML schema element. This approach enforces the expected

[5]The statusId codes are: 0 for success, 1-199 reserved by SSE and 200 and above for service specific codes.

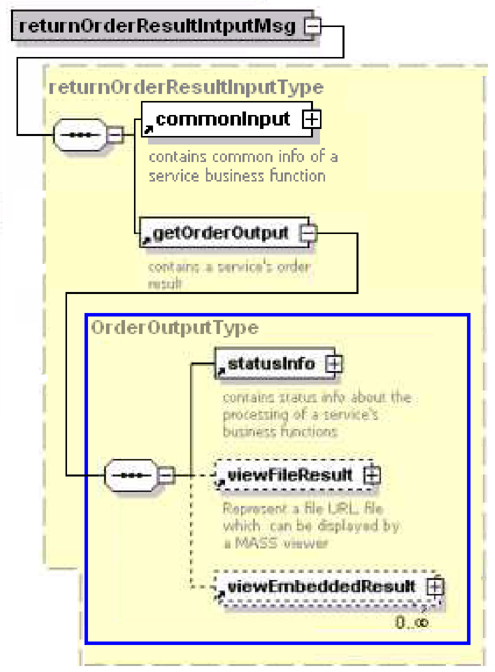

processOrderInputMsg
ProcessOrderInputType
commonInput
contains common info of a service business function
searchInput
element that represents the needed information to make a search
searchOutput
0..∞
element that contains the Search/Present Results
sendRFQInput
element that contains a service's RFQ input parameters
rfqOutput
0..∞
element contains one RFQ result
sendOrderInput
contains a service's order input parameters
OrderInputType
oi:originPart
This represent the targetPart elements within the Z39.50-CIP- Order ExtendedRequest
aoi:areaOfInterest
processOrderOutputMsg
ProcessOrderOutputType
getOrderOutput
contains a service's order result
OrderOutputType
statusInfo
contains status info about the processing of a service's business functions
viewFileResult
Represent a file URL file which can be displayed by a MASS viewer
viewEmbeddedResult
0..∞

(a) Input message element (b) Output message element

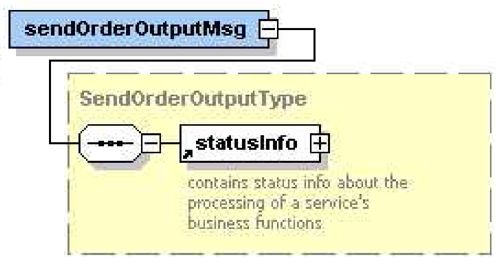
Figure 2.5: SSE synchronous order mode types


sendOrderInputMsg
sendOrderInputType
commonInput
contains common info of a service business function
searchInput
element that represents the needed information to make a search
searchOutput
0..∞
element that contains the Search/Present Results
sendRFQInput
element that contains a service's RFQ input parameters
rfqOutput
0..∞
element contains one RFQ result
sendOrderInput
contains a service's order input parameters
OrderInputType
oi:originPart
This represent the targetPart elements within the Z39.50-CIP- Order ExtendedRequest
aoi:areaOfInterest

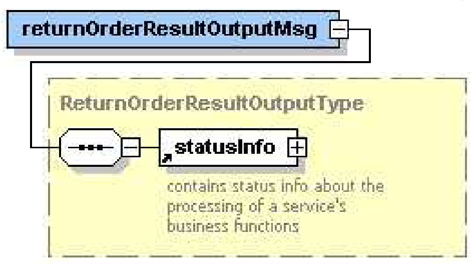
(a) Send Input message element


returnOrderResultIntputMsg
returnOrderResultInputType
commonInput
contains common info of a service business function
getOrderOutput
contains a service's order result
OrderOutputType
statusInfo
contains status info about the processing of a service's business functions
viewFileResult
Represent a file URL file which can be displayed by a MASS viewer
viewEmbeddedResult
0..∞

(b) Return Input message element


sendOrderOutputMsg
SendOrderOutputType
statusInfo
contains status info about the processing of a service's business functions

(c) Return Input message element


returnOrderResultOutputMsg
ReturnOrderResultOutputType
statusInfo
contains status info about the processing of a service's business functions

(d) Return Output message element

Figure 2.6: SSE asynchronous order mode types

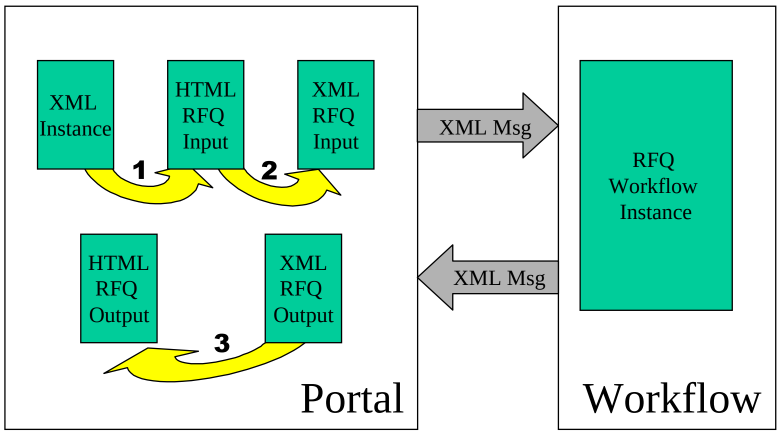


Figure 2.7: Overview of XSLT transformations

structure and minimum required information of messages but forces the use of the SSE schema namespace (mass).

import with this approach, introduced in ICD version 1.5 (and recommended for new services based on this version), the service provider imports the needed schemas and defines in its own namespace the types containing the extensible types and must reference the mandatory elements from the SSE common schema.

2.4.3 Graphical interfaces

The SSE system provides a standard way to collect the services input data and displays the output result information in a graphical environment (namely the user's browser) by means of the transformations defined in the service's stylesheet; The stylesheet is also used to generate the XML messages exchanged by the Portal and the Workflow manager enabling the user to collect the input data from the input form and display the output data to an output page. The aforementioned XSLT transformations, in the case of RFQ, are shown in Fig. 2.7[15] as yellow arrows.

Since the transformations are defined by the service provider in its service schema the graphical interface turns out to be fully customizable to the service provider needs. A stylesheet template is provided in the ICD (considering the import approach to schema definition) and can be conceptually divided into tree parts: a preamble containing the namespace definitions and the required template imports, a dispatcher used to apply the correct template based on the SSE system provided *part* parameter and the operation specific templates definitions.

The preamble most noticeable remark is the definition of the *sns* namespace to the service provider namespace which would be replaced by a fixed mass namespace in the other schema definition approach.

Listing 2.7 shows an excerpt of the stylesheet corresponding to the dispatcher part where the choose element is used to apply the suitable template for the current part parameter value. The template names referred here must be provided by the service provider in the third part of the stylesheet according to the operations implemented by the service; the listing displays only two order related choices since the others have almost the same structure. A complete list of the template which may be supplied by the service provider along with the corresponding *part* parameter value is provided in table 2.2.

Listing 2.7: Stylesheet dispatcher section

```
<xsl:param name="part"/>

<xsl:template match="/*">
<xsl:choose>
<!-- [...] other choices omitted -->
<!-- order operation templates -->
<xsl:when test="$part='sendOrderInputHTML'">
<xsl:apply-templates select="mass:sendOrderInput"/>
</xsl:when>
<xsl:when test="$part='sendOrderInputXML'">
<sendOrderInputMsg
 xmlns="http://www.mycompanyname.com/ws/mynamespace"
          :sns="http://www.mycompanyname.com/ws/mynamespace"
          :mass="http://www.esa.int/mass"
          :aoi="http://www.esa.int/xml/schemas/mass/aoifeatures"
          :oi="http://www.esa.int/oi">
<xsl:apply-templates select="mass:sendOrderInput" mode="XML"/>
</sendOrderInputMsg>
</xsl:when>
<!-- [...] Other choices omitted -->
</xsl:template>
```

There is one template call for each operation and mode thus the first shown template call is for displaying the order input data collection form and the second for the transformation of the collected data into the appropriate XML message. The corresponding template definition templates are shown in Listing 2.8 and are to be completed by the service provider.

Listing 2.8: Stylesheet service specific section

```
<!-- Template for the Order input information using HTML format -->
<xsl:template match="sendOrderInput">
<script language="JavaScript" type="text/javascript">
// Insert your Javascript code hereafter in order to validate input fields values
// Cdata used to avoid Java Script operators problems (like a<b)
          <![CDATA[
          {/* return true if check ok else false */}
          function checkMandatoryFields(form)
          ]]>
</script>
          <!--insert here HTML code to collect Order specific input data -->
</xsl:template>
```

```
<!-- Template for the order input information using XML format -->
<xsl:template match="sendOrderInput" mode="XML">
<mass:commonInput>
        <mass:orderId>
                <xsl:value-of select="orderId"/>
        </mass:orderId>
</mass:commonInput>
<sendOrderInput xmlns="http://www.mycompanyname.com/ws/mynamespace">
        <!--only if the AOI tool is used and Order is the first operation -->
        <xsl:copy-of select="AOI/*"/>
        <!-- insert XSLT statements to generate valid XML -->
</sendOrderInput>
</xsl:template>
```

The template definitions are responsible for building the appropriate service operations interfaces thus the first template shown must produce a valid part of an HTML document that will be merged into the SSE portal page for a service order operation while the second must build a valid XML document according to the specific service schema. A complete list of the templates definable by the service provider is contained in table 2.2.

part value	Template name	*mode* parameter
processPresentInputXML	mass:sendPresentInput	XML
getPresentOutputHTML	mass:processPresentOutputMsg	
sendSearchInputHTML	mass:sendSearchInput	
processSearchInputXML	mass:sendSearchInput	XML
getSearchOutputHTML	mass:multiCataloguesSearchOutputMsg	
sendRFQInputHTML	mass:sendRFQInput	
processRFQInputXML	mass:sendRFQInput	XML
sendRFQInputXML	mass:sendRFQInput	XML
getRFQOutputHTML	mass:getRFQOutput	
sendOrderInputHTML	mass:sendOrderInput	
sendOrderInputXML	mass:sendOrderInput	XML
processOrderInputXML	mass:sendOrderInput	XML
getOrderOutputHTML	sns:getOrderOutput	
	mass:getMaxNbrOfItemsTemplate	
	mass:rfqOutputHeader	
	sns:rfqOutput	

Table 2.2: Stylesheet part values and template names

Noticeable things about templates are:

- The SSE HTML form provides some fields usable by the stylesheet (an hidden orderId field and the submit and reset buttons);
- A validation mechanism is provided by the checkMandatoryFields JavaScript (JS) function, fully customizable by the service provider;
- Some predefined style classes are defined for the stylesheet text outputs (stylesheet-Text, stylesheetBoldText, stylesheetHeader).

Chapter 3

SOIL MAPPER integration

The SOIL MAPPER classification algorithm described in Section 1.3 has been implemented by MEEO as an IDL program, compiled into binary form, executable by an ENVI-IDL enabled machine. A dedicated Linux Fedora Core 6 system has been configured with the required software to host this implementation (the legacy service). This legacy service cannot be directly published onto the SSE Portal since it lacks of a SOAP interface. The TOOLBOX has been installed and configured on the host system to build the required SOAP interface to the service, the SOAP backend has been implemented by means of Linux shell scrips linked to the TOOLBOX using its dedicated scripting language.

The classification service must be applied to both user provided images and EOLI catalogued images. To meet this requirement, according to the SOA principles, only one FTP based Web Service has been built and then it has been orchestrated into BPEL workflows to provide EOLI catalogue support. Thus a user can use the FTP service alone to process its own images or the chained service to perform a preliminary EOLI catalogue search and then process an image selected from the catalogue. This chapter describes the Web Service development process and its integration onto the SSE system.

3.1 System Overview

Figure 3.1 provides an overview of the complete integrated system: this figure shows all the primary system components and each component is associated to its related XML-based document types. The XML-based documents are the configuration or programming logic for the components and have been developed in the integration project subject of this thesis.

The central coordinating component for all services is the Process Manager (PM), an Oracle BPEL execution engine, which keeps trace of all instances of each service's operation

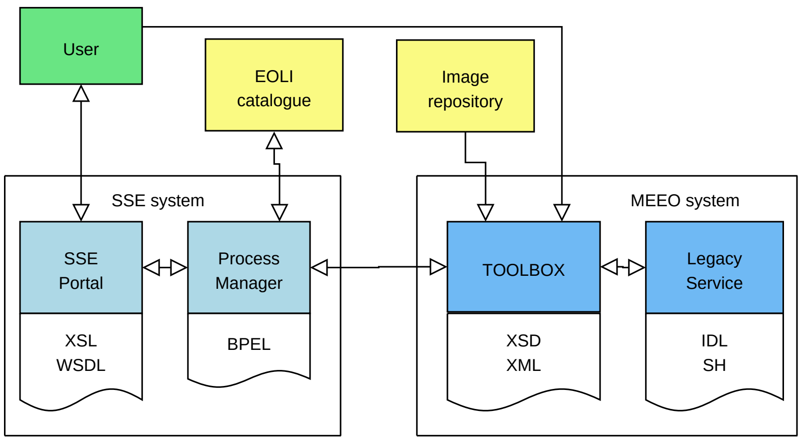


Figure 3.1: Overview of the integrated system

logic; even the most basic service interaction (i.e. a basic service order request and response scheme) is in fact described by a BPEL process and managed by this coordinator. The Process Manager uses BPEL documents (compiled into deployment suitcases) as its input for the service coordination.

The SSE Portal is therefore the front end interface to the PM component which provides the user with all the interfaces to enable and ease his service access: graphical input forms, graphical output result displaying and order tracing facilities. The graphical interface for AOI selection is always like the same (although configurable by the service provided) but each service will have its own input and output data, so the related graphical interfaces need to be customized to fit the specific service schema; this personalization is done building a template for each operation step in the service XSL stylesheet. The WSDL description of each service is also published by the SSE Portal for other service workflows to link it in chained services.

The EOLI catalogue holds the metadata for its catalogued images. Image metadata contains information about an image (geograpgic coverage area, acquisition time, ...) that can be matched to search constraints in order to find images with the desired carachteristics. Data required for the Present operation (like image thumbnails) can also be found within the catalogue data.

Formal SOAP interface definitions are provided for Search and Present catalogue operations, allowing the orchestration of catalogue services into corresponding Search and Present workflows which define the communication scheme among the EOLI catatlogue and the SSE PM. The interface to the EOLI catalogue is called EOLI-XML and is defined by its ICD document[16].

Catalogued Images are stored in the Image repository identified by a *collectionId* value which should be unique within a catalogue service. By using the metadata retrieved from the catalogue, the service provider system can access the correct Image repository to download the image to be processed.

The MEEO Service Provider system, for the purposes of this writing, consists of a single host machine with the following two main components installed:

TOOLBOX this component is an installation of the Intecs TOOLBOX specifically configured with the SM service schema and programmed to behave as the described sysyem requires by means of its scripting language.

Legacy Service this component consists of the full ENVI-IDL installation required for SM classifier execution, a launcher bash shell script (actually implementing the SOAP backend interface between the TOOLBOX and the classifier) and the SM binary program itself.

Communications among the system components are based on the following protocols:

HTTP is used to convey graphical service interfaces (implemented as web pages) to the user by the SSE Portal.

FTP is used to transfer files from the Image repository to the MEEO system and to transfer elaboration result to the user.

SOAP is used for all the other communications, with the exception of the TOOLBOX-Legacy service link which is non standardized since it is implementation dependent.

The rest of this chapter details these components and their configuration.

3.2 The Legacy Service

The service to be exposed as a Web Service consists of two IDL programs, the first is the radiometric calibration of the source image while the second is the two-step classificator which also generates the output image. This section describes the service Input and Output (I/O) model and the backend shell script, the SM architecture is described in section 1.3.1 and details about the machine hosting the service can be found in appendix A.

3.2.1 Service I/O

The SM system is designed to elaborate a source image to produce an output image consisting in the original image with an additional layer holding the classification information, thus primary inputs for SM are the source image location and the desired classification level; the implementation has an input format identification routine and two possible output image formats so the desired output format must also be specified as input. The service output consists on the classified image.

Both the radiometric calibrator and the classifier are designed to work in the same directory which contains the data to be processed, thus the calibration has no input parameters and SM accepts on its command line the classification level and output format parameters in the following form:

classification level is the first command line parameter and can contain one of the following three values: 0 for Large set, 1 for Medium set and 2 for Small set of categories, referred to as Full, Reduced and Limited number of classes in the following sections[1];

output format The output format is the second command line parameter and can have value 0 for ENVI or 1 for GeoTiff output. format.

3.2.2 Backend script

An automation of the legacy service must: download the source image archive on the service host, unzip the archived data, place data and IDL programs in the same directory (working directory), launch the radiometric calibration program and, once it terminates, launch the classificator with correct input parameters. This automation, needed for the Web Service development, has been implemented by a Linux shell script; an overview of the automation is presented in Figure 3.2.

In the overview figure dotted lines represent control paths (invocation of components and directory manipulation), solid thick lines are the data path and solid thin lines are parameters path.

The launcher script implements the backend required for the service to be used by the TOOLBOX, namely, exposed as a Web Service. The archive download activity is deemed

[1]The SM classification level nomenclature has changed during the course of service integration. For this reason the service described in this thesis and published on the SSE Portal uses a classification level nomenclature differing from the one adopted by SM at the time of writing and described in section 1.3

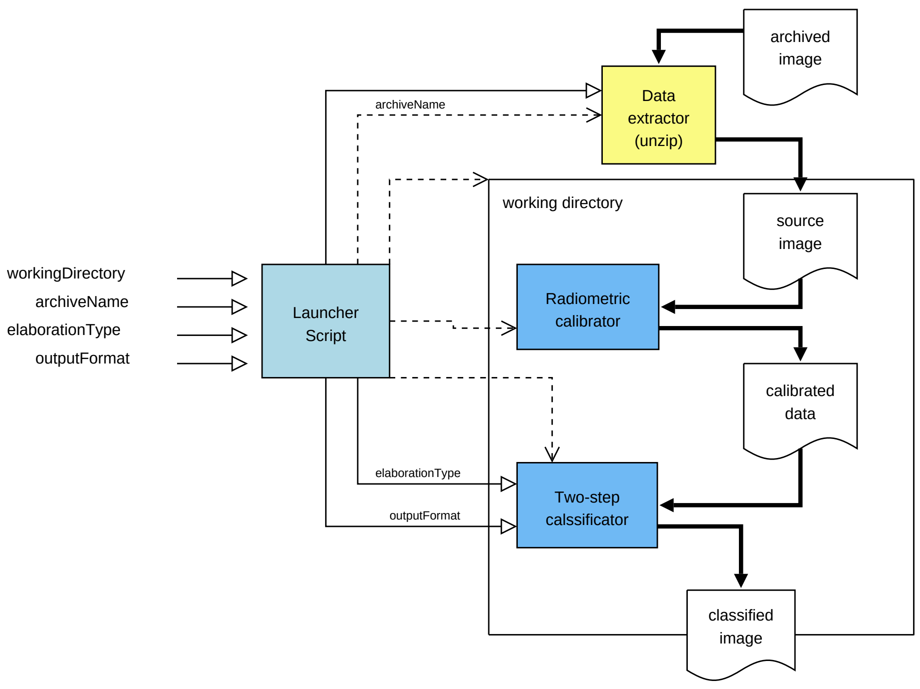


Figure 3.2: Overview of the SOAP backend shell script

to the TOOLBOX while an elaboration outcome detection mechanism is implemented directly in the backend script which communicates the outcome to the TOOLBOX using a text file named *outcome*, the presence of this file also indicates elaboration termination, either it was successful or not. Listing 3.1 is the part of the script determining elaboration outcome, the full launcher script is provided in Listing B.5.

Listing 3.1: Launcher script: elaboration outcome determination

```
#Processing outcome determination
        results=`ls | grep SpCl`
        if [ -z "$results" ];
        then
          echo "failure" > $tempath/outcome

        else
          echo "success" > $tempath/outcome
        fi
#$
```

All output files generated by SM contain the string “SpCl” in their name, outcome is determined by the presence of at least one file containing that string in its name. This part is executed after waiting for the last IDL batch to complete so either the file is found (the elaboration completed successfully) or it is not present (something was wrong). In both cases the log files generated by the IDL programs are moved to the results directory and then handled by the TOOLBOX.

The working directory is set up by extracting into it the source image; the IDL binary executables are not launched directly but each one is called by the respective IDL batch script, the calibrator script is shown in Listing 3.2 and the classification script can be found in Listing 3.3, these scripts are also copied into the working directory from a predefined repository directory on the hosting machine. The working directory is deleted once the elaboration results have been put into the results directory.

Listing 3.2: The IDL calibrator launcher
```
/usr/local/bin/idl <<!here
PREF_SET, 'IDL_PATH', '$programsPath:<IDL_DEFAULT>', /COMMIT
etm_calib_prom
exit
```

Listing 3.3: The IDL classificator launcher
```
/usr/local/bin/idl <<!here
PREF_SET, 'IDL_PATH', '$programsPath:<IDL_DEFAULT>', /COMMIT
etm_spcl_v77_prom ,$1,$2
exit
```

The developed script enables full automation of the service execution and has a formally defined interface for its input parameters thus it can be integrated with the TOOLBOX to act as the SOAP backend for the SM service. These parameters are also to be considered for the service schema definition and a way to deduct or convert them from user input must be implemented in the TOOLBOX operation scripts.

3.3 TOOLBOX scripting

The toolbox is configured to perform its interface role by means of a web interface and the underlying logic is programmed using its XML-based scripting language; the service schema is also needed for a correct message validation. This section describes the service schema and the scripts developed for the SM integration.

3.3.1 Service schema

The service schema must define the service input and output interface types, the I/O model described in section 3.2.1 needs to be adapted to the working logic of the Web Service in order to build the schema.

The source file is not available on the service hosting machine so there must be a way to retrieve it. The FTP part of the service name comes from the method chosen for image retrieval: the source image location is given by means of a full FTP address in the form:

`ftp://userName:passWord@ftpHost/pathToFile/fileName`

The full FTP address has been chosen as the service input because it is also the output proposed for EO catalogue services as result for an image order with *FTP pull* retrieval method[17]. Other input parameters are acquired as strings.

The service output needs to be accessible to the user who requested the elaboration, the chosen publication method is a temporary FTP access through the TOOLBOX integrated FTP server, thus access credentials (username and password), FTP address of the result and expiration time for data retrieval must be provided to the user as service outputs.

The schema definition has been done using the redefine approach since the include approach introduced by version 1.5 of the SSE ICD was not available at schema definition time. Figure 3.3 is a graphical representation of the service I/O interface, the corresponding schema can be found in appendix B, Listing B.1.

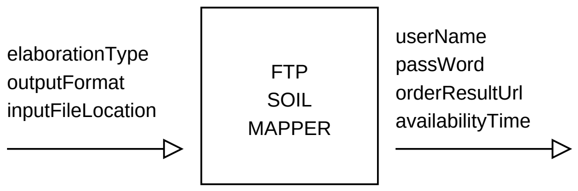


Figure 3.3: Service Input and Output schema

3.3.2 Order scripts

SM handles only the asynchronous order operation. To define an asynchronous operation within the TOOLBOX, three scripts must be developed. The TOOLBOX automatically calls the appropriate script for each current elaboration depending on its state, the scopes of these scripts are:

first script it is executed upon reception of a valid request message, this script is responsible for gathering the needed resources and start the elaboration;

second script it is repeatedly executed with a predefined time interval between each execution and is responsible for signaling the readiness of the resources needed to build a service response. This script returs a boolean value;

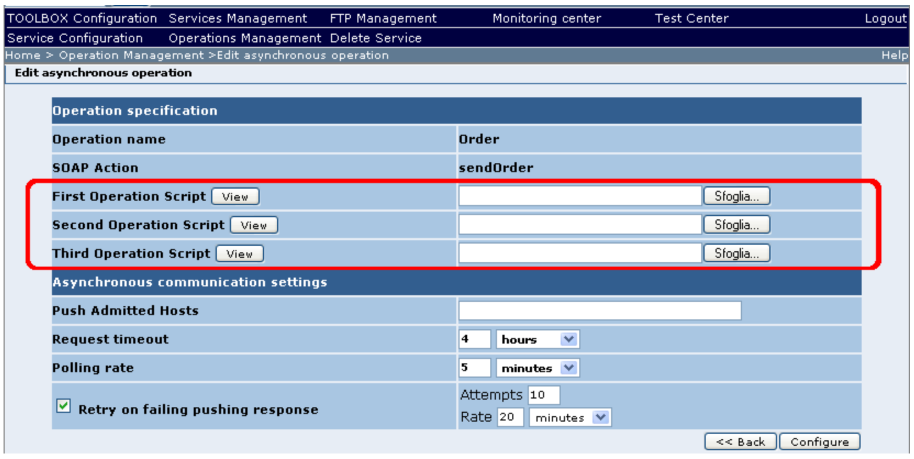
third script it is executed once the second script signals the readiness of the needed resources and is responsible for building the XML contents of the message that will be sent to the SSE system in response for the corresponding request.

An overview of the overall logic implemented by the scripts developed for the FTP SOIL MAPPER service order follows:

- Two variables (errorRaised and errorMsg) are used to notify execution status between the scripts, this is possible due to the persistence system implemented by the TOOLBOX which ensures that the values of some kinds of variables (boolean and string being among them) are persisted between the scripts during their execution.
- Some variables are used for script portability from Linux to windows systems (for testing purposes), these variables define the working directories for the actual service installation along with the appropriate slash character.
- The first script (ftpOrder1.xml) starts setting the elaboration status variables to " success" values and the portability variables to its relative path values. The input parameters are then extracted from the incoming request and the FTP address is divided into its building parts which are placed into variables later used to download the source image. A working directory for the order elaboration is then created based on the *orderId* value and the source image downloaded into that directory with fault control. If the download completes successfully the scripts starts the image elaboration calling the launcher shell script and returns. This script is listed in Listing B.2.
- The second script checks if an error has been raised or for elaboration completion, if none of this conditions are true it returns false to be executed again after the predefined time elapses, otherwise it returns true enabling execution of the third script. This script is listed in Listing B.3.
- The third script checks for the presence of an error raised in the first script and, if presents, builds an error message and sends it back to the SSE system. If there is no raised error the content of the *outcome* file is examined. This file contains a single word which can be either " success" or "failure" depending on the elaboration outcome. If the elaboration succeeded an account with username equal to the *orderId* and a 6 letters long random password, which will expire in three days, is created on the TOOLBOX FTP server enabling the retrieval of the elaboration results, a response message with the access information is then sent back to the SSE system. If the elaboration failed no account is created and an appropriate error message is

sent back to the SSE system. This script is listed in Listing B.4.

The service schema and the order scripts have been integrated to the Intecs TOOLBOX using its web interface, the scripts configuration interface is shown in Figure 3.4.



Operation specification	
Operation name	Order
SOAP Action	sendOrder
First Operation Script View	Sfoglia...
Second Operation Script View	Sfoglia...
Third Operation Script View	Sfoglia...
Asynchronous communication settings	
Push Admitted Hosts	
Request timeout	4 hours
Polling rate	5 minutes
✓ Retry on failing pushing response	Attempts 10 Rate 20 minutes

Figure 3.4: Order configuration web interface

Figure 3.4 evidences the requirement of the three aforementioned scripts to correctly configure the asynchronous order operation with a red box. The *Polling rate* parameter is the amount of time elapsing between two executions of the second script.

3.4 BPEL processes

BPEL processes are used by the Process Manager not only to orchestrate several services but also to manage each operation of even a single basic service. The user output, collected by the interfaces described in section 3.5, is sent as input to the corresponding operation's workflow and the results are sent back to the SSE Portal for the visualization to the user.

As stated in section 3.3.2, the FTP version of SM service handles only the order operation but EOLI catalogue operations (search and present) must also be supported by the service while maintaining the user ability to request elaboration of images already in his/her possession; to meet these needs the service is published twice: an EOLI enabled version and a basic FTP version.

The basic service remains the same and the differences are in the operation management (the BPEL workflows associated with them) and in the respective stylesheet of the two versions. This section deals with the BPEL processes developed for the service's versions and the next is about the stylesheets.

The SSE service registration interface enables the selection of default workflows for the service's operations which can be used if the service to be registered is a basic service, thus only the basic FTP version uses the default Order workflow while the EOLI enabled version uses custom workflows developed using the templates provided by the SSE Portal as their basis.

The *defaultOrder* workflow used by the basic FTP service sends the user input to the MEEO toolbox and awaits for the service results.

3.4.1 Custom BPEL workflows

The SSE system provides a defaultEOLI workflow for services implementing their own EOLI-XML compliant interface. This default workflow is not suitable for the SM service since, for this thesis context project scope, the images have to be retrieved from the official ESA catalogue. SSE provides a template BPEL process for each operation for the service provider to use as the basis for customization.

The *processSearchEOLI* and *processPresentEOLI* template workflows have been modified to send the catalogue related requests to the ESA catalogue instead of the service provider; to do these modifications the source BPEL process template has been edited with the Oracle BPEL Designer[2]. The ESA catalogue service is included in the process as a partner link by a modified versions of the WSDL descriptions provided as an annex to document [17]. The default workflows performs the conversion between the SSE message format and the EOLI format using XSLT transformations; the search workflow also makes AOI format conversion and generates the GML string to display results location on the SSE map. The workflows has then been built into a deployable suitcase and deployed on the SSE system using its Oracle BPEL console.

The order workflow for the SM service has been built to chain the ESA catalogue (data provider) with the basic FTP service. This workflow takes the selected result of the last Search operation as input (along with classification level and output format); derives from the Search result the archive name corresponding to the selected image and composes it into a complete FTP address by adding predefined access credentials and the Internet address of the repository hosting the archive. The composed FTP address is then used to

[2]Oracle BPEL Designer is a plugin for the Eclipse Integrated Development Environment (IDE) developed and distributed free of charge by Oracle, this plugin, however, at the time of writing has been devolved to the Eclipse project and is no longer maintained by Oracle. The officially suggested tool for BPEL editing is now the Oracle JDeveloper suite available on the oracle website (`http://www.oracle.com/technology/products/jdev/index.html`).

invoke the basic FTP SOIL MAPPER service along with the other service inputs. Once the basic service returns its result the workflow eventually sends it to the SSE Portal and terminates.

3.5 Service interfaces

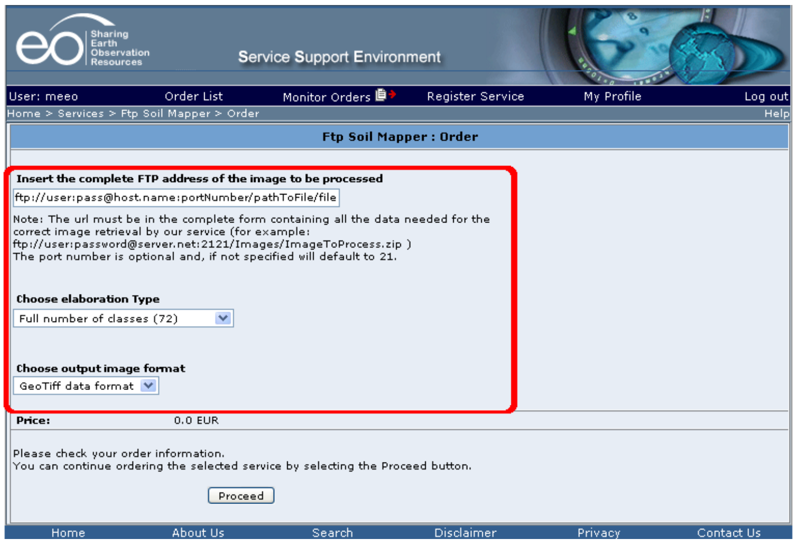
The service's graphical interface for the user is defined in its XSL stylesheet, this stylesheet also contains the transformations required to build the XML messages sent to the PM component as workflow inputs and also to convert the information in the returned messages into an user friendly form. Here follows the description of the stylesheets developed for the two service versions: FTP SOIL MAPPER service which allows elaboration of image already in the user's possession (provided those are accessible using the FTP protocol) and EOLI SOIL MAPPER SERVICE which integrates the EOLI catalogue allowing the user to perform classification of images selected from this integrated catalogue.

3.5.1 FTP SOIL MAPPER service stylesheet

This basic service has been published on the SSE Portal to allow the processing of an image accessible via an user specified FTP address. The only supported SSE operation is *Order* therefore the interface for this service is a graphical front end (namely an HyperText Markup Language (HTML) form) which allows the user to input the image address, the desired classification level and output format.

Besides the input interface template, the stylesheet also provides the XML mode input transformation and the result display enabling templates required by the SSE Portal (see 2.4.3). The service stylesheet is shown in Listing B.6 on page 65. The first template processed by the SSE Portal is the input interface template: *mass:sendOrderInput*. This template produces an HTML table which holds the input fields available to the user for data entry along with a JavaScript function validating the specified FTP address to ensure its completeness. Figure 3.5 shows the transformation result (highlighted by a thick red block) embedded into the SSE order page.

Each input field generated by the first template has a name and that name is used in the second template to retrieve the user input. The second template, called by the SSE Portal when the user confirms the order, builds the XML message that will be sent by the Portal to the PM component as the order workflow input. This template is named as the previous one but has the *mode=XML* attribute; this emphasises template XML-related


Sharing Earth Observation Resources
Service Support Environment
User: meeo
Order List
Monitor Orders
Register Service
My Profile
Log out
Home > Services > Ftp Soil Mapper > Order
Help

Ftp Soil Mapper : Order

Insert the complete FTP address of the image to be processed

ftp://user:pass@host.name:portNumber/pathToFile/file

Note: The url must be in the complete form containing all the data needed for the correct image retrieval by our service (for example: ftp://user:password@server.net:2121/Images/ImageToProcess.zip)
The port number is optional and, if not specified will default to 21.

Choose elaboration Type

Full number of classes (72)

Choose output image format

GeoTiff data format

Price: 0.0 EUR

Please check your order information.
You can continue ordering the selected service by selecting the Proceed button.

Figure 3.5: Input interface transformation result

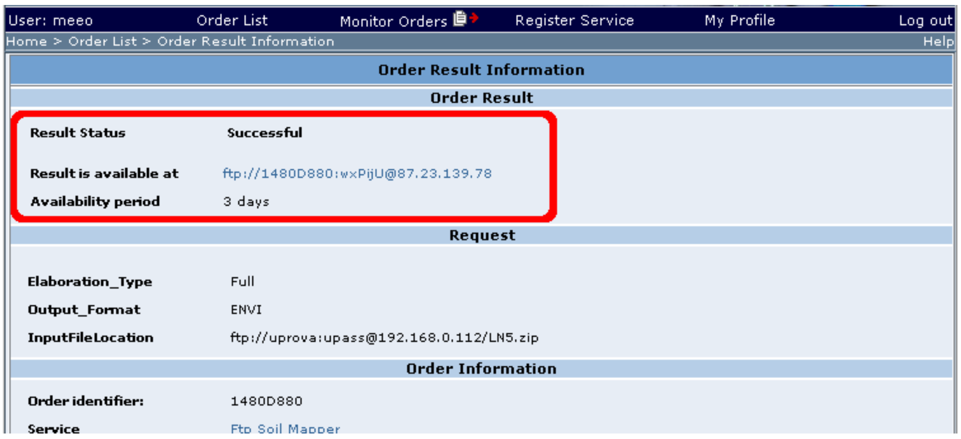
purpose. The template is shown in Listing 3.4.

Listing 3.4: The order input template with XML mode

```
<xsl:template match="mass:sendOrderInput" mode="XML">
  <mass:commonInput>
    <mass:orderId>
      <xsl:value-of select="orderId"/>
    </mass:orderId>
  </mass:commonInput>
  <mass:sendOrderInput>
    <mass:elaborationType>
      <xsl:value-of select="elaborationType"/>
    </mass:elaborationType>
    <mass:outputFormat>
      <xsl:value-of select="outputFormat"/>
    </mass:outputFormat>
    <mass:inputFileLocation>
      <xsl:value-of select="inputFtpAddress"/>
    </mass:inputFileLocation>
  </mass:sendOrderInput>
</xsl:template>
```

The output of this template is an XML document part which conforms to the *mass:sendOrderInputType*(see Figure 2.5 on page 25) where every *xsl:value-of* element has been replaced with the appropriate input field value. The template output will be embedded into the outgoing message as child node of the default *sendOrderInputMsg* element. The mandatory *orderId* child of the common input is provided by the *orderId* hidden input field provided by the SSE Portal.

When the service result is pushed back to the SSE system, the PM component sends it back to the SSE Portal which processes it calling the third order related template: *mass:getOrderOutput*. This template generates the HTML document part used to display elaboration results to the user on the service results page. An example (for a successful order) of the transformation result output, highlighted by a thick red border, is shown on the service result page in Figure 3.6.



Order Result Information

Order Result

Result Status	**Successful**
Result is available at	ftp://1480D880:wxPijU@87.23.139.78
Availability period	3 days

Request

Elaboration_Type	Full
Output_Format	ENVI
InputFileLocation	ftp://uprova:upass@192.168.0.112/LN5.zip

Order Information

Order identifier:	1480D880
Service	Ftp Soil Mapper

Figure 3.6: Output interface transformation result

The user can simply click on the result FTP address hyperlink to retrieve the elaboration results. In case the elaboration failed the user is warned of the fault instead.

3.5.2 EOLI SOIL MAPPER service stylesheet

The stylesheet for the EOLI enabled version of the service works much like the one for the FTP version: it provides additional templates for the Search and Present operations related inputs and XML message generation. Since the FTP address of the source image is provided by the EOLI catalogue, the order input interface template does not generate the corresponding input field of the FTP service version, thus only classification level and output format are requested to the user.

3.6 Publishing and testing

Both versions of the SM service have been published on the SSE Portal in testing mode using the service registration page. These services will be turned on operating mode (i.e. become accessible to the end users) after their delivery and setup on the ESA servers.

The basic FTP version of the SM service has been thoroughly tested using both the TOOLBOX test center and the SSE Portal. The test center has proven to be a very useful tool during service develpment. Its integrated *push server* allows testing locally even asynchronous operations that require a response publication enpoint to be provided by the caller. The *push server* provides not only this required endpoint, but also some related monitoring facilities for incoming response messages tracing and their presentation with a graphical interface.

During the test phase the FTP version has been tested with both local and remote FTP servers. Stress tests have also been performed on the service, these tests revealed a flaw probably lying in the TOOLBOX queue management system: when an error is successfully catched in the first script the TOOLBOX sometimes does not proceed on calling the second script and the service execution hangs. This does not allow the next request to exit the queue for execution thus blocking the queue. When the above situation occours even cancelling the pending elaboration, it doesn't unblock the queue thus forcing a restart of the service host. On the other hand, when there are no queued elaborations, the fault is correctly catched and handled by the third script, allowing correct eleboration termination. This is a major issue wich will hopefully be addressed in the upcoming new version of the TOOLBOX.

The EOLI version of the service has been tested online using the SSE system directly since its primary implementation resides in the BPEL workflows. Different search and present requests have been made using the service interface obtaining correct results in every case. For the image archive retreval an FPT repository has been built only for testing purpose. Once the services will be delivered to ESA, the official repository, along with its correct acces credentials, will be linked by changing the parameters hardcoded in the order BPEL workflow.

Chapter 4

Conclusions and further work

4.1 Conclusions

The SSE system allows the integration of services provided by service providers and data providers in an open environment which facilitates their collaboration; in the user's perspective SSE provides a catalogue of EO related value adding services with a standardized access interface (namely web pages) and facilities for service finding, access and order tracking. SSE-related research and development started in year 2003, at the time of writing the SSE is in its operational phase and service providers are encouraged to integrate their services in an effort to validate the system and contribute to the expansion of the EO market. The integration into the SSE system of the SOIL MAPPER automated soil classification system has been the subject of this thesis.

The SSE system has a formal interface specification at the service provider disposal and supporting tools are available to facilitate service integration. Nevertheless a service provider still needs good knowledge of the involved XML related standards. If a chained service has to be integrated in the system the orchestration logic, defined using BPEL processes, imposes also a good knowledge of the BPEL language and its related development tools.

Chapter 3 describes the realization of the Web Service interface to the SM system and the work done to integrate this service into the SSE system using the technologies described in chapter 2.

The Web Service related to the SM system has been also used in the context of an ESA system (the KEI project) aimed at developing an EO knowledge-driven information mining technique. The integration of the SM service makes available to ESA its innovative unsupervised Landsat classification functionality which can be used to build part of this knowledge-based system; the developed service, in fact, will be delivered to ESA for setup onto a dedicated server configured like the machine described in this work. The service will then be available to authorized users for research and non commercial purposes.

The SSE is still evolving and the schemas, tools and systems are continuously fine tuned in order to solve any problems emerging during services integration, development and utilization. The Intecs TOOLBOX is also validated by service providers utilization, they contribute identifying operational problems or suggesting needed fixing to obtain a reliable and stable tool which facilitates integration of many other services onto the SSE.

Integration onto the SSE system forces the service provider to entirely automate the service providing system. This service automation along with the automated management of the related operations can be an incentive to the service provider to apply the same management principles and open technologies to its entire business management system, further spreading the SOA model.

4.2 Further work

Some possible further work and research directions related to the SSE framework, supporting tools and the SOIL MAPPER service are presented in this section.

TOOLBOX

The Intecs TOOLBOX is a service integration supporting component under continuous development. By using this tool, the service provider doesn't need to develop the Web Service's SOAP interface from the scratch in any programming language. On the other hand its XML scripting language needs to be learned by the service provider and that language has some drawbacks like some missing functionality with respect to usual programming languages and its XML-inherited verbosity.

1. If more than one service has to be integrated by a service provider, some parts of the TOOLBOX scripts may be reused to handle similar service logics. As an example, the *ftpGet* scripting element cannot be used with a complete FTP address. Since complete FTP address is the output form proposed in the OpenGIS catalogue specification[17] its handling should be a builtin TOOLBOX feature. Nevertheless, the part of the SM script which parses a full FTP address to use it in the *ftpGet* element, is suitable for parameterized usage. At the time of writing the inclusion of this part into another script is feasible only by literal inclusion which leads to script

duplication and lesser maintainability. Hence the utility of a tool to include common scripts for parameterized usage in a TOOLBOX installation, something like a “load XML script library” configuration option might be useful.

2. Scripting features limitation can be overtaken by means of the Java method invocation element. This element enables the service provider to develop its own Java classes, deploy them within the TOOLBOX and then invoke these classes methods inside a script. This is the key feature to full TOOLBOX extensibility since it allows seamless inclusion of arbitrary complex Java programs in the service. This feature is only available for Java classes thus a service provider with good Java knowledge can take full advantage of this feature with minimal effort. On the other hand, supporting only Java classes, this feature is likely to be ignored by service provider having development infrastructure based on other programming languages.

3. The major TOOLBOX issue evidenced in section 3.6 is a critical reliability problem since it can cause the denial of service on the service provider system. This queuing-related problem should be fixed as soon as possible in the TOOLBOX implementation or it must be worked around by the implementation of a customized queuing system by the service provider.

Graphical interfaces and stylesheets

To provide graphical interfaces and XML-messaging handling to his service’s operations, the service provider needs to build the service stylesheet, thus XSL, HTML and JavaScript knowledge is required for service integration. The SSE framework can ease the service integration by providing a graphical interface editor for service interfaces enabling service providers to design basic service front ends without the need for the aforementioned knowledge. Moreover, by service schema examination, the editor could automatically build such basic service interface. A service provider needing a more advanced interface can always design the general layout and then directly edit the service stylesheet.

SOIL MAPPER

The integrated SM service described in this writing will be delivered to ESA for non commercial utilization in its research activities while a commercial version of this service will also be integrated onto the SSE system and maintained directly by MEEO. To be published as a commercial service, SM needs at least the following additions:

- development of a Web Service interface for an accounting system to manage elaboration payments and credit available to the user;
- linking of the accounting interface into the service workflow;
- implementation of the RFQ operation in case of non fixed elaboration costs;
- development of an authorization workflow if the service access has to be restricted.

Appendix A

Server Setup

This chapter describes the hardware configuration and details the software setup of the service hosting machine.

A.1 Hardware

This section contains a brief hardware description of the server hosting the SM service.

The server, named "*eoservice*", has an Asus P5B motherboard equipped with a dual core CPU "Intel Core2 6600" working at 2,4GHz clock and 2 DDR2 1 GB DRAM modules for a total of 2 GB RAM memory. Storage devices are managed by the on board JMicron SATA controller and consist of 2 WDC SATA hard drives of 320 GB capacity each; the mirror raid configuration ensures a total fault tolerant storage capacity of 320 GB. Graphical processing is supplied by an ATI Radeon X1600 Accelerated Graphic Card while networking devices are the (unused) on board RTL8111 Gigabit controller and an additional RTL8139 PCI Ethernet controller.

A.2 Software

This section describes the installation and configuration steps of the software components needed to make the hardware machine into a web service providing host.

A.2.1 Software components setup

Operating System

First of all an operating system (OS) is needed to manage the hardware ad run the other applications. Three Linux distributions were tested for fitness on the server:

Debian 3.1 (sarge) distribution was the first choice but resulted not installable on the server due to a lack of kernel support for the on board JMicron controller.

Ubuntu 6.10 (edgy) is the second distribution tested. It has proven to have be the best hardware compatibility with the server but caused problems with the Intecs TOOLBOX.

Fedora Core 6 (FC6) is the third and last distribution tested and chosen for production. A compatibility issue with the on board RTL8111 controller has led to the need for the additional RTL8139 PCI card.

Since the chosen distribution is FC6 here follows the setup description for this distribution only: The FC6 installation begins with a standard setup DVD from which *base system*, *gnome* and *KDE* desktop managers where chosen to support the optional graphical tools for simplified server management; additional packages include the *Samba* network file server and the GNU Java runtime.

The following sections describe more in detail the setup of the other software applications needed on the FC6 OS: the Sun's Java Software Development Kit (JDK), the Apache Tomcat servlet container[1] and the Intecs web services TOOLBOX.

SUN Java JDK

The Java platform JDK version 1.5 is needed to run Tomcat 5.5 which is in turn a mandatory requirement for the Intecs TOOLBOX; Since the distributed Sun JDK rpm is not compatible with FC6, installation of the sun JDK is made with a bit complex procedure described here as a series of steps aimed to use the alternatives package and the supported JPackage.org project for a clean and elegant installation[18].

1. Acquire root privileges either by logging in as root or by the *su* command.
2. Get the `1.5.0_09` JDK from Sun download archive [2] choosing Linux RPM in self-extracting file (`jdk-1_5_0_09-linux-i586-rpm.bin`) and saving it to the current directory.
3. Check the availability of *wget* on the system installing it if it's not present:

[1]Apache Tomcat is an open source servlet container used in the official Reference Implementation of Java server side technologies. A servlet is a Web-based Java application running server side. More information on these topics can be found on the Tomcat website: `http://tomcat.apache.org/`.

[2]sun java archive: `http://java.sun.com/products/archive/j2se/5.0_09/index.html`

```
yum install wget
```

4. Download the keys for the JPackage repository

   ```
   rpm --import http://jpackage.org/jpackage.asc
   ```

   and install the repository information for the yum tool while saving (pushd) the current working directory where the self extracting file will be downloaded

   ```
   pushd /etc/yum.repos.d
   wget http://jpackage.org/jpackage.repo
   ```

5. The Jpackage .repo file defaults to jpackage-general repository enabled, however, according to [18] it is better to disable it by editing the /etc/yum.repos.d/jpackage.repo file and changing line(s) enabled=1 to enabled=0 avoiding unwanted updates without notice. JPackage repository can be always enabled in yum by adding the *–enablerepo=jpackage-generic* and *–enablerepo=jpackage-generic-nonfree* options to the command line.

6. Make also sure that the packages to manipulate RPMs are installed

   ```
   yum install fedora-rpmdevtools
   yum install rpm-build
   ```

7. Next we need a complete GNU JDK installation to let the alternatives package configure both the java runtime and compiler tools so list the GNU Java compiler(gcj) available but not installed package and install them

   ```
   yum list available '*gcj*
   yum install listedPackage1 listedPackage2 ...
   ```

8. Next execute the downloaded shell script from Sun to install the JDK

   ```
   popd
   chmod 755 jdk-1_5_0_09-linux-i586-rpm.bin
   ./jdk-1_5_0_09-linux-i586-rpm.bin
   ```

9. Now, with the JDK installed (although not correctly for FC6) it is time to install the SUN JDK compatibility RPM from the JPackage.org to create all the needed links in /etc/alternatives, /usr/lib/jvm and `/usr/java/jdk1.5.0_09` where FC6 expects them for the JDK to work correctly:

   ```
   yum --enablerepo=jpackage-generic-nonfree install java-1.5.0-sun-compat
   ```

10. At this point the SUN JDK should be the default on the system (this can be checked by `java -version`) and changed using the alternative package:

```
      alternatives --config java
```

As a final note on this procedure the correct way to remove a JDK installed with this procedure is:

```
yum erase java-1.5.0-sun-compat
rpm -e jdk-1.5.0_09-fcs
```

As the last step in order to have system wide Java environment variables settings put the following text in the file "/etc/profile.d/java.sh:

Listing A.1: Java environment configuration

```
JAVA_HOME="/usr/lib/jvm/java"
CLASSPATH=${JAVA_HOME}/lib/tools.jar:./

export JAVA_HOME CLASSPATH
```

Tomcat and TOOLBOX

Once the Sun JDK is installed as described in A.2.1 installing Tomcat and the Intecs TOOLBOX is straightforward:

1. Acquire root privileges either by logging in as root or by the *su* command.
2. Download the TOOLBOX installer from Intecs[3] (at the time of writing i downloaded the TOOLBOX v5.0:TOOLBOX-install-linux-5.0.jar).
3. Launch the installer:
   ```
   java -jar TOOLBOX-install-linux-5.0.jar
   ```
   and follow the automated procedure choosing the full TOOLBOX installation and a new Tomcat installation in the `/usr/share` directory.

ENVI and IDL

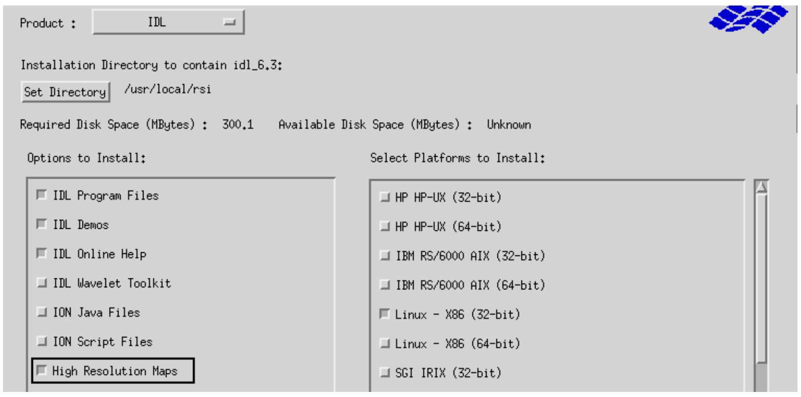
In order to execute the soil mapper implementation on this server the setup of ENVI and IDL is needed. The setup launcher is common to both the setup CD and assuming we have the cdrom device linked to /dev/cdrom and a mounting directory called /media/cdrom the shell commands to mount the CD and launch the setup process are:

Listing A.2: Cd mount and setup launch

```
mount -o rw -t iso9660 /dev/cdrom /media/cdrom
/bin/sh /med     ia/cdrom/xinstall.sh
```

[3]Intecs Toolbox download site: `http://mass.pisa.intecs.it/download/download.jsp`

The first step is to setup IDL from the installation CD so insert the IDL setup CD and execute the scripts in Listing A.2. Follow the on screen instructions and enable the options as shown in Figure A.1.


Product : IDL

Installation Directory to contain idl_6.3:

Set Directory /usr/local/rsi

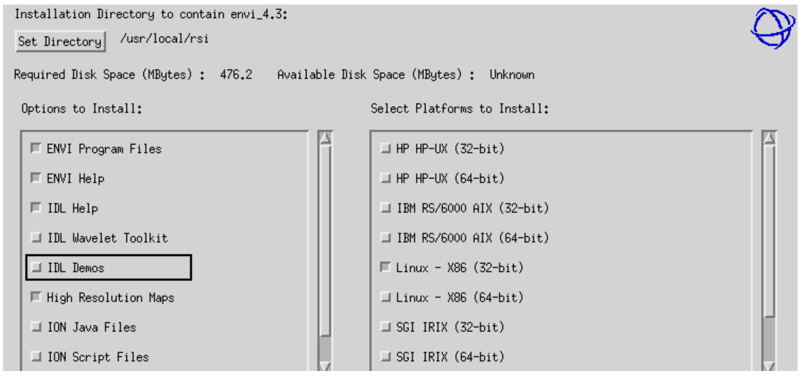
Required Disk Space (MBytes) : 300.1 Available Disk Space (MBytes) : Unknown

Options to Install:
- ☑ IDL Program Files
- ☑ IDL Demos
- ☑ IDL Online Help
- ☐ IDL Wavelet Toolkit
- ☐ ION Java Files
- ☐ ION Script Files
- ☑ High Resolution Maps

Select Platforms to Install:
- ☐ HP HP-UX (32-bit)
- ☐ HP HP-UX (64-bit)
- ☐ IBM RS/6000 AIX (32-bit)
- ☐ IBM RS/6000 AIX (64-bit)
- ☑ Linux - X86 (32-bit)
- ☐ Linux - X86 (64-bit)
- ☐ SGI IRIX (32-bit)

Figure A.1: IDL setup configuration

Complete the IDL setup by creating the proposed links in `/usr/local/bin` then perform the same operations with the ENVI setup CD, the ENVI options to enable are shown in Figure A.2. Once the ENVI setup process has been completed a valid licence has to be


Installation Directory to contain envi_4.3:

Set Directory /usr/local/rsi

Required Disk Space (MBytes) : 476.2 Available Disk Space (MBytes) : Unknown

Options to Install:
- ☑ ENVI Program Files
- ☑ ENVI Help
- ☑ IDL Help
- ☐ IDL Wavelet Toolkit
- ☐ IDL Demos
- ☑ High Resolution Maps
- ☐ ION Java Files
- ☐ ION Script Files

Select Platforms to Install:
- ☐ HP HP-UX (32-bit)
- ☐ HP HP-UX (64-bit)
- ☐ IBM RS/6000 AIX (32-bit)
- ☐ IBM RS/6000 AIX (64-bit)
- ☑ Linux - X86 (32-bit)
- ☐ Linux - X86 (64-bit)
- ☐ SGI IRIX (32-bit)
- ☐ SGI IRIX (64-bit)

Figure A.2: ENVI setup configuration

provided for the SM binary file to be correctly executed. For this installation, a network licence provided by the MEEO server has been used.

A.2.2 Final system configuration

The final configuration consist in Tomcat boot time startup and administrative Tomcat and TOOLBOX user account configuration.

Tomcat boot time startup

Here follows the script i wrote for the Tomcat boot time startup:

Listing A.3: tomcat service startup script

```
#!/bin/bash
#
# Tomcat 5.5 service startup script
#
# chkconfig: 345 99 10
# description:  Tomcat 5.5 service startup script
#

#
# Source function library.
. /etc/init.d/functions
#
#
RETVAL=$?
TOMCATHOME="/usr/share/apache-tomcat-5.5.17"
#
case "$1" in
 start)
        if [ -f $TOMCATHOME/bin/startup.sh ]; then
                echo $"Starting Tomcat"
    bash $TOMCATHOME/bin/startup.sh
  fi
 ;;
 stop)
  if [ -f $TOMCATHOME/bin/shutdown.sh ]; then
                echo $"Stopping Tomcat"
    bash $TOMCATHOME/bin/shutdown.sh
  fi
 ;;
 restart)
        $0 stop
        sleep 5
        $0 start
 ;;
 status)
        echo "Sorry, no status information provided for this service"
 ;;
 *)
        echo $"Usage: $0 {start|stop|restart}"
  exit 1
 ;;
esac
#
exit $RETVAL
```

The above script must be placed in /etc/init.d/tomcat. In order for the system to execute the script at boot time register it with:

```
chkconfig --add tomcat
```

Account settings

To set a Tomcat user account with admin and manager rights overwrite the *tomcat-users.xml* file located in the conf subdirectory of the Tomcat installation with the following contents (replacing the asterisks with a strong password):

Listing A.4: tomcat users

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="tomadm" password="*******" roles="admin,manager"/>
</tomcat-users>
```

To change the default TOOLBOX administrative account edit the file `./webapps/TOOLBOX/WEB-INF/xml/adminUserx.xml` relative to the tomcat installation directory and replace the default username and password (admin/admin) with the liked username (TOOLBOX in this case) and a strong password. The server base configuration is now complete.

Appendix B

Listings

This appendix contains listings related to the work discussed in this thesis wich are too long for inclusion in other chapters.

B.1 Service schema

Listing B.1: The soilMapper.xsd schema file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://www.esa.int/mass" version="1.0"
  xmlns="http://www.esa.int/mass"
  xmlns:aoi="http://www.esa.int/xml/schemas/mass/aoifeatures"
  xmlns:mass="http://www.esa.int/mass"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:redefine schemaLocation="http://services.eoportal.org/schemas/1.4/mass.xsd">
    <xsd:complexType name="OrderInputType">
      <xsd:complexContent>
        <xsd:extension base="mass:OrderInputType">
          <xsd:sequence>
            <xsd:element name="elaborationType" type="xsd:string"/>
            <xsd:element name="outputFormat" type="xsd:string"/>
            <xsd:element name="inputFileLocation" type="xsd:string"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="OrderOutputType">
      <xsd:complexContent>
        <xsd:extension base="mass:OrderOutputType">
          <xsd:sequence>
            <xsd:element name="userName" type="xsd:token"/>
            <xsd:element name="passWord" type="xsd:token"/>
            <xsd:element name="orderResultURL" type="xsd:string"/>
            <xsd:element minOccurs="0" name="availabilityTime" type="xsd:string"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:redefine>
</xsd:schema>
```

B.2 TOOLBOX scripts

This section lists the three scripts used on the SM TOOLBOX installation.

B.2.1 First toolbox script: ftpOrder1.xml

Listing B.2: The first order script file: ftpOrder1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!--
    Document    : ftpOrder1.xml
    Created on : 31 gennaio 2007, 14.51
    Author      : alan
    Description:
        This is the first toolbox script fot the ftp order.
        It retrieves the source file by ftp as specified in the request
        and starts the requested elaboration.
        The url must be in the form:
          proto://user:pass@host/pathToFile

        The protocol is always considered to be ftp regardless of its actual value

        errorRaised  boolean variable and
        errorMessage string are used to signal a catched execution error
             to the other scripts so these must be initially set to false and "success"
-->

<sequence xmlns='http://pisa.intecs.it/mass/toolbox/xmlScript'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='http://pisa.intecs.it/mass/toolbox/xmlScript
    file:/F:/meeo/SoilMapper/toolboxScripting/src/tbx-scripting.xsd'>

<!--  VARIABLES FOR ERROR HANDLING BETWEEN SCRIPTS -->
  <setVariable name="errorRaised">
    <literal type="boolean" value="false"/>
  </setVariable>
  <setVariable name="errorMessage">
    <string>Success</string>
  </setVariable>


<!--  VARIABLES FOR LINUX AND WINDOWS PORTABILITY -->
<setVariable name="slash">
    <!-- <string>\</string> -->
    <string>/</string>
</setVariable>
<setVariable name="logdir">
    <!-- <string>e:\TBX_LOG</string> -->
    <string>/var/log/toolbox</string>
</setVariable>
<setVariable name="workdir">
    <!-- <string>e:\SM_WORK</string> -->
    <string>/usr/share/toolboxFtp</string>
</setVariable>
<setVariable name="launchCommand">
    <!-- <string>...</string> -->
    <string>/eoservice/soilmapper/scripts/soilmapper.sh</string>
</setVariable>

<setVariable name="sourceUrl">
  <xPath xmlns:mass="http://www.esa.int/mass">
    <xmlRequest/>
    <string>mass:sendOrderInputMsg/mass:sendOrderInput/mass:inputFileLocation</string>
  </xPath>
</setVariable>

<!-- FTP ACCESS DATA EXTRACTION begin
        the ftp data from the single variable sourceUrl
        in the form user:pass@host/parhToFile
        are extracted to the respective 4 variables:
        srcFtpUser, srcFtpPass, srcFtpHost, srcFtpPor, srcFtpPath
           an additional variable srcFileName is provided  containing
                 only the filename-->

<setText new="true">
    <variable name="sourceUrl"/>
</setText>

<if>
    <search absolute="true">
            <string>://</string>
    </search>
    <sequence>
        <!-- Protocol specification handling -->
        <gotoLineStart/>
        <mark name="protoStart"/>
        <fieldMove absolute="true" greedy="true" position="end" separators=":"/>
        <mark name="protoEnd"/>
        <setVariable name="protocol">
            <extract start="protoStart" end="protoEnd"/>
        </setVariable>
        <fieldMove greedy="true" separators="/"/>
        <mark name="srcStart"/>
        <gotoLineEnd/>
        <mark name="srcEnd"/>
        <setVariable name="sourceUrl">
            <extract start="srcStart" end="srcEnd"/>
        </setVariable>
        <log level="DEBUG">
            <stringCat>
                <variable name="protocol"/>
                <string> - </string>
                <variable name="sourceUrl"/>
            </stringCat>
        </log>
    </sequence>
</if>

<!-- access parameters extraction -->
<setText new="true">
    <variable name="sourceUrl"/>
</setText>
<fieldMove absolute="true" separators=":" position="start"/>
<mark name="fieldUserStart"/>
<fieldMove separators=":" position="end">
    <literal value="0"/>
</fieldMove>
<mark name="fieldUserEnd"/>
<setVariable name="srcFtpUser">
    <extract start="fieldUserStart" end="fieldUserEnd"/>
</setVariable>

<fieldMove absolute="true" separators=":">
    <literal value="2"/>
</fieldMove>
<mark name="fieldPasswordStart"/>
<fieldMove absolute="false" separators=":@" position="end">
    <literal value="0"/>
</fieldMove>
<mark name="fieldPasswordEnd"/>
<setVariable name="srcFtpPass">
    <extract start="fieldPasswordStart" end="fieldPasswordEnd"/>
</setVariable>

<fieldMove absolute="false" separators="@"/>
<mark name="fieldHostStart"/>
<fieldMove absolute="false" separators="/" position="end">
    <literal value="0"/>
</fieldMove>
<mark name="fieldHostEnd"/>
<setVariable name="srcFtpHost">
    <extract start="fieldHostStart" end="fieldHostEnd"/>
</setVariable>

<fieldMove absolute="true" separators="/">
    <literal value="2"/>
</fieldMove>
<mark name="fieldPathStart"/>
<fieldMove absolute="false" separators="@" position="end">
    <literal value="0"/>
</fieldMove>
<mark name="fieldPathEnd"/>
<setVariable name="srcFtpPath">
    <extract start="fieldPathStart" end="fieldPathEnd"/>
</setVariable>

<setVariable name="srcFtpPort">
    <string>21</string>
</setVariable>

<!-- Cecking if a nonstandard port has been specified
   (the hostname should contain a : ) -->
<setText new="true">
    <variable name="srcFtpHost"/>
</setText>
<if>
    <search absolute="true">
        <string>:</string>
    </search>
    <sequence>
        <fieldMove absolute="true" separators=":"/>
        <mark name="hostFieldS"/>
        <fieldMove separators=":" position="end">
            <literal value="0"/>
        </fieldMove>
        <mark name="hostFieldE"/>
        <setVariable name="srcFtpHost">
            <extract start="hostFieldS" end="hostFieldE"/>
        </setVariable>

        <fieldMove absolute="true" separators=":">
            <literal value="2"/>
        </fieldMove>
        <mark name="portFieldS"/>
        <gotoLineEnd/>
        <mark name="portFieldE"/>
        <setVariable name="srcFtpPort">
            <extract start="portFieldS" end="portFieldE"/>
        </setVariable>
    </sequence>
</if>

<!-- Filename Extraction -->
<setVariable name="srcFileName">
    <variable name="srcFtpPath"/>
```

```
</setVariable>
<setVariable name="bars">
  <literal value="1"/>
</setVariable>
<setText new="true">
    <variable name="srcFtpPath"/>
</setText>
<while>
  <search>
      <string>/</string>
      <variable name="bars"/>
  </search>
  <inc>
      <variable name="bars"/>
  </inc>
</while>
<if>
    <not>
        <eq>
            <literal value="1"/>
            <variable name="bars"/>
        </eq>
    </not>
    <sequence>
        <gotoLineEnd/>
        <mark name="fnameE"/>
            <fieldMove absolute="true" separators="/" >
                <variable name="bars"/>
            </fieldMove>
        <mark name="fnameS"/>
        <setVariable name="srcFileName">
            <extract start="fnameS" end="fnameE"/>
        </setVariable>
    </sequence>
</if>

<log level="DEBUG">
    <stringCat>
        <string>Ftp access data: User: </string>
        <variable name="srcFtpUser"/>
        <string> - Pass: </string>
        <variable name="srcFtpPass"/>
        <string> - Host: </string>
        <variable name="srcFtpHost"/>
        <string> - Port: </string>
        <variable name="srcFtpPort"/>
        <string> - FilePath: </string>
        <variable name="srcFtpPath"/>
        <string> - FileName: </string>
        <variable name="srcFileName"/>
    </stringCat>
</log>
<!-- FTP ACCESS DATA EXTRACTION end -->

<!-- Elaboration parameters extraction BEGIN -->
<setVariable name="paramElabType">
  <xPath xmlns:mass="http://www.esa.int/mass">
    <xmlRequest/>
    <string>mass:sendOrderInputMsg/mass:sendOrderInput/mass:elaborationType</string>
  </xPath>
</setVariable>
<setVariable name="paramOutFormat">
  <xPath xmlns:mass="http://www.esa.int/mass">
    <xmlRequest/>
    <string>mass:sendOrderInputMsg/mass:sendOrderInput/mass:outputFormat</string>
  </xPath>
</setVariable>
<!-- Elaboration parameters extraction END -->

<setVariable name="orderWorkDir">
    <stringCat>
        <variable name="workdir"/>
        <variable name="slash"/>
        <orderId/>
    </stringCat>
</setVariable>
<mkdir>
    <variable name="orderWorkDir"/>
</mkdir>
<setVariable name="localSrcFile">
     <stringCat>
         <variable name="orderWorkDir"/>
         <variable name="slash"/>
         <variable name="srcFileName"/>
     </stringCat>
</setVariable>

<!-- FTP IMAGE RETRIEVAL begin -->
<try>
    <ftpGet transfer="binary">
        <host>
            <variable name="srcFtpHost"/>
        </host>
        <port>
            <atoi><variable name="srcFtpPort"/></atoi>
        </port>
        <user>
            <variable name="srcFtpUser"/>
        </user>
        <password>
            <variable name="srcFtpPass"/>
        </password>
        <remotePath>
            <variable name="srcFtpPath"/>
        </remotePath>
        <localPath>
            <variable name="localSrcFile"/>
        </localPath>
    </ftpGet>
    <ifError errorMessageName="FtpGetError">
      <sequence>
        <log level="ERROR">
          <string>FTP get Error catched</string>
        </log>
        <setVariable name="errorRaised">
          <literal type="boolean" value="true"/>
        </setVariable>
        <setVariable name="errorMessage">
          <string>
            Ftp image retrieval failed.
            Please check the ftp download availability
            of the source image and
            the correctness of the provided data.
          </string>
        </setVariable>
      </sequence>
    </ifError>
</try>
<!-- FTP IMAGE RETRIEVAL end -->

<!-- ELABORATION LAUNCHER -->
<if>
    <not>
        <variable name="errorRaised"/>
    </not>
    <sequence>
        <setVariable name="cmdString">
            <stringCat>
                <variable name="launchCommand"/>
                <string> </string>
                <variable name="orderWorkDir"/>
                <string> </string>
                <variable name="srcFileName"/>
                <string> </string>
                <variable name="paramElabType"/>
                <string> </string>
                <variable name="paramOutFormat"/>
            </stringCat>
        </setVariable>
        <log level="DEBUG">
            <stringCat>
                <string>Exec command: </string>
                <variable name="cmdString"/>
            </stringCat>
        </log>
        <command asynchronous="true">
            <variable name="cmdString"/>
        </command>
    </sequence>
    <log level="INFO">
      <string>Error catched in first script, elaboration not started</string>
    </log>
</if>

</sequence>
```

B.2.2 Second toolbox script: ftpOrder2.xml

Listing B.3: The second order script file: ftpOrder2.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : ftpOrder2.xml.xml
    Created on : 2 febbraio 2007, 16.36
    Author     : alan
    Description:
        This is the second toolbox script which checks
                for response readiness
-->

<sequence xmlns='http://pisa.intecs.it/mass/toolbox/xmlScript'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='http://pisa.intecs.it/mass/toolbox/xmlScript
```

```
    file:/F:/meeo/SoilMapper/toolboxScripting/src/tbx-scripting.xsd'>

  <if>
    <variable name="errorRaised"/>
    <literal xmlns="http://pisa.intecs.it/mass/toolbox/xmlScript"
             type="boolean" value="true"/>
    <sequence>
      <if>
        <fileExists>
          <stringCat>
            <variable name="orderWorkDir"/>
            <variable name="slash"/>
            <string>outcome</string>
          </stringCat>
        </fileExists>
          <literal xmlns="http://pisa.intecs.it/mass/toolbox/xmlScript"
                   type="boolean" value="true"/>
          <literal xmlns="http://pisa.intecs.it/mass/toolbox/xmlScript"
                   type="boolean" value="false"/>
      </if>
    </sequence>
  </if>
</sequence>
```

B.2.3 Third toolbox script: ftpOrder3.xml

Listing B.4: The third order script file: ftpOrder3.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : ftpOrder3.xml
    Created on : 31 gennaio 2007, 14.30
    Author     : alan
    Description:
        This is the third toolbox script wich builds the response for the sse portal
        and configures the ftp accounting for results retrieval.
-->

<sequence
  xmlns='http://pisa.intecs.it/mass/toolbox/xmlScript'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='http://pisa.intecs.it/mass/toolbox/xmlScript
  file:/F:/meeo/SoilMapper/toolboxScripting/src/tbx-scripting.xsd'>

  <!-- Checking if an error has occoured (errorRaised) -->
  <if>
    <variable name="errorRaised"/>
    <sequence> <!-- Build ERROR response -->
      <setVariable name="ResultURL">
        <stringCat>
          <variable name="srcFtpHost"/>
          <string>:</string>
          <variable name="srcFtpPort"/>
          <string>/</string>
          <variable name="srcFtpPath"/>
        </stringCat>
      </setVariable>
        <xmlGetResponse>
          <xml attributePrefix="x" textTag="evaluate">
            <mass:getOrderOutput xmlns:mass="http://www.esa.int/mass">
              <mass:statusInfo>
                <mass:statusId>300</mass:statusId>
                <mass:statusMsg><evaluate name="errorMessage"/></mass:statusMsg>
              </mass:statusInfo>
              <mass:userName><evaluate name="srcFtpUser"/></mass:userName>
              <mass:passWord><evaluate name="srcFtpPass"/></mass:passWord>
              <mass:orderResultURL>
                <evaluate name="ResultURL"/>
              </mass:orderResultURL>
              <mass:availabilityTime>NA</mass:availabilityTime>
            </mass:getOrderOutput>
          </xml>
        </xmlGetResponse>
      </sequence>
      <sequence> <!-- checking elaboration outcome -->
        <setText new="true">
          <loadFile>
            <stringCat>
              <variable name="orderWorkDir"/>
              <variable name="slash"/>
              <string>outcome</string>
            </stringCat>
          </loadFile>
        </setText>
        <if>
          <search>
            <string>success</string>
          </search>
            <sequence> <!-- Normal termination (success): retrieve data response -->
              <setVariable name="ResultURL">
```

```
                <stringCat>
                  <string>ftp://87.23.139.78</string>
                </stringCat>
              </setVariable>
              <setVariable name="resFtpUser">
                <orderId/>
              </setVariable>
              <setVariable name="resFtpPass">
                <randomString length="6"/>
              </setVariable>
              <setVariable name="availTime">
                <string>3 days</string>
              </setVariable>

              <ftpAccount duration="3d">
                <user>
                  <variable name="resFtpUser"/>
                </user>
                <password>
                  <variable name="resFtpPass"/>
                </password>
                <rootDir>
                  <variable name="orderWorkDir"/>
                </rootDir>
              </ftpAccount>

              <!-- WORKDIR CLEANUP -->
              <fileDelete>
                <stringCat>
                  <variable name="orderWorkDir"/>
                  <variable name="slash"/>
                  <string>outcome</string>
                </stringCat>
              </fileDelete>
              <fileDelete>
                <stringCat>
                  <variable name="orderWorkDir"/>
                  <variable name="slash"/>
                  <string>log.out</string>
                </stringCat>
              </fileDelete>
              <fileDelete>
                <stringCat>
                  <variable name="orderWorkDir"/>
                  <variable name="slash"/>
                  <string>log.err</string>
                </stringCat>
              </fileDelete>

              <!-- XML RESPONSE CONSTRUCTION -->
              <xmlGetResponse>
                <xml attributePrefix="x" textTag="evaluate">
                  <mass:getOrderOutput xmlns:mass="http://www.esa.int/mass">
                    <mass:statusInfo>
                      <mass:statusId>0</mass:statusId>
                      <mass:statusMsg>Successful</mass:statusMsg>
                    </mass:statusInfo>
                    <mass:userName><evaluate name="resFtpUser"/></mass:userName>
                    <mass:passWord><evaluate name="resFtpPass"/></mass:passWord>
                    <mass:orderResultURL><evaluate name="ResultURL"/></mass:orderResultURL>
                    <mass:availabilityTime><evaluate name="availTime"/></mass:availabilityTime>
                  </mass:getOrderOutput>
                </xml>
              </xmlGetResponse>
            </sequence>
            <sequence>  <!-- Elaboration Failed: reporting error -->
              <xmlGetResponse>
                <xml attributePrefix="x" textTag="evaluate">
                  <mass:getOrderOutput xmlns:mass="http://www.esa.int/mass">
                    <mass:statusInfo>
                      <mass:statusId>500</mass:statusId>
                      <mass:statusMsg>Image processing failed</mass:statusMsg>
                    </mass:statusInfo>
                    <mass:userName>NA</mass:userName>
                    <mass:passWord>NA</mass:passWord>
                    <mass:orderResultURL>NA</mass:orderResultURL>
                    <mass:availabilityTime>NA</mass:availabilityTime>
                  </mass:getOrderOutput>
                </xml>
              </xmlGetResponse>
            </sequence>
        </if>
     </sequence>
  </if>
</sequence>
```

B.3 Launcher shell script

Listing B.5: The service launcher shell script

```
#Radiometric Calibration and
#Spectral Categorization Script
#MEEO S.n.c.
#Version 2.2
#Date:jan. 23 2007
#
# Path to execution scripts
# scriptPath="/eoservice/soilmapper/scripts"
scriptPath="/eoservice/soilmapper/scripts"
export programsPath="/eoservice/soilmapper/programs"
#
#Input Data Parameters Consistency Check
if [ $# -ne 4 ]
then
 echo "Error : Incorrect format"
   echo "Usage : $0 workdir filename classification_mode output_format"
   echo "workdir is the relative working directory for the elaboration"
   echo "filename is the source image zipped file"
   echo "classification_mode allowed values are Full,Reduced,Limited"
   echo "output_format allowed values are ENVI,GeoTIFF"
   exit 1
fi

#flag1="$3"
#flag2="$4"
if  [ "$3" != "Full" ] && [ "$3" != "Reduced" ] && [ "$3" != "Limited" ]
then
 echo "Error : bad 'number of classes'"
        echo "Usage: Classification Mode values permitted: Full,Reduced,Limited"
        exit 1
fi
if  [ "$3" == "Full" ]
then
        flag1="0"
fi
if  [ "$3" == "Reduced" ]
then
        flag1="1"
fi
if  [ "$3" == "Limited" ]
then
        flag1="2"
fi

if  [ "$4" != "ENVI" ] && [ "$4" != "GeoTIFF" ]
then
echo "Error : bad 'image format' flag "
        echo " Usage : Outout format values permitted: ENVI,GeoTIFF"
        exit 1
fi
if  [ "$4" == "ENVI" ]
then
        flag2="0"
fi
if  [ "$4" == "GeoTIFF" ]
then
        flag2="1"
fi

echo $flag1 $flag2

#Working Directory Definition
        cd $1
        tempath=`pwd`
        l7dirname=`echo $2 | cut -d. -f 1` #removing the filename extension
        mkdir $l7dirname
        cp $2 $l7dirname
        cd $l7dirname
        unzip $2
        cd SCENE1
#*******************************************
#  please, check the directory
#*******************************************
#       cp /usr/local/temp_soil_mapper/sm_scr_envi_1_prom ./
        cp $scriptPath/sm_scr_envi_1_prom.sh ./
#       cp /usr/local/temp_soil_mapper/sm_scr_envi_2_prom ./
        cp $scriptPath/sm_scr_envi_2_prom.sh ./
#
#********************************************

#Radiometric Calibration Application
        /bin/sh sm_scr_envi_1_prom.sh

#
#Spectral Categorization
        /bin/sh sm_scr_envi_2_prom.sh $flag1 $flag2 1>log.out 2>log.err
#
#Processing outcome determination
        results=`ls | grep SpCl`
        if [ -z "$results" ];
        then
          echo "failure" > $tempath/outcome
```

```
            else
              echo "success" > $tempath/outcome
            fi

#Working Directory Clean-up
            mv *SpCl* $tempath
            mv *.out  $tempath
            mv *.err  $tempath
            cd $tempath
            rm -Rf $17dirname
            rm -f *.tfw
```

B.4 FTP Service Stylesheet

Listing B.6: The FTP based service's stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:mass="http://www.esa.int/mass"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:oi="http://www.esa.int/oi"
  xmlns:aoi="http://www.esa.int/xml/schemas/mass/aoifeatures"
  xmlns:eoli="http://earth.esa.int/XML/eoli"
  xmlns:gml="http://www.opengis.net/gml" >
   <!-- Import statements -->
   <xsl:import href="./mass.xsl"/>
   <xsl:import href="./massCatalogue.xsl"/>
   <!-- Apply the template for the root element from MASS standard template -->
   <xsl:template match="/">
     <xsl:apply-imports/>
   </xsl:template>
   <!-- Templates used to display Search input information using HTML format  -->
   <xsl:template match="mass:sendSearchInput"/>
   <!-- Templates used to generate Search input information using XML format -->
   <xsl:template match="mass:sendSearchInput" mode="XML"/>

   <xsl:template match="mass:getOrderOutput" mode="XML"/>

   <xsl:template match="mass:sendOrderInput">
     <script language="JavaScript" type="text/javascript">
       function checkMandatoryFields(form)
       {
       //complete FTP address validation
       var validator = new RegExp
         ("^ftp://[\\w@.-]+:[\\w-]+@[\\w-]+\\.[\\w.-]+(:\\d+)?/[\\w-%\/.]+$","i");
       if (!validator.test(form["inputFtpAddress"].value))
       {
       alert("The FTP address provided is not correct, please check.");
       return false;
       }
       return true;
       }
     </script>
     <table  width="500" height="250" border="0">
       <tbody>
   <tr><td><br/></td></tr>
   <tr>
     <td valign="bottom" align="left" CLASS="stylesheetBoldText">
       Insert the complete FTP address of the image to be processed
     </td>
   </tr>
   <tr>
     <td align="center" CLASS="stylesheetText">
       <input name="inputFtpAddress" size="60" type="text"
       value="ftp://user:pass@host.name:portNumber/pathToFile/fileName.ext"/>
     </td>
   </tr>
   <tr>
     <td valign="top" align="left" CLASS="stylesheetText">
       Note: The url must be in the complete form containing all the data needed
       for the correct image retrieval by our service (for example:
       ftp://user:password@server.net:2121/Images/ImageToProcess.zip )<br/>
       The port number is optional and, if not specified will default to 21.
     </td>
   </tr>
   <tr><td><br/></td></tr>
   <tr>
     <td valign="bottom" align="left" CLASS="stylesheetBoldText">
       Choose elaboration Type
     </td>
   </tr>
   <tr>
     <td valign="top" align="left" height="30" CLASS="stylesheetText">
       <select name="elaborationType">
         <option value="Full" selected="1">Full number of classes (72)</option>
         <option value="Reduced">Reduced number of classes (38)</option>
         <option value="Limited">Limited  number of classes (15)</option>
       </select>
     </td>
   </tr>
```

```
<tr><td><br/></td></tr>
<tr>
  <td valign="bottom" align="left" CLASS="stylesheetBoldText">
    Choose output image format
  </td>
</tr>
<tr>
  <td  valign="top" align="left" height="30" CLASS="stylesheetText">
    <select name="outputFormat">
      <option value="ENVI">ENVI data format</option>
      <option value="GeoTIFF" selected="1">GeoTiff data format</option>
    </select>
  </td>
</tr>
    </tbody>
  </table>
</xsl:template>

<!-- Template for the order input information using XML as output format -->
<xsl:template match="mass:sendOrderInput" mode="XML">
  <mass:commonInput>
    <mass:orderId>
      <xsl:value-of select="orderId"/>
    </mass:orderId>
  </mass:commonInput>
  <mass:sendOrderInput>
    <mass:elaborationType>
      <xsl:value-of select="elaborationType"/>
    </mass:elaborationType>
    <mass:outputFormat>
      <xsl:value-of select="outputFormat"/>
    </mass:outputFormat>
    <mass:inputFileLocation>
      <xsl:value-of select="inputFtpAddress"/>
    </mass:inputFileLocation>
  </mass:sendOrderInput>
</xsl:template>

<!-- Template for the order output information using HTML as output format -->
<xsl:template match="mass:getOrderOutput">
  <table width="640">
  <xsl:if test="mass:statusInfo/mass:statusMsg">
    <tr>
      <td height="30" class="stylesheetBoldText">
        Result Status
      </td>
      <td class="stylesheetBoldText">
        <xsl:value-of select="mass:statusInfo/mass:statusMsg"/>
      </td>
    </tr>
  </xsl:if>

  <xsl:choose>
  <xsl:when test="mass:statusInfo/mass:statusId = 0">
  <tr>
    <td height="20" class="stylesheetBoldText">
      Result is available at
    </td>
    <td class="stylesheetText">
      <a target="_blank"><xsl:attribute name="href">ftp://
      <xsl:value-of select="mass:userName"/>:
      <xsl:value-of select="mass:passWord"/>@
      <xsl:value-of select="substring-after(mass:orderResultURL,'//')"/>
      </xsl:attribute>ftp://<xsl:value-of select="mass:userName"/>:
      <xsl:value-of select="mass:passWord"/>@
      <xsl:value-of select="substring-after(mass:orderResultURL,'//')"/></a>
    </td>
  </tr>
  <tr>
    <td class="stylesheetBoldText">
      Availability period
    </td>
    <td class="stylesheetText">
      <xsl:value-of select="mass:availabilityTime"/>
  </td>
  </tr>
</xsl:when>

<xsl:otherwise>
  <tr>
    <td class="stylesheetBoldText">
      Additional error information:
    </td>
    <td class="stylesheetText">
      <xsl:text> </xsl:text>
    </td>
  </tr>
  <tr>
    <td class="stylesheetBoldText">
      Username
    </td>
    <td class="stylesheetText">
      <xsl:value-of select="mass:userName"/>
      </td>
    </tr>
    <tr>
      <td class="stylesheetBoldText">
        Password
      </td>
      <td class="stylesheetText">
        <xsl:value-of select="mass:passWord"/>
      </td>
    </tr>
    <tr>
      <td class="stylesheetBoldText">
        Related URL
      </td>
      <td class="stylesheetText">
        <xsl:value-of select="mass:orderResultURL"/>
      </td>
    </tr>
  </xsl:otherwise>
      </xsl:choose>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

B.5 FTP service WSDL description

Listing B.7: The FTP based service's WSDL description

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://www.esa.int/mass"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:mass="http://www.esa.int/mass"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:tns="http://www.esa.int/mass"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <schema attributeFormDefault="qualified"
      elementFormDefault="qualified"
      targetNamespace="http://schemas.xmlsoap.org/wsdl/"
      xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://www.esa.int/mass"
      schemaLocation="http://87.23.139.78:8080/TOOLBOX/WSDL/SoilMapper/soilMapper.xsd"/>
    <import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
      schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
    </schema>
  </types>
  <message name="StartHeader">
    <part element="wsa:MessageID" name="MessageID"/>
    <part element="wsa:ReplyTo" name="ReplyTo"/>
  </message>
  <message name="ContinueHeader">
    <part element="wsa:RelatesTo" name="RelatesTo"/>
  </message>
  <message name="sendOrderInput">
    <part element="tns:sendOrderInputMsg" name="parameters"/>
  </message>
  <message name="sendOrderOutput">
    <part element="tns:sendOrderOutputMsg" name="parameters"/>
  </message>
  <message name="returnOrderResultInput">
    <part element="tns:returnOrderResultInputMsg" name="parameters"/>
  </message>
  <message name="returnOrderResultOutput">
    <part element="tns:returnOrderResultOutputMsg" name="parameters"/>
  </message>
  <portType name="SoilMapper">
    <operation name="sendOrder">
      <input message="tns:sendOrderInput" name="sendOrderInput"/>
      <output message="tns:sendOrderOutput" name="sendOrderOutput"/>
    </operation>
  </portType>
  <portType name="SoilMapperCallback">
    <operation name="returnOrderResult">
      <input message="tns:returnOrderResultInput" name="returnOrderResultInput"/>
      <output message="tns:returnOrderResultOutput" name="returnOrderResultOutput"/>
    </operation>
  </portType>
  <binding name="SoilMapperSoapBinding" type="tns:SoilMapper">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sendOrder">
      <wsdlsoap:operation soapAction="sendOrder"/>
      <input>
        <wsdlsoap:header message="tns:StartHeader" part="MessageID" use="literal"/>
        <wsdlsoap:body use="literal"/>
      </input>
      <output>
        <wsdlsoap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <binding name="SoilMapperCallbackSoapBinding" type="tns:SoilMapperCallback">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="returnOrderResult">
      <wsdlsoap:operation soapAction="returnOrderResult"/>
      <input>
        <wsdlsoap:header message="tns:ContinueHeader" part="RelatesTo" use="literal"/>
        <wsdlsoap:body use="literal"/>
      </input>
      <output>
        <wsdlsoap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="SoilMapper">
    <port binding="tns:SoilMapperSoapBinding" name="SoilMapper">
      <wsdlsoap:address location="http://87.23.139.78:8080/TOOLBOX/services/SoilMapper"/>
    </port>
  </service>
  <service name="SoilMapperCallback">
    <port binding="tns:SoilMapperCallbackSoapBinding" name="SoilMapperCallback">
      <wsdlsoap:address location="http://openuri.org"/>
    </port>
  </service>
  <plnk:partnerLinkType name="SoilMapper">
    <plnk:role name="SoilMapperServiceProvider">
      <plnk:portType name="tns:SoilMapper"/>
    </plnk:role>
    <plnk:role name="SoilMapperServiceRequester">
      <plnk:portType name="tns:SoilMapperCallback"/>
    </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

Bibliography

[1] Mihai Datcu, Klaus Seidel, Oscar Guerra, and Manfred Schroeder. *KIM Knowledge Driven Information Mining in Remote Sensing Image Archives.* Project executive summary, ESA, Nov 2002. downloaded from `http://earth.esa.int/rtd/Documents/KIM_Executive_Summary.doc` on Dec-18-2006.

[2] International Institute for Geo-Information Science and Earth Observation. *ITC's database of Satellites and Sensors.* Website. consulted on Feb-21-2007 from `http://www.itc.nl/research/products/sensordb/searchsat.aspx` searching “landsat”.

[3] Marco Folegani, Simone Mantovani, and Stefano Natali. *SOIL MAPPER System and Products Description*, Jan 2007. downloaded from `http://www.meeo.it/docs/SOIL_MAPPER_report.pdf` on Jan-28-2007.

[4] A. Baraldi, V. Puzzolo, P. Blonda, L. Bruzzone, and C. Tarantino. Automatic spectral rule-based preliminary mapping of calibrated landsat tm and etm+ images. *IEEE Trans. Geosci. And Remote Sensing*, 44(9):2563–2586, Sep 2006.

[5] José Achache. A new perspective for earth observation. *ESA bullettin*, 116:22–33, Nov 2003.

[6] Yves Coene and Claire Bawin. *Service Support Environment Architecture, Model and Standards*, Dec 2004. downloaded from `http://earth.esa.int/rtd/Documents/SSE_Whitepaper_2.pdf` on Dec-18-2006.

[7] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.0.* W3C, fourth edition, Aug 2006. W3C Recommendation, Available online at `http://www.w3.org/TR/2006/REC-xml-20060816/`.

[8] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. *XML Schema Part 1: Structures.* W3C, second edition, Oct 2004. W3C Recommendation, Available online at `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`.

[9] W3C. *The Extensible Stylesheet Language Family (XSL).* Website. available online at `http://www.w3.org/Style/XSL/`.

[10] James Clark and Steve DeRose. *XML Path Language (XPath) 1.0.* W3C, Nov 1999. W3C Recommendation, Available online at `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[11] W3C. *Web Services Architecture.* Website, Feb 2004. available online at `http://www.w3.org/Style/XSL/`.

[12] W3C. *Web Services Description Language (WSDL) 1.1.* Website. available at `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[13] W3C. *SOAP Specifications.* Website. available online at `http://www.w3.org/TR/soap/`.

[14] Tony Andrews et al. *Business Process Execution Language for Web Services version 1.1.* IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, May 2003. OASIS standard, Available online at `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`.

[15] SSE Team. *Service Support Environment Interface Control Document.* SpaceBel, Sep 2006. Issue 1.5, unpublished draft for service providers. Issue 1.4 available online at `http://services.eoportal.org/massRef/documentation/icd.pdf`, Dec-18-2006.

[16] M. C. Terzi P. Nencioni, S. Gianfranceschi. *Earthnet Online XML Font End.* ESA, Intecs et al., Apr 2006. Issue 2.4, available online at `http://earth.esa.int/rtd/Documents/EOLI-XML-ICD.pdf`.

[17] Jolyon Martin. *OpenGIS® Catalogue Services - Minimal Profile for EO products using WSDL and SOAP.* ESA, OpenGis et al., 0.1.0 draft edition, Jun 2005. downloaded from `http://earth.esa.int/XML/eoli/` on Jan-19-2007.

[18] Jan K. Labanowski. *Sun Java Development Kit on FC6.* Computational Chemistry List ltd., 2007. downloaded from `http://ccl.net/cca/software/SOURCES/JAVA/JSDK-1.5/index6.shtml` on Jan-08-2007.

Acronyms

AOI	Area Of Interest
BPEL	Business Process Execution Language
CEOS	Committee on Earth Observation Satellites
EO	Earth Observation
EOLI	Earthnet OnLine Interactive
ESA	European Space Agency
ETM	Enhanced Thematic Mapper
FTP	File Transfer Protocol
GML	Geography Markup Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICD	Interface Control Document
IDE	Integrated Development Environment
IIM	Image Information Mining
JS	JavaScript
KEI	KIM Extension and Installation
KIM	Knowledge based Information Mining
MEEO	Meteorological and Environmental Earth Observation

NASA National Aeronautics and Space Administration

PM Process Manager

RS Remote Sensing

RTD Research and Technology Development

SM SOIL MAPPER

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SSE Service Support Environment

TM Thematic Mapper

USGS United States Geological Survey

W3C Word Wide Web Consortium

WS XML Web Service

WSA Web Service Architecture

WSDL Web Services Description Language

XML EXtensible Markup Language

XPath XML Path Language

XSD XML Schema Definition

XSL EXtensible Stylesheet Language

XSLT XSL Transformations

Acknowledgements

I wish to thank all the people who made the realization of this thesis possible: my supervisor Mirco Andreotti and Eleonora Luppi for their precious advices; Stefano Natali, Marco Folegani and Simone Mantovani (the MEEO staff), for all their support and for introducing me in the international context of Earth Observation-related projects; Carlos Ferreo for the kind support on the EOLI interfacing and all the people involved in developing and supporting the SSE system.

Ringrazio Ermi, per la dolcezza del suo sorriso, capace di illuminare anche i momenti più cupi, e per tutto il suo amore; la ringrazio inoltre per le sue dolci ed immancabili russate sul secondo tempo dei film d'azione (ho le prove) e per la sua tendenza ad aggiungere un grado di incertezza nei disegni dell'universo con i suoi piccoli disastri, seguiti da un candido "Vedi, con me non ci si annoia mai.": è vero!

Un ringraziamento speciale a mia madre "la Ross", per il suo immancabile affetto, ed a tutta la mia famiglia.

Ringrazio Bibi e Lilla, le gatte più coccolone del pianeta, che stavolta hanno fatto a turno per acciambellarsi sui libri di testo o sulla tastiera del PC, introducendo nei miei scritti innumerevoli serie di caratteri casuali, ad annunciare l'obbligo di una pausa coccole; se avessero un centesimo per ogni loro fusa ora sarebbero milionarie.

Ringrazio Bicio ed Elena, Toli, Denis, Davide, Valerio, Mirko (in giro per il mondo), Busso e tutti quelli che per brevità non menziono per la loro sincera amicizia.

Ringrazio i miei compagni di corso di Laurea specialistica per il tempo trascorso assieme, per gli innumerevoli "scambi culturali", e per le nostre cene di "specialisticafe"; ringrazio in particolare: Mattia, Neryo, il Coss, Mario, Valerio, Marika ed Alfredo.