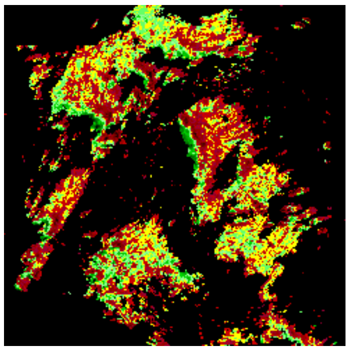
COMETA

A Metadata Combination Framework
in support of Nowcasting R&D activities



MASTER THESIS

LORENZO CLEMENTI

April 2008

Thesis supervisors:

Prof. Dr. Rolf INGOLD
DIVA Group
and
Dipl. Phys. Igor GIUNTA
MeteoSwiss

UNIVERSITÉ DE FRIBOURG SUISSE
UNIVERSITÄT FREIBURG SCHWEIZ

DIVA Group
Department of Computer Science
University of Fribourg (Switzerland)

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology
MeteoSwiss

"The trouble with weather forecasting is that it's right too often for us to ignore it and wrong too often for us to rely on it."

- *Patrick Young*

Acknowledgements

First, I would like to thank Prof. Rolf Ingold for giving me the opportunity to do my Master thesis at MeteoSwiss and for supervising my work. Thank you very much also to Gianmario Galli and Igor Giunta at MeteoSwiss, that conceived the framework described in this document.
A special thank to Igor for his essential support during my work in Locarno and for the knowledge he passed on to me. Thanks very much to Luca for the challenging jogging sessions and to all the MeteoSwiss co-workers in Locarno for the very nice period I spent there.
Last but not least, I would like to thank my family that always supported me during my studies.
This work is dedicated to my grandfather.

Abstract

COMETA is a flexible framework which facilitates the characterisation of meteorological data by means of metadata and supports the consistent combination of very different kinds of data. Its goal is to develop and validate integrated approaches in the context of the meteorological forecasting class called nowcasting. COMETA is conceived extensible, allowing to easily incorporate further or more refined data characteristics and combination methods. XML files are used to describe data types and processing information, while a set of Java classes composing the framework provides functionalities to verify data consistency and to manage data combination. In addition, it allows to improve existing algorithms through parameter perturbation functionalities.

Its capabilities have been tested through a concrete use case. A synergetic application, which uses both radar and satellite observations has been implemented and exploited in case of thunderstorms developments over the Alpine region. High quality instantaneous precipitation fields, retrieved by radar, have been forecasted according to dynamical features, retrieved by satellite.

COMETA proved to be a valid tool for setting up and automating large R&D sets of experiments with little effort; first quality assessments shed a light on potential meteorological developments of this nowcasting technique.

Keywords: Nowcasting, XML Metadata, JAXB, Radar and Satellite data forecasting

Table of Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation and goal

Weather prediction has attracted the attention of human being throughout the centuries. Already the Babylonians used to predict the weather through the observation of clouds movements, shapes and development [13]. Since then, and in particular in the last century, prediction skills dramatically improved, thanks to scientific findings and technological instruments. Clouds and their life cycle remain a key element for predicting weather evolution, particularly in case of rapidly forming, developing and decaying thunderstorms, which consistently impact on human activities also through precipitation.

Depending on the time and space scale of the atmospheric phenomena, three main meteorological development streams have historically established: *climatology* (years, centuries), *weather forecasts* (∼3 hours to ∼10 days) and nowcasting and very short range forecasts (0 to ∼3 hours).

This work focuses on nowcasting applications. Meteorological services have at their disposal large amounts of data coming from various measurement instruments, like ground stations, ground radars and satellite based sensors. A special category of emerging applications are combining these data for inferring as fast as possible relevant information about ongoing meteorological events [Wil04]. Due to computational limitations, nowcasting algorithms are balancing between (fast, nearly linear) semi-empirical and (slow, complex) physically based approaches. Special care and intensive testing of several concepts are requested to the specialists.

The goal of this work is to design and implement an information system which organises the different types of data, e.g. radar-derived objects and satellite derived displacement vectors, for supporting R&D activities in refining existing or implementing new nowcasting algorithms.

This framework relies on *metadata*, which describe *data types* and *processing information*. When a new combination method has to be tested, consistency checks among data types and processing information are firstly performed. The tool will then establish all needed steps for combining the data, taking into account all the details according to the degree of complexity.

1.2 A starting example

Suppose your mission as a nowcasting scientist is to improve the nowcasting skills of your company. You dispose of a vast spectrum of data like, for example, satellite and radar images, air temperature, etc. It happens that while you are having a coffee break, a forecaster tells you about a very new approach he is been using: it seems that a careful combination of some data allowed him to significantly improve his ability to predict the displacement of thunderstorm cells. Thrilled by this finding, you now would like to scientifically validate it. Here is where COMETA comes into play: by writing a few XML metadata files and a Java class, COMETA allows you to easily perform a large amount of tests whose results can be statistically analysed. Furthermore, the parameters perturbation technique – described later in Chapter 2 and 3 – provides a way for tuning your method configuration, thus optimising its performances for the region of your interest.

1.3 Work organisation and schedule

This master thesis is the fruit of a collaboration between the DIVA Group of the Department of Computer Science at the University of Fribourg[1] (Switzerland) and the RASA Team of MeteoSwiss[2]. The work has been accomplished by Lorenzo Clementi at the MeteoSwiss centre in Locarno-Monti over a period of 20 weeks from November 2007 until April 2008. Table 1.1 shows a detailed tasks description for each phase of the project.

The present document reports on the work accomplished during this master thesis and it serves as a basis for its evaluation. It encloses both a technical description of COMETA and the documentation concerning the use case we developed to test the framework functionalities.

1.4 Structure of this document

Concluded the Introduction, Chapter 2 gives an overview of the framework, describing its architecture and main components. Chapter 3 provides a more detailed description of COMETA; this chapter contains information about the development, the set up and the use of the framework. Chapter 4 illustrates the use case prepared to test the framework with additional insights on meteorological implications. Chapter 5 concludes this work and proposes an outlook.

1.5 About this document

This documentation is based on a LaTeX template created by Patrik Fuhrer and Dominique Guinard (Software Engineering Group, University of Fribourg, Switzerland).

[1]`http://diuf.unifr.ch/diva`, DIVA stands for *Document, Image and Voice Analysis*
[2]`http://www.meteosvizzera.ch`

The template is open source and freely available at `http://diuf.unifr.ch/softeng/guidelines.html`.

1.5.1 Project name

Originally called *MEMcaf*, this project has been renamed to COMETA towards the end of the work (February 2008). You will still find the old acronym in some documents (e.g. [Cle07, Cle08]).

1.5.2 Notations and Conventions

- Formatting conventions:
  - **Bold** and *italic* are used for emphasis and to signify the first use of a term.
  - `SansSerif` is used for web addresses.
  - `Code` is used in all Java code and generally for anything that would be typed literally when programming, including keywords, constants, method names and variables, class names, and interface names.
- The present report is divided in Chapters. Chapters are broken down into Sections. Where necessary, sections are further broken down into Subsections, and Subsections may contain some Paragraphs.
- Figures, Tables and Listings are numbered inside a chapter. For example, a reference to Figure j of Chapter i will be noted *Figure i.j*.
- Source code is displayed as follows:

```java
Matrix3f rotMat = new Matrix3f();
rotMat.fromAngleNormalAxis( FastMath.DEG_TO_RAD * 45.0f, new Vector3f( 1.0f, 0.0f, 0.0f));
box.setLocalRotation( rotMat );
```

Several acronyms are used throughout the text, their definitions can be found in Appendix A.

Date	Description
5th Nov. 2007	Beginning of work
5th Nov. 2007 - 14th Dec. 2007	**First phase** Literature study and familiarisation with the thesis subject Framework architecture specification Prototype development First presentation at MeteoSwiss First mid-term report editing
14th Dec. 2007	**First deadline** First mid-term presentation at the DIVA Group, in Fribourg Delivery of the first mid-term report (see [Cle07])
14th Dec. 2007 - 8th Feb. 2008	**Second phase** Implementation of a real, end-to-end use case. Result analysis and second presentation at MeteoSwiss Second mid-term report editing (see [Cle08])
21st Dec. 2007 - 6th Jan. 2008	Winter vacations
8th Feb. 2008	**Second deadline** Second presentation at MeteoSwiss Delivery of the second mid-term report
8th Feb. 2008 - 4th Apr. 2008	**Third phase** Framework consolidation Thesis editing
4th Apr. 2008	**Final deadline** Source code and thesis report delivery
17th Apr. 2008	**Thesis presentation** Second presentation, in Fribourg

Table 1.1: Timeline and repartition of tasks.

2 Framework overview

2.1 Nowcasting, meteorological background

The study of an environmental object like the atmosphere requires the use of a wide spectrum of observation types and processing techniques. Beneath well-known acquisition instruments like weather stations and radio-soundings, remote-sensing techniques have been establishing an important and well defined role in weather applications. It accounts of many active and passive instruments like weather radars, lidars and weather satellites (e.g. Meteosat, MetOp, NOAA).

The whole gathered observation flux is then used by the weather services (singularly or through international organisations, like the European ECMWF) for very different applications, which account for numerical weather models for simulating the atmosphere over several days, climate applications for studying radiative balances over years, and very-short range weather predictions for monitoring and following of specific weather events (nowcasting).

In all these fields, at a certain point of the long chain linking data acquisition to end users, it is necessary to apply a multi-source approach either for extracting new parameters or for simply presenting the results to the final users (e.g. the forecaster) [TK04]. The applied researches work on back-stage for implementing new application or adapting old techniques to new data. In some cases, they developed refined instruments that, however, apply only to very peculiar inter-comparisons, like calibration between radar and rain-gauges or inter-satellite sensors. These applications are based on a bottom-up approach, since they are built to quickly respond to a concrete question. In other cases, overall relational databases for initialising big climate or weather models have been developed (for example at the ECMWF); their final result, however, represents a nearly completely independent dataset projected into another space and time scale.

This work aims to furnish to MeteoSwiss Locarno-Monti a simple but flexible framework for characterising and preparing the data, mainly coming from different remote-sensing sensors, in a best suitable way for subsequently apply comparison methods (physically or statistically based). The final goal of this instrument is to optimise the research and development (R&D) effort for inter-comparing several methods and several data, taking under control data-quality and the complexity of the problem [GG07].

2.1.1 State of the art: quick look

This Subsection reports on how Cometa relates to existing applications.
The use of metadata to characterise meteorological information is attracting more and more attention (see Subsection 2.3.1). The growth of geographical web-services requiring standard information in form of XML files has boosted the need for metadata. As well, numerical weather prediction models (e.g. the European centre ECMWF) are heavy and complex mathematical systems interfacing with a huge amount of different physical observations. Due to the nowcasting context, which this works deals with, only main concepts are taken as inspiration.
Some projects deal specifically with nowcasting. For example, the Nowcasting Working Group (NWG) of the WWRP aim to coordinate national efforts to "achieve implementation of end-to-end nowcasting systems, techniques and procedures" [AAV03]. Similarly, the SAF/NWC Consortium, aims "to produce the Software to deal with the Nowcasting and VSRF using the characteristics of the MSG SEVIRI data" using a combination of satellite data and numerical weather prediction models [12]. Again, the features required to our framework are somehow different. First, the problem is tackled at a different scale. We do not want to produce an end-to-end nowcasting system but, rather, a specific tool for testing some peculiar combination of existing algorithms. Second, the framework is not required to work in real time, but it will be used offline as a support to R&D specialists for testing and validating new techniques based on historical observations and using an experimental approach. Third, since the data treated by Cometa can be very heterogeneous (e.g. remote sensing images and rain gauge measurements), it is very difficult to define a specific standard for metadata, like the ones mentioned above. For these reasons, although Cometa bases on existing technologies, the resulting framework does not look like other existing applications.

2.1.2 Terminology

For major clarification, we defined some specific terms used all along this document and in the framework implementation.

Product refers to a physical quantity, e.g. rain rate or brightness temperature.

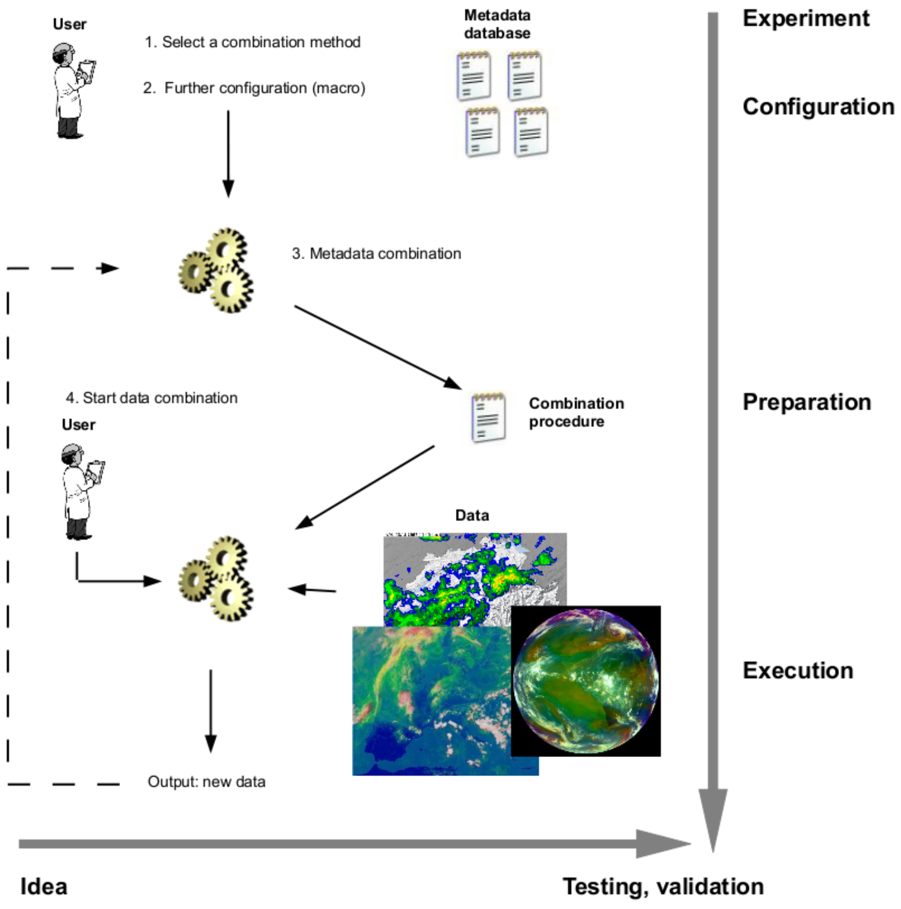


Figure 2.1: Information flow for the COMETA framework.

Observation is a measure of a *product* valid for a certain time or timespan.

Data is a numerical representation of an *observation*.

Combination method or method is a sequence of actions applied onto metadata to verify their coherence (e.g. synchronisation).

Product metadata describe the parameters of a *product*.

Method metadata describe the parameters of a *method*.

Combination procedure is the result of the metadata combination. This file contains all the information needed to execute the algorithm.

2.2 Information flow

Figure 2.1 shows the information flow for the COMETA framework.

1. **Basic configuration**: the method's default configuration is given by the existing metadata.

2. **Additional configurations**: further information are either inserted by the user at runtime or collected through other metadata files. Conceptually, in this step a product is transformed into an observation by locating it on the timeline.
3. The framework collects and **combines the metadata and generates an intermediate procedure file**. This file can be reviewed by the user and it also serves as system log.
4. On user demand, **the combination procedure file is launched** and, according to the information contained in it, a sequence of UNIX commands is executed in order to perform the required actions on the data. This step may depend on third party software that has to be installed on the machine where COMETA is running (see 3.6).

The data combination can be automatically executed several times with a varying value for a given parameter, without requiring additional configuration (see 3.2.1). This process is called *parameter perturbation* and is explained in detail later on.

2.3 Design and key elements

As **flexibility** is a fundamental requirement for our framework, we designed a reusable architecture so that several combination methods can be implemented. All these methods exploit the functionalities furnished by the COMETA framework.

2.3.1 Metadata

Metadata is “information about data” or, in other words, it provides the information needed to correctly interpret some specific data. We use metadata to describe both the features of the products and the parameters of the combination methods.

With the exponential growth in density and coverage of observations, the handling of meteorological metadata became crucial. At national and international scale, great efforts performed in recent years to produce standard representations of *geografical metadata* (e.g. in the USA, [8] [NIS04], or through the WMO [WMO05]). Most of these projects rely on the **XML** markup language to structure metadata: it offers a well defined syntax that can be automatically treated by a computer and, at the same time, can be easily understood by a human. Furthermore, XML documents are validated thanks to XML Schema, so that the validity of their structure can be formally verified.

Huge metadata standards are far too big for the purposes of our application [8]. The approach, however, has proven valuable even at a smaller scale [AS00] and, for this reason, we adopted XML and XML Schema to describe the metadata of our framework.

2.3.2 Products hierarchy

Our products metadata shall be **hierarchically organised** in order to reduce information redundancy. Think for example of two different satellite sensors that take images of the same size, but that observe different wavelengths. In this case, the information about the image resolution is the same for the images issued from the two sensors; the content of the images, however, differs consistently. Metadata shall allow a hierarchical organisation so that it is possible to create a class of satellite images that share the same resolution but that differ for other features. XML ability to establish links between documents is exploited to create parent - child relations between different metadata files, therefore generating a hierarchy of products.

In the case of radar and satellite images metadata, this hierarchy mainly reflects the processing of an observation (scanning strategy, frequency, calibration and so on). For example, in case of satellite images, several base data can be combined together to generate a new derived product that inherits the features of his parents.

2.3.3 Software components

The choice of XML has a number of consequences. It showed natural to store metadata files in a native XML database and – among several options [11] – *eXist XML database* has been chosen due to its opensource nature and the availability of a good Java API [3]. Java has accordingly been selected as development language, in particular because it offers two useful libraries:

1. JAXB, which provides an elegant and efficient way to map XML files to Java objects and vice versa [9].
2. The XMLDB API that is a collection of classes allowing to easily interact with the *eXist* XML database where the metadata are stored [15].

An additional argument in favour of using Java and XML is that they provide a **multiplatform** environment. As shown by Figure 3.4, the framework components can run on several machines equipped with different operating systems.

Unlike what has been done for metadata, no particular storage method has been defined for data itself. As a matter of fact, in most cases data is already organised in specific databases where it is stored and archived. At the RASA Team, for example, the main source of data are radar and satellite images, which are already organised in specific filesystems that also provide the utilities needed to retrieve current and historical observations.

Cometa provides further on a graphical user interface to ease its use (see 3.7).

2.3.4 Code organisation

The source code of Cometa is structured into 10 packages that we will now briefly describe. For a more detailed documentation please refer to Chapter 3 or see the Javadoc in Appendix C.

cometa contains the main class of the project.

cometa.combine contains the classes that handle the metadata combination phase.

cometa.execute controls the data combination phase.

cometa.exist governs the communication with the metadata database.

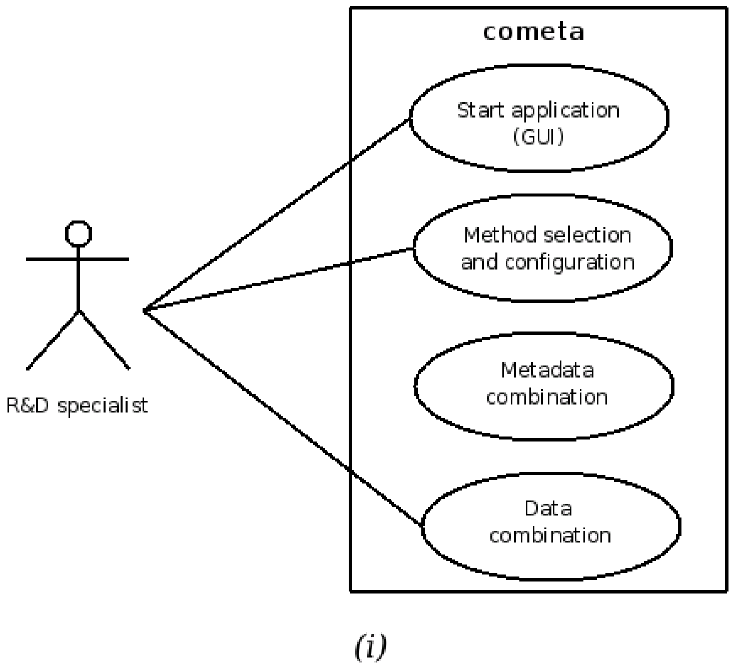
cometa.exist.triggers groups the classes implementing database triggers.

cometa.gui contains the implementation of the framework's GUI.

cometa.translator handles the translation between Java objects and XML documents. This package relies on the JAXB technology.

cometa.translator.method, cometa.translator.procedure, cometa.translator.product are the three packages that contain the code generated by JAXB.

Figure 2.2 *(ii)* provides a graphical representation of the packages described above. The framework's logic is implemented in the packages cometa.exist, cometa.combine and cometa.execute, whereas the other packages perform some support tasks (e.g. management of GUI elements or translation of XML files to Java objects). The UML *use case* diagram of Figure 2.2 *(i)* highlights the points where the user interaction is required



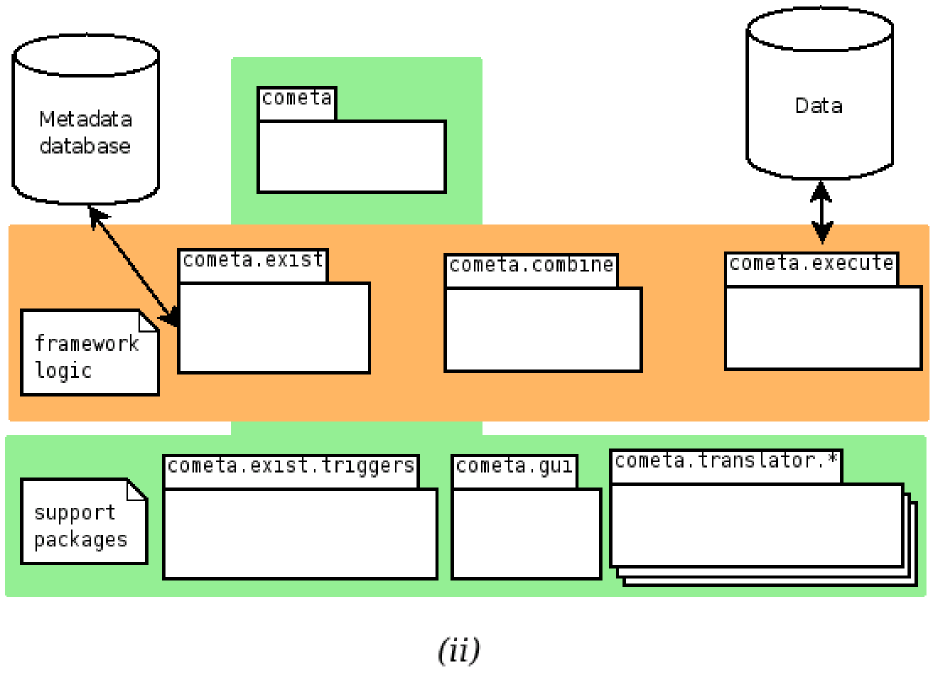


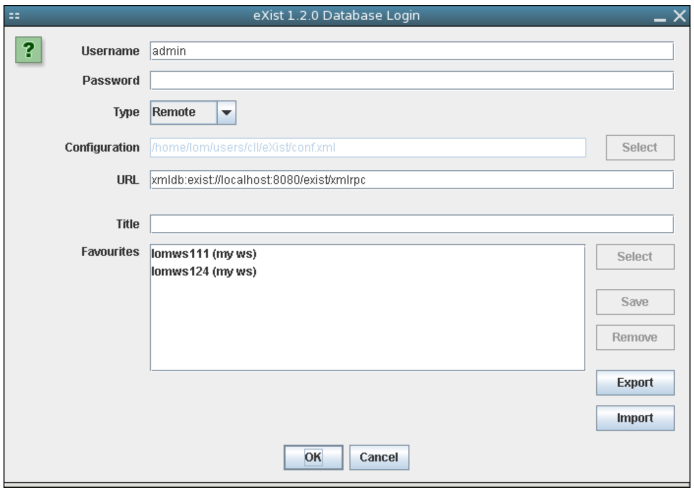
Figure 2.2: Framework general structure.

3 Framework technical description

3.1 Metadata database: eXist

eXist is a platform independent, native XML database written in Java. *eXist* comes in different versions (see [3]), the selected one is the jar standalone version 1.2, which runs on any machine offering a JVM v. ≥ 1.4.

The database installation is easy and can be done in a few steps thanks to a graphical interface. In addition to the database server, which has to be started from the command line, *eXist* offers a graphical user interface to administrate the database.


eXist 1.2.0 Database Login

Username: admin
Password:
Type: Remote
Configuration: /home/lom/users/cll/eXist/conf.xml
URL: xmldb:exist://localhost:8080/exist/xmlrpc
Title:
Favourites:
lomws111 (my ws)
lomws124 (my ws)

Figure 3.1: *eXist* startup dialog.

3.1.1 eXist configuration

After the installation, the following operations allow to start and configure the database:

```
$ cd eXist/bin
$ ./startup
```

This starts the database server. The administration interface is then launched:

```
$ cd ..
$ java -jar start.jar
```

A window like the one of Figure 3.1 should appear. This dialog allows the administrator to configure the database and to save the information for further logins. Fill in the form with the following information:

Username, password: the ones chosen during the installation procedure.

Type: Remote

URL: xmldb:exist://[hostname]:[port]/exist/xmlrpc, where [hostname] stands for the name of the machine where *eXist* is running and [port] for the port it uses (usually 8080).

Save the session and log into the database. A new window appears: this is the main interface where collections and documents can be managed. The use of this interface is very intuitive and, therefore, we will not discuss it here. For additional information, please refer to the official documentation [3].

Cometa requires three collections to be set up at the root of the database:

methods contains the methods metadata.

products contains the products metadata.

validators contains the XML Schema used to validate methods and products metadata, as well as procedure files.

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <triggers>
    <trigger event="store,update" class="cometa.exist.triggers.TriggerName"/>
  </triggers>
<7collection>
```

Listing 3.1: collection.xconf file.

3.1.2 Setting up triggers in eXist

In a database, *triggers* are actions executed in response to specific events. In *eXist*, there are four different kinds of events that can trigger an action:

remove: when a document is removed.

rename: when a document is renamed.

store: when a new document is stored in a collection.

update: when a document is modified.

As we have seen in 2.3.2, products hierarchical structure is modeled by links between metadata files: triggers are therefore used to ensure *referential integrity* between the documents. To set up a trigger, follow these steps:

1. Create a file called collection.xconf that looks like the one of Listing 3.1. The attribute event indicates the events that trigger a call to the class defined by the value of the class attribute.
2. The collection /db/system/config/db mirrors the database structure (i.e. the structure of /db). For the trigger to be active in a given collection, collection.xconf has to be placed into the corresponding collection under /db/system/config/db (typically, triggers are placed in /db/system/config/db/products).
3. Restart *eXist*.

For a Java class to be a valid trigger, it must implement the DocumentTrigger[1] interface and it must be accessible from the *eXist* classpath. A good folder to place the compiled version of the trigger is eXist/lib/user.
ProductReferentialIntegrity[2] is a default trigger provided by the framework. It verifies the referential integrity every time that a new product is stored into the products collection or when an existing product is modified. This is done by checking if the products corresponding to the mentioned IDs actually exist in the database. For more information about this trigger, refer to the Cometa Javadoc or see the source code in Appendix C.

3.2 XML metadata and XML Schema validation

Cometa deals with three classes of XML files: method metadata, product metadata and procedure files. Each of these classes has a specific structure, which is enforced by three different XML Schema [18]. Method and product metadata are stored in the methods and products database collections, whereas XML Schema files are stored in the validators

[1]org.exist.collections.triggers.DocumentTrigger
[2]cometa.exist.triggers.ProductReferentialIntegrity

```java
private static boolean isValid(File schema, XMLResource xmlDocument)   {
  SAXParser parser = new SAXParser();
  try{
    parser.setFeature("http://xml.org/sax/features/validation",true);
    parser.setFeature("http://apache.org/xml/features/validation/schema",true);
    parser.setFeature("http://apache.org/xml/features/validation/schema-full-checking",
           true);
    parser.
    setProperty("http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation",
           schema.getPath());

    Validator handler = new Validator();

    parser.setErrorHandler(handler);
    parser.parse(new InputSource(Translator.getInputStream(xmlDocument)));
    if (handler.validationError == true) {
      System.out.println("XML Document has Error : "+handler.saxParseException.getMessage());
      return false;
    } else {
      return true;
    }

    } catch(java.io.IOException ioe){
      System.out.println("IOException " + ioe.getMessage());
    } catch (SAXException e) {
      System.out.println("SAXException " + e.getMessage());
    } catch (JDOMException jde) {
      jde.printStackTrace();
    }
      return false;
}
```

Listing 3.2: Validation of an XML document.

collection. Procedure files, on the other hand, are not stored in the database but are just written as output on a given location in the filesystem, at the end of the metadata combination phase.

Method and products metadata are validated against their XML Schema before every metadata combination in order to avoid run time errors due to malformed metadata. Validation is done thanks to the open source modules offered by *Xerces-J* [14], which provide a Java toolkit to validate XML documents against a given XML Schema. Listing 3.2 shows the isValid method of the class ValidateXML[3] that performs the validation of an XML document.

As we shall see later in Section 3.3, the schema are also requested by JAXB in order to generate the source code for the corresponding Java objects. Next Subsection provides a closer look at each one of these three XML file classes[4].

3.2.1 Methods metadata

A combination method is characterised by two main elements, a set of observations and a set of parameters, that specify the configuration with which the method is executed.

[3]cometa.translator.ValidateXML

[4]For some concrete example of XML and XML Schema files, refer to the source code in Appendix C.

A method metadata file has a root element named method, whose attribute methodID must be unique within the collection and it constitutes the method identifier used by COMETA. The root element contains two child elements:

parameters is a sequence of at least one observation element that describes the observation(s) required by this combination method. Every observation is specified by a name, a type and possibly a default value.
Both the parameters element and its children of type observation can contain some simpleParam or compoundParam. A simple parameter is made of a *(name, value)* pair describing a particular feature of an observation or of the method itself. The values for simple parameters are set by the user at the runtime. Parameters of type integer can contain a perturbate attribute, which is described below.
The value of a compoundParam element, on the other hand, is not directly defined by the user, but it depends on some simple parameters and it is computed by a specific procedure, called *action*. More precisely, a compoundParam contains three attributes: a name, an action and a list of arguments. The latter is a list of comma separated names of previously defined simpleParams; when the compoundParam needs to be read, its value is computed on the fly by passing to the method defined by action the current value of the simple parameters listed by the arguments attribute. It is important to remark that the content of action must correspond to a method implemented in CompoundParamActions[5]; more details on how this class has to be updated can be found in the Javadoc (Appendix C).

combinationProcedure is a simple element that contains the reference to a Java class implementing the metadata combination method. This reference is given by the attribute class of the element ref and it must correspond to the name of a Java class located in the package cometa.combine. It is compulsory that any class appearing here implements the cometa.combine.Combiner interface (for a more detailed description, see 3.4.1).

As mentioned above, every simpleParam element may contain a perturbate attribute which can be set to true or false. If this attribute is present and its value is set to true, then the corresponding parameter is *perturbed*, i.e. its value will vary within a given range of values specified by the user at the runtime. The result of the perturbation is a set of experiments that can be used to asses the impact of a given parameter in a certain combination method.

3.2.2 Products metadata

Products carry some important information: the features of the physical quantity (or quantities) they measure and – for elaborated products – their processing chain. Metadata must hence express this information in a structured format.

A product metadata starts with a product root element containing a productID attribute that must be unique for all the products in the collection. The root element contains three child elements:

dataFeatures is a list of dataParam elements, that is to say elements describing pairs of *(name, value)* properties for the current product. Additionally, the dataFeatures

[5] cometa.combine.CompoundParamActions

element can have a base attribute whose value is the identifier of another product from which the current product will inherit the data features.

lineage The name of this element is inspired by [AS00] and its goal is to model the sequence of actions needed to generate the current product. lineage contains a list of products identifier and one method identifier; the meaning of this information is "the current product is generated by method x and it depends on products y and z". During the metadata combination phase, the framework recursively verifies this hierarchy, until the *base products* (i.e. the products whose metadata contain no lineage element) are found.

additionalInformation contains some additional information that is ignored by COMETA (this can be considered metadata of metadata).

It is important to point out the difference between the base attribute of dataFeatures and the lineage element. The latter is used to model the chain of operations that generate the current product (i.e. the product described by the metadata). For this reason, lineage can contain several parent products, because a complex product can be produced by a single method that needs several products as input.
The use of the base attribute, on the other hand, merely indicates that the current product inherits the featrues of another product. Moreover, base can contain a single value, which means that a child product has at most a parent, contrary to what happens with lineage.

3.2.3 Procedure files

Procedure files contain the UNIX commands that are executed in the data combination phase, together with some additional information (which is optional). The structure of an XML procedure file is therefore very simple: the root element is named combinationProcedure and it contains a list of info and execute elements, which are both simple types containing only a text string. When a procedure is run, the content of the execute elements is executed, whereas the info elements are ignored.

3.3 XML to Java mapping: JAXB

JAXB is an API that allows a comfortable interaction between Java programs and XML documents [OM03]. With respect to the very common DOM and SAX, JAXB tackles the problem at a higher level, without having to "manually" parse XML documents. In fact, JAXB uses the XML Schema [17] defining the structure of a set of XML files to generate one (or more) Java class(es) that models the schema.

Figure 3.2 shows the architecture of the JAXB API. Given an XML Schema, the *binding complier* that comes with the default distribution of JAXB generates the Java classes mapping XML files to Java objects. *Marshalling* is the action of converting Java objects to an XML file and *unmarshalling* consist in creating Java objects from an XML file; both these operations are handled by JAXB.

The following listing illustrates the unmarshalling of an XML metadata file:

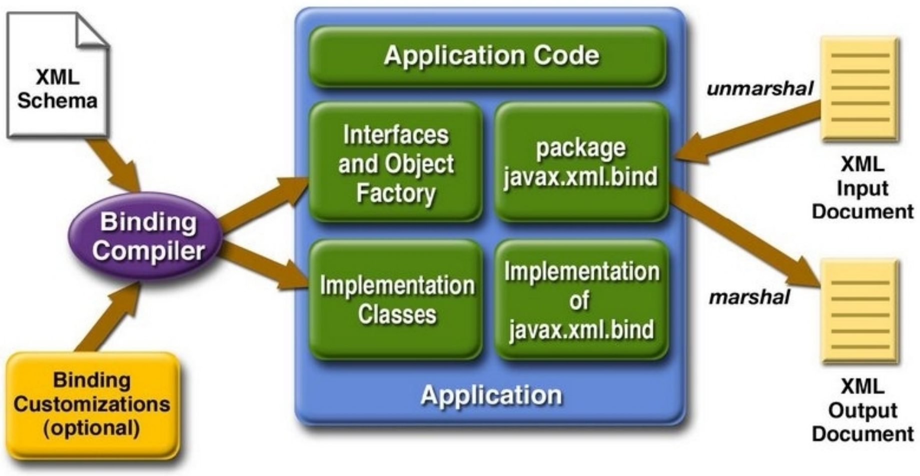


Figure 3.2: JAXB architecture. Source: `sun.com`

```java
JAXBContext jaxbContext = JAXBContext.newInstance("cometa.translator.method");
Unmarshaller u = jaxbContext.createUnmarshaller();
method = (Method) u.unmarshal(Translator.getInputStream(xmlMethod));
```

The next lines execute the marshalling of an XML procedure file:

```java
JAXBContext jaxbContext = JAXBContext.newInstance("cometa.translator.procedure");
Marshaller marshaller = jaxbContext.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, new Boolean(true));
String filename = method.getParamValue("outputFile");
marshaller.marshal(combinationProcedure, new FileOutputStream(filename));
```

As shown by the previous code snippets, the marshalling and unmarshalling operations offer an elegant way to map Java objects to XML files.
To generate the Java classes corresponding to a certain XML Schema, execute the following command:

```
$ xjc.sh -p package schema.xsd
```

where xjc.sh is the binding complier that comes with JAXB and package is the name of the package to which the generated code will belong (e.g. cometa.translator.method).

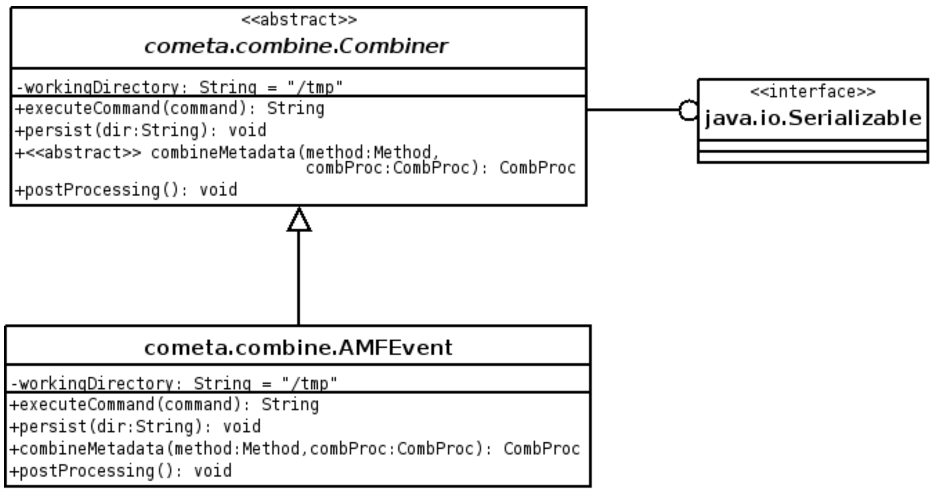
3.4 Metadata combination

The metadata combination is implemented by the CombinationEngine[6] class. In this phase, the Java objects representing a combination method contain the parameter values selected by the user at the runtime.

The operations executed by CombinationEngine are:

1. Products hierarchy verification: the processing chain for each product is generated; this information is then written to the procedure file in form of info tags (see 3.2.3).

[6] cometa.combine.CombinationEngine



<<abstract>>
cometa.combine.Combiner
-workingDirectory: String = "/tmp"
+executeCommand(command): String
+persist(dir:String): void
+<<abstract>> combineMetadata(method:Method, combProc:CombProc): CombProc
+postProcessing(): void
<<interface>>
java.io.Serializable
cometa.combine.AMFEvent
-workingDirectory: String = "/tmp"
+executeCommand(command): String
+persist(dir:String): void
+combineMetadata(method:Method,combProc:CombProc): CombProc
+postProcessing(): void

Figure 3.3: AMFEvent implements a concrete combination method.

2. Metadata combination: the combination method referenced by the method metadata is instantiated and executed. This step produces the exec information contained in the procedure file (see 3.2.3).
3. Finally, the procedure file is written to the disk.

The metadata combination is executed in a new thread that terminates when the combination is over. This optimisation avoids the application freezing while the combination is being executed.

3.4.1 Implementing a new combination method

For a Java class to be a valid combination method, and thus to appear in a ref element of a method metadata (see 3.2.1), it must extend the cometa.combine.Combiner abstract class, as shown by Figure 3.3. This class defines (among others) the following, important elements:

1. The workingDirecotry field represents the path of the directory where the procedure file is written at the end of the metadata combination. Furthermore, this folder is used to write some temporary files generated by COMETA at the runtime.
2. combineMetadata is an abstract method that must be implemented by all the subclasses. It defines the concrete steps that must be carried out by this combination method. Typically, this method adds to the procedure file a sequence of actions such as retrieving data from an archive and processing it.
   Note: the mentioned actions are *not* directly executed by this method but they are written to the procedure file which will be executed later (on user demand).
3. The postProcessing method is executed at the end of the data combination phase. Its use is optional: if no post processing needs to be done, the implementation of this method can be a dummy, empty method.

The post processing method enables a clean division between the metadata combination phase and the data combination phase. postProcessing is invoked when the data combination phase is over, allowing therefore to execute additional operations like, for example, the evaluation of the result.

For some concrete examples of combination method implementations, please refer to Chapter 4 that presents an end-to-end use case for our framework.

3.4.2 How to add a combination method to Cometa

This section aims to provide a short description of the steps required to add a combination method to our framework.

1. First, define the combination method metadata by adding a new XML file to the method collection in the metadata database. For more information about this procedure, have a look at 3.1 and 3.2.1.
2. Next, create a new combiner class, as described in 3.4.1. For example, your new class could be implemented in the file MyNewCombiner.java.
3. Compile the newly created class:

   ```
   $ javac -cp /path/to/cometa.jar MyNewCombiner.java
   ```

   where /path/to/cometa.jar is the path to the Cometa jar archive.
4. Finally, add the .class file to Cometa:

   ```
   $ cometa.sh add MyNewCombiner.class
   ```

   cometa.sh is a UNIX shell script described later in 3.6.

After restarting the application, the new combination method should appear in the drop down menu of the main dialog (see 3.7 for more details).

3.5 Data combination

The data combination process is straightforward. In fact, given as input a procedure file, the framework simply go through the exec tags and executes the corresponding UNIX commands, as shown by Listing 3.3. Note that the code simulates the *output redirect* mechanism used by the UNIX shell.

As for the metadata combination, also the data combination is executed in a dedicated thread so that the application can be used even when the combination is running. According to the number of operations and the size of data to be treated, this phase can require a considerable amount of time.

3.6 Framework development and deployment at MeteoSwiss

The framework described in this document has been developed at MeteoSwiss using the following hardware and software configuration:

- Sun ultra 25, 1.34 GHz processor, 1-MB level 2 cache [7]

```
Iterator<String> execIterator = procedureFile.getExecute().iterator();

// execute all the "exec" tags
while (execIterator.hasNext()) {
  String command = convertEntities(execIterator.next());
  String outputOfcommand = "";
  if (command.contains(">") && ! command.contains(">>")) {
    StringTokenizer tok = new StringTokenizer(command, ">");
    command = tok.nextToken();
    String filename = tok.nextToken().replaceAll(" ", "");
    outputOfcommand = executeCommand(command);
    writeToFile(filename, outputOfcommand);
  } else if (command.contains(">>")) {
    StringTokenizer tok = new StringTokenizer(command, ">>");
    command = tok.nextToken();
    String filename = tok.nextToken().replaceAll(" ", "");
    outputOfcommand = executeCommand(command);
    appendToFile(filename, outputOfcommand);
  } else {
    outputOfcommand = executeCommand(command);
}
```

Listing 3.3: Data combination.

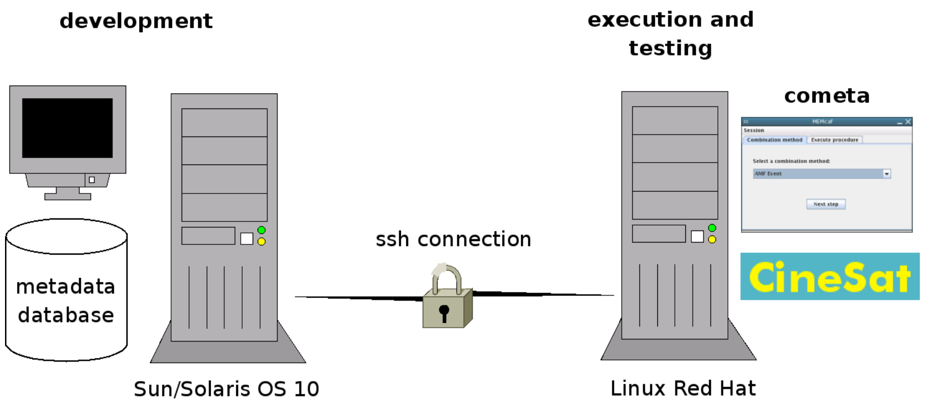


Figure 3.4: Framework deployment at MeteoSwiss.

- Solaris 10 operating system [6]
- Netbeans IDE v. 5.5.1 [5]

Figure 3.4 illustrates the deployment of the framework at MeteoSwiss. Netbeans is installed on the development machine lomws111. Since the combination method we used to test the framework requires some existing software (CineSat, [Sch06]), the framework's jar archive is copied to cinews01 where this software is installed. The metadata database, on the other hand, runs on lomws111 and COMETA communicates with it through the XML-RPC[7] implementation furnished by *eXist* [3, 16].

In addition to Java programming, the development has required the writing of various UNIX shell scripts in order to speed up some repetitive tasks, such as moving the framework jar archive from lomws111 to cinews01. These scripts, together with their documentation, can be found in Appendix C.

[7]XML-RPC is a "remote procedure calling using HTTP as the transport and XML as the encoding"[16].

3.6.1 How to set up Cometa in another environment

In order to set up Cometa in a new environment, execute the following operations:

1. Install and configure the *eXist* metadata database as described in Section 3.1.
2. Copy the Cometa Netbeans project to its new location and open it with Netbeans. Edit the values of the constants in MedataDB[8] so that they fit the new constraints and recompile. More information about those values are provided by the Javadoc and by the comments in the code.
3. Add a new combination method, as described in Subsection 3.4.2.
4. Launch the framwork's GUI (see Seciton 3.7) and test the combination method.

3.7 Cometa GUI

Section 3.4.1 illustrated the necessary steps to implement a new combination method. This operation requires some programming knowledge since a new Java class has to be implemented. The use of the framework, on the other hand, should not require Java skills and the goal of the graphical user interface is just to ease the use of Cometa.

To start the application, execute the following command:

```
$ java -jar cometa.jar
```

Figure 3.5 illustrates the dialogs composing the GUI, that we now briefly describe.

(i) This dialog is shown when Cometa is started. It contains two tabs, the first one allowing to select form a drop down menu the required combination method (the list of available methods is retrieved at startup from the metadata database).
In this dialog it is also possible to open a previously saved experiment by selecting Open from the Session menu.

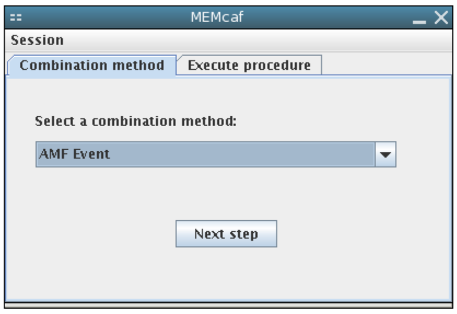
(ii) The second tab of the main dialog allows to select a procedure file to be executed. It is possible to either preview the file and then run it, as shown in *(v)*, or run it right away.

(iii) This form gathers the parameter values for the current experiment. Note that some parameters have a small icon on the right side: these are the parameters that can be perturbed (see 3.2.1). Furthermore, the user can save the current experiment or open an old one by using the Session menu.

(iv) The parameter perturbation dialog. As mentioned in Section 3.2.1, for a parameter to be perturbed, the user must indicate the lower bound, the upper bound and the step width. This will result in a set of experiments.

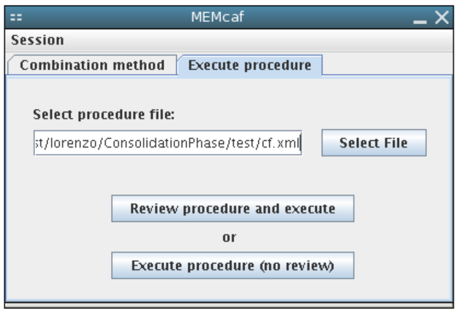
(v) The procedure review dialog allows to see and modify a procedure file before executing the data combination.

[8] cometa.exist.MetadataDB



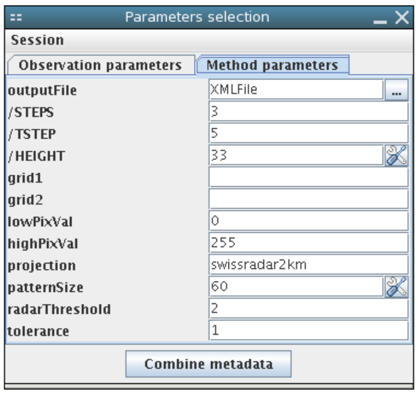
Select a combination method: AMF Event

(i)



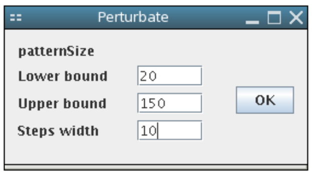
Select procedure file: st/lorenzo/ConsolidationPhase/test/cf.xml

(ii)



outputFile	XMLFile
/STEPS	3
/TSTEP	5
/HEIGHT	33
grid1	
grid2	
lowPixVal	0
highPixVal	255
projection	swissradar2km
patternSize	60
radarThreshold	2
tolerance	1

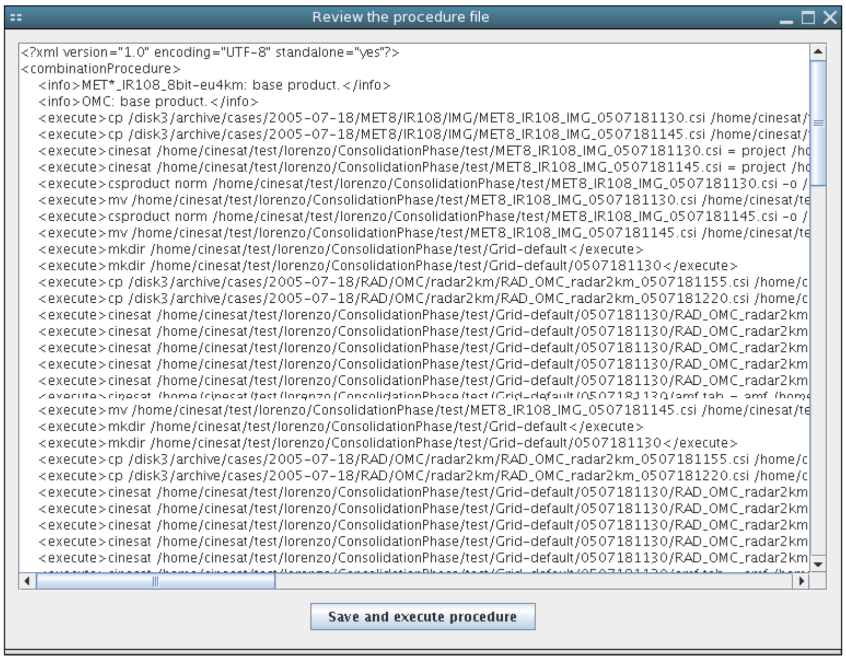
(iii)



patternSize

Lower bound	20
Upper bound	150
Steps width	10

(iv)



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<combinationProcedure>
   <info>MET*_IR108_8bit-eu4km: base product.</info>
   <info>OMC: base product.</info>
   <execute>cp /disk3/archive/cases/2005-07-18/MET8/IR108/IMG/MET8_IR108_IMG_0507181130.csi /home/cinesat/
   <execute>cp /disk3/archive/cases/2005-07-18/MET8/IR108/IMG/MET8_IR108_IMG_0507181145.csi /home/cinesat/
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/MET8_IR108_IMG_0507181130.csi = project /ho
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/MET8_IR108_IMG_0507181145.csi = project /ho
   <execute>csproduct norm /home/cinesat/test/lorenzo/ConsolidationPhase/test/MET8_IR108_IMG_0507181130.csi -o /
   <execute>mv /home/cinesat/test/lorenzo/ConsolidationPhase/test/MET8_IR108_IMG_0507181130.csi /home/cinesat/te
   <execute>csproduct norm /home/cinesat/test/lorenzo/ConsolidationPhase/test/MET8_IR108_IMG_0507181145.csi -o /
   <execute>mv /home/cinesat/test/lorenzo/ConsolidationPhase/test/MET8_IR108_IMG_0507181145.csi /home/cinesat/te
   <execute>mkdir /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default</execute>
   <execute>mkdir /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130</execute>
   <execute>cp /disk3/archive/cases/2005-07-18/RAD/OMC/radar2km/RAD_OMC_radar2km_0507181155.csi /home/c
   <execute>cp /disk3/archive/cases/2005-07-18/RAD/OMC/radar2km/RAD_OMC_radar2km_0507181220.csi /home/c
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/amf.tab = amf /home
   <execute>mv /home/cinesat/test/lorenzo/ConsolidationPhase/test/MET8_IR108_IMG_0507181145.csi /home/cinesat/te
   <execute>mkdir /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default</execute>
   <execute>mkdir /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130</execute>
   <execute>cp /disk3/archive/cases/2005-07-18/RAD/OMC/radar2km/RAD_OMC_radar2km_0507181155.csi /home/c
   <execute>cp /disk3/archive/cases/2005-07-18/RAD/OMC/radar2km/RAD_OMC_radar2km_0507181220.csi /home/c
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/RAD_OMC_radar2km
   <execute>cinesat /home/cinesat/test/lorenzo/ConsolidationPhase/test/Grid-default/0507181130/amf.tab = amf /hom
```

(v)

Figure 3.5: COMETA graphical user interface.

4

Use case: radar data forecasting by means of satellite derived displacement vectors

4.1 Description and goal

Weather forecasts predict the weather evolution for a few days and are based on complex weather prediction models that simulate the physical processes occurring in the atmosphere. Due to their complexity, these models require up to several hours to be computed and, therefore, they cannot be used for short range forecasts. For this reason, nowcasting activities are based on methods that require less time to be computed, that

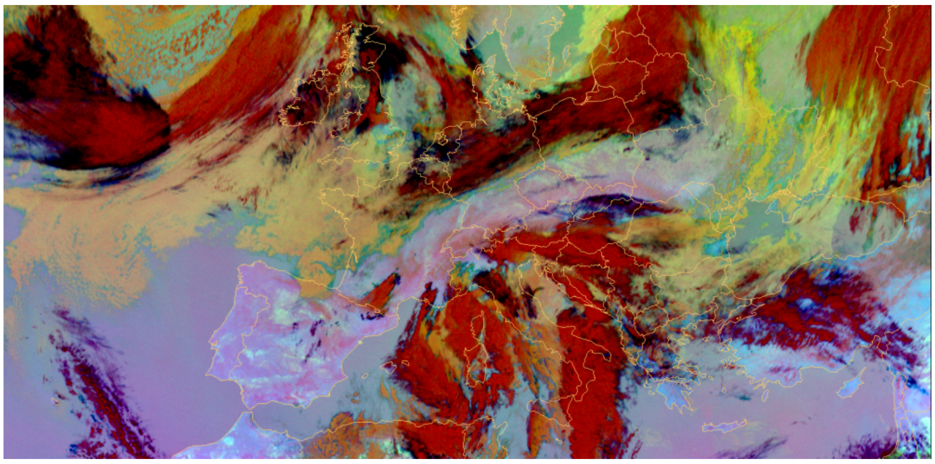


Figure 4.1: An example of an MSG microphysics product.

is to say, methods involving a smaller amount of data.

In estimating the displacement of a severe weather event (e.g. a thunderstorm accompanied by hail precipitations), a technique that is often used is the **time extrapolation** of radar images. A radar image can be interpreted as the intensity estimation of an ongoing precipitation: the extrapolation is obtained by observing how the rain field has moved in the past and by projecting this movement into the future, producing a forecasted image. This approach has proven valuable especially for flat regions and, as shown by [LZT04, Zaw05], for certain events it can yield better results that the numerical models for up to 6 hours. In mountainous regions, however, the orography plays a major role in the whole life cycle of thunderstorms.

In our use case an instantaneous radar retrieved rain field is forecasted by using satellite derived displacement vectors. The result is evaluated by comparing it against actually measured data at the following time step. We aim to answer the following questions:

1. Does the forecast quality change using different radar products?
2. Is it possible to tune this method to optimise the result for our region (Alps)?
3. How does the forecast quality decrease over time?

4.2 Products

This section describes the features of these data and the way we modeled it by means of metadata.

4.2.1 Satellite images

Satellite images come from geostationary satellites Meteosat-8 and Meteosat-9, both of them belonging to Meteosat Second Generation (MSG) [2]. The observations are carried by the imager SEVIRI which passively analyses 12 different wavelengths (channels), as

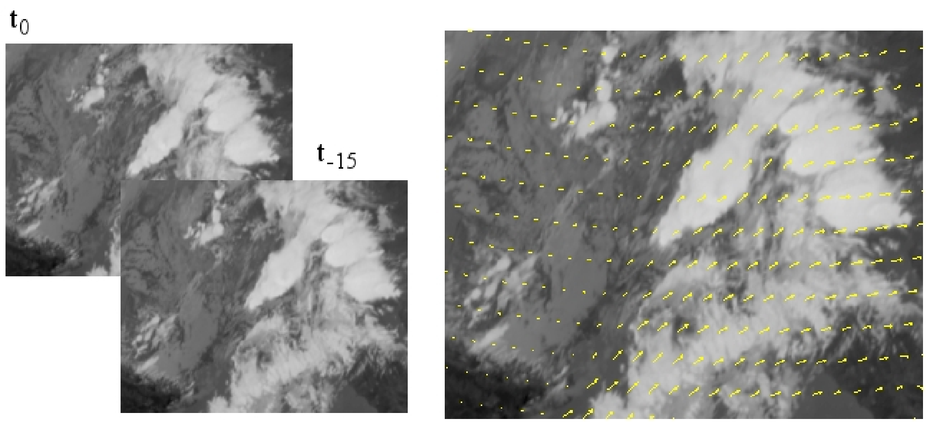


Figure 4.2: An atmospheric motion field derived by two successive images.

described in detail by [AAV04]. Figure 4.1 shows an example of an MSG product in which RGB colours are used by the forecasters to highlight the microphysics of clouds. In our application the black body temperature derived by the infrared channel centred at 10.8 μm (IR 10.8) is exploited to derivate an *Atmospheric Motion Field* (AMF), which is a vector field describing the displacement of clouds between two successive images through pattern matching or image correlation[1]. Figure 4.2 shows an example of an AMF (yellow vectors) overlayed on a IR 10.8 channel image.
IR 10.8 is located in the so called "infrared window" (most transparent to water vapour) and it is often used to estimate the cloud top temperature [KKE05]. Image calibration consists in linking pixels intensities to a physical parameter, in this case temperature: the colder the pixel, the higher the cloud. This is an important relationship, in particular for thunderstorms, which consist of very high and thick clouds whose top can reach the tropopause. The height assignment to satellite retrieved atmospheric motion vectors is a topic which is currently being heavily investigated by several studies, as reported by [AAV07].

Listing 4.1 and Listing 4.2 show the XML files we created to model the products described above. Listing 4.1 constitutes a data type from which all satellite images should inherit, whereas Listing 4.2 specifies the features of IR 10.8 images. The use of the attribute `base` (described in 3.2.2) indicates that IR 10.8 images inherit the features of satellite images.

```
<product productID="Satellite">
    <dataFeatures>
        <dataParam name="deltaTime" value="10"/>
        <dataParam name="archive" value="/disk3/archive/cases"/>
    </dataFeatures>
    <additionalInformation>
        <addInfoParam name="createdBy" value="cll"/>
        <addInfoParam name="createdOn" value="20080116"/>
    </additionalInformation>
</product>
```

Listing 4.1: Metadata for satellite images.

[1]A new image is issued every 15 minutes

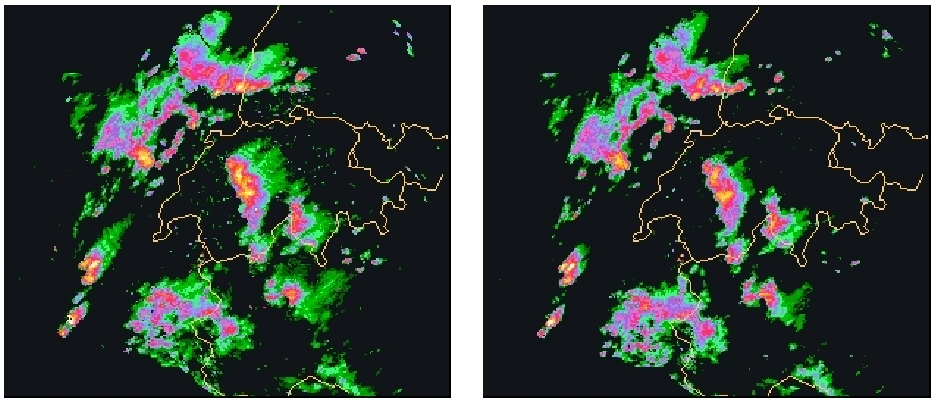


Figure 4.3: OMC (left) and PJC (right) images for 18th July 2005, 11.45 UTC.

```
<product productID="MET*_IR108_IMG">
    <dataFeatures base="Satellite">
        <dataParam name="channel" value="IR108"/>
        <dataParam name="frequency" value="15"/>
        <dataParam name="db" value="cinesat dblist \"MET*_IR108_IMG_*\""/>
    </dataFeatures>
    <additionalInformation>
        <addInfoParam name="createdBy" value="cll"/>
        <addInfoParam name="createdOn" value="20071205"/>
    </additionalInformation>
</product>
```

Listing 4.2: Metadata for satellite images, channel IR 10.8 .

4.2.2 Radar images

Several meteorological radar applications aim to produce the best estimate of precipitation rate at the ground level. This information is often exploited in the field of very short range rain forecasts. MeteoSwiss owns and manages three radars: La Dôle (near Geneva and close to the french border) Albis (near Zürich) and Monte Lema (in the southern Switzerland, Ticino). Each radar needs little less than 5 minutes to perform a complete scan and new radar products are generated with a frequency of 5 minutes [AAV98]. As part of our project we used two different radar products:

Overview Max Composite (OMC) is a class of images produced by applying a projection of the maximum reflectivity in vertical direction [AAV98]. The composite image integrates the data coming from La Dôle, Albis and Monte Lema.

Rain Composite (PJC) products provide the best estimation of the precipitation at ground level [AAV98]. With respect to OMC, Rain products undergo several processing to improve the estimation (e.g. ground clutter elimination). As for OMC, the composite image integrates the data coming from La Dôle, Albis and Monte Lema.

Both OMC and PJC products furnish a rain-rate estimation, their unit of measure is therefore mm/h. This information is coded on 4 bits, which corresponds to 16 possible values that are used to represent 16 intervals on a logarithmic scale as shown by Table 4.1. An example of an OMC and a PJC image is provided by Figure 4.3, where greenish

Class	Rain-rate $[mm/h]$
00	< 0.16
01	0.16 - 0.25
02	0.25 - 0.40
03	0.40 - 0.63
04	0.63 - 1.00
05	1.00 - 1.60
06	1.60 - 2.50
07	2.50 - 4.00
08	4.00 - 6.30
09	6.30 - 10.0
10	10.0 - 16.0
11	16.0 - 25.0
12	25.0 - 40.0
13	40.0 - 63.0
14	63.0 - 100
15	> 100

Table 4.1: Rain-rates for radar data. Source: [AAV98].

pixels correspond to low rain-rates and yellow / white pixels to very high rain-rates. The spatial resolution of both PJC and OMC is of 269 × 305 pixels, each of which having an edge of size 2 km [AAV98].

As for satellite images, also for radar products we defined a “virtual” parent product specifying some common features, and two child products for OMC and PJC respectively.

```
<product productID="Radar2km">
    <dataFeatures>
        <dataParam name="radars" value="ADL"/>
        <dataParam name="projection" value="swissradar2km"/>
        <dataParam name="frequency" value="5"/>
        <dataParam name="archive" value="/disk3/archive/cases"/>
    </dataFeatures>
    <additionalInformation>
        <addInfoParam name="createdBy" value="cll"/>
        <addInfoParam name="createdOn" value="2008/01/08"/>
    </additionalInformation>
</product>
```

Listing 4.3: Metadata for radar images, parent product.

Listing 4.3 shows the metadata for the radar parent product, named Radar2km. It specifies the frequency with which the products are generated (5 minutes) as well as the name of the projection (swissradar2km). Listing 4.4 and Listing 4.5 represent the metadata of OMC and PJC products; the base attribute of dataFeatures indicates that the parent product's ID is Radar2km. The actual existence of a product named Radar2km is verified by the database trigger ProductReferentialIntegrity[2], described in Subsection 3.1.2.

[2]cometa.exist.triggers.ProductReferentialIntegrity

```
<product productID="OMC">
    <dataFeatures base="Radar2km">
        <dataParam name="abbreviation" value="OMC"/>
        <dataParam name="calibration" value="METER 0.001 0"/>
        <dataParam name="db" value="cinesat dblist \"RAD_OMC*\""/>
    </dataFeatures>
    <additionalInformation>
        <addInfoParam name="createdBy" value="cll"/>
        <addInfoParam name="createdOn" value="2008/01/08"/>
    </additionalInformation>
</product>
```

Listing 4.4: Metadata for OMC radar product.

```
<product productID="PJC">
    <dataFeatures base="Radar2km">
        <dataParam name="abbreviation" value="PJC"/>
        <dataParam name="calibration" value="METER 0.001 0"/>
        <dataParam name="db" value="cinesat dblist \"RAD_PJC*\""/>
    </dataFeatures>
    <additionalInformation>
        <addInfoParam name="createdBy" value="cll"/>
        <addInfoParam name="createdOn" value="2008/01/08"/>
    </additionalInformation>
</product>
```

Listing 4.5: Metadata for PJC radar product.

4.3 The AMF combination method

The combination method described in this Chapter is modeled by the metadata of Listing 4.6. It is composed by two observations and several method parameters.

1. The first observation is of type MET*_IR108_IMG and it denotes the data used to compute the AMFs. It contains two parameters – from and to – that specify which event has to be considered (see 2.1.2).
2. The second observation accepts three different products: OMC, PJC or MET*_IR108_IMG and it designates which kind of data has to be forecasted.

```
<method methodID="AMF Event">
    <parameters>
        <observation type="MET*_IR108_IMG">
            <simpleParam name="from" type="string" default="YYMMDDHHMM"/>
            <simpleParam name="to" type="string" default="YYMMDDHHMM"/>
        </observation>
        <observation type="OMC,PJC,MET*_IR108_IMG"/>
        <simpleParam name="outputFile" type="XMLFile"/>
        <simpleParam name="steps" type="int" default="3"/>
        <simpleParam name="tstep" type="int" default="5"/>
        <simpleParam name="height" type="int" default="33"/>
        <simpleParam name="grid1" type="string" default=""/>
        <simpleParam name="grid2" type="string" default=""/>
        <simpleParam name="lowPixVal" type="int" default="0"/>
        <simpleParam name="highPixVal" type="int" default="255"/>
        <simpleParam name="projection" type="string" default="swissradar2km"/>
        <simpleParam name="patternSize" type="int" default="60" perturbate="true"/>
        <simpleParam name="radarThreshold" type="int" default="2"/>
        <simpleParam name="tolerance" type="int" default="1"/>
        <compoundParam name="timeSpan" action="deltaTime" arguments="tstep,steps"/>
    </parameters>
    <combinationProcedure>
        <ref class="AMFEvent"/>
    </combinationProcedure>
</method>
```

Listing 4.6: Metadata for the AMF combination method.

The AMF combination method has several parameters, among them the most important are:

outputFile indicates the combination procedure file. Furthermore, the directory where this file is located is used as working directory during the metadata and data combination phases.

steps determine how many forecasted images are generated.

tstep is the number of minutes between two consecutive forecasted images.

patternSize refers to an important parameter for the computation of the AMF. Its value indicates the size (expressed in number of pixels) of the window used by the AMF algorithm to perform the pattern matching between two consecutive images. The value of this parameter plays a major role in the algorithm performances (more details concerning this aspect can be found in Section 4.6 and in [Sch06]).

radarThreshold is the number of rain-rate classes[3] to be ignored in the radar image. It order to track only the motion of intense precipitation cells, it is possible to remove the lower rain-rates classes by setting their intensity to the background value.

tolerance indicates the tolerance for the quality index (see details in 4.4).

timeSpan is a compound parameter, i.e. a parameter depending on other simple parameters. timeSpan computes the reference time for the forecast evaluation. Suppose, for example, that steps is 3 and tstep is 10 minutes. Three new images are generated for the instants $t_0 + 10$, $t_0 + 20$ and $t_0 + 30$; only the last one ($t_0 + 30$) is considered for the forecast evaluation (see 4.4).

The set of method parameters and their values constitutes the *method configuration*. Furthermore, it is interesting to highlight the use of the attribute perturbate (described in Section 3.2.1) that allows to perturb the value of patternSize. As we shall see later in

[3]Rain-rate classes are listed in Table 4.1.

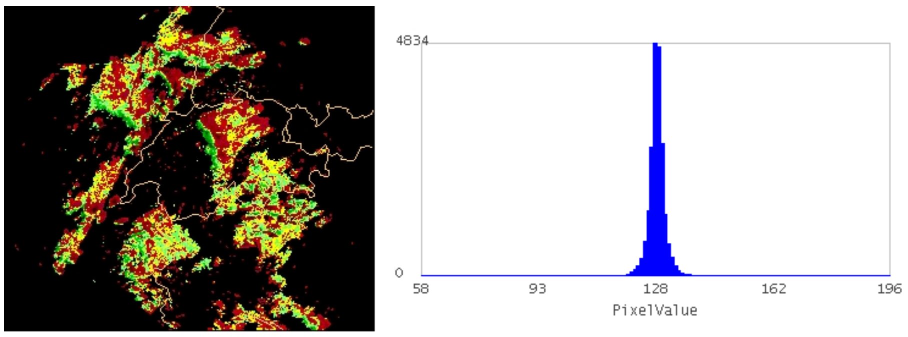


Figure 4.4: Forecast evaluation: the difference image and its histogram.

Section 4.6, this feature brought to light some interesting results.

As shown by Listing 4.6, the actual implementation of the combination method is given by AMFEvent[4] whose code can be found in Appendix C. This class concretely produces the sequence of actions that is written to the procedure file. In particular, AMFEvent has to handle the perturbation of the parameter patternSize.

4.4 Forecast evaluation

As we mentioned before, forecasted images are evaluated by comparing them against the actually measured data. In fact, in order to determine which method configuration produces the best result, we need a way to quantify the result's quality. Result evaluation is performed after the data combination phase, in the post processing phase (see Subsection 3.4.1). Two verification systems have been developed and are described in the following Subsections.

4.4.1 Quality flag

The *quality* function, given as input the forecasted and the measured images, returns a *quality flag* value within 0 and 1 indicating the percentage of correct pixels in the forecasted image. In this context, *"correct"* means pixels whose value in the forecasted image is exactly the same as the measured one, or it falls within an acceptable error range that can be selected by the user (see 4.3).

To compute the quality flag, a temporary image is firstly produced by performing a linear combination that subtracts the measured image from the forecasted one. Then, 128 is added to the value of each pixel of the newly created image and, therefore, pixels whose value is in $[128 \pm tolerance]$ are considered correct pixels. An example is given by Figure 4.4: yellow regions represent correct pixels, reddish regions are under estimated pixels and greenish regions are over estimated pixels. Once such an image has been

[4]memcaf.combine.AMFEvent

produced, the quality flag can be computed as follows:

$$q_{flag} = \frac{N_y}{N_y + N_r + N_g}$$

where N_y is the number of yellow pixels, N_g the number of greenish pixels and N_r the number of reddish pixels. The histogram of Figure 4.4 illustrates the distribution of pixels around the value 128. In this histogram the black pixels (background) are not considered as they would corrupt the meaning of the result. Note that, in the context of this project, the tolerance value is always ± 1, if not otherwise specified.

As we will discuss in more detail later, the quality flag value has to be considered with care. In fact, even small shift errors in the forecasted image can result in very bad quality scores, since our quality measure is very simple and does not use any pattern matching technique. In the conclusions, we provide some hints on how the quality flag could be improved. Despite this shortcoming, the results presented in this thesis remain valid because the quality flag has been used to compare results, thus using relative measures and not considering the quality flag's absolute value.

4.4.2 Visual inspection

Ideally, if the forecast technique worked perfectly, there would be no difference between the measured image and the forecasted one. Of course this is not the case and various kind of errors have an impact on the result. Through *visual inspection* atmospheric motion fields between the forecasted and the measured image are computed and the resultant vector field qualitatively evaluated. The goal of this approach is to detect systematic shifts that could be due, for example, to synchronisation issues or to panoramic distortions (e.g. parallax) [Ric94].
Further developments could account of qualitatively asses these errors through root mean square (RMS) analyses.

4.4.3 Output metadata

The combination method result consists of a directory hierarchy, so that each directory contains the files corresponding to exactly one experiment (i.e. the result of the data combination with a specific set of parameters). For example, imagine the patternSize parameter is perturbed for the 5th of July 2005, at 12:15 UTC. Then, the directory 0507181215/Pattern50/ contains the files where patternSize is 50, 0507181215/Pattern60/ the files where patternSize is 60 and so on. In all these directories, a file named result.xml summarizes the parameter values for the current experiment: Listing 4.7 shows an example of such a file.

Event date	**C**	**S**
18th July 2005	×	
6th July 2006	×	
12th July 2006	×	
3d March 2006		×
8th March 2006		×
25th October 2007		×

Table 4.2: Case studies: **C** stands for convective, **S** for stratiform.

```
<result>
    <grid>default</grid>
    <patternSize>50</patternSize>
    <percentCold>50</percentCold>
    <radarThreshold>2</radarThreshold>
    <tolerance>1</tolerance>
    <quality>0.697952728</quality>
</result>
```

Listing 4.7: Output metadata, the file result.xml.

4.5 Case studies

So far we have seen how data and processing information is modeled and how the result's quality is evaluated. This Section illustrates the meteorological events taken into account and the methodology used for the experiments.

4.5.1 Selected events

Precipitation events can be classified in two categories with different features: *convective precipitation* and *stratiform precipitation*. Intuitively, it can be said that convective precipitation tends to be very intense and isolated in space and time, while stratiform precipitation is characterised by a lighter, widespread and long-lasting rain [10]. As shown by Table 4.2, we selected six precipitation events: although the difference between convective and stratiform precipitation is not always clear, it could be stated that three of them are strongly convective, while the remaining ones consist in more widespread, stratiform precipitations.

4.5.2 Methodology

Before executing the radar forecasting experiments described in Section 4.1, some satellite - satellite forecast tests have been performed. IR 10.8 images have been forecasted for 15 minute steps, based on AMFs computed on the two previous images (also coming from channel IR 10.8). The goal of this phase was to determine a reference quality we could expect from the forecast of radar data.

The forecast method has then been applied to radar data, repeating the following procedure through the observation timespan with a frequency of 15 minutes:

1. Retrieve the latest consecutive IR 10.8 satellite images and compute the corresponding AMF.
2. Apply the previously derived AMF to the radar data (OMC or PJC) in order to produce the forecast for a 5 minutes step.
3. Evaluate the result by comparing it against the measured data.

This procedure has been applied to the six events listed in Table 4.2 for both OMC and PJC products.
Third, the patternSize perturbation has been executed for both satellite and radar forecasted data, with values varying from 10 to 150 pixels.
Finally, radar data has been forecasted up to 120 minutes into the future, in order to estimate how quality decreases over time.

4.6 Results discussion

According to the methodology described above, a number of experiments have been carried out. The current Section discusses the results of these experiments.

4.6.1 OMC versus PJC

The six cases taken into account (see Table 4.2) have shown that the forecast of OMC products yields better results with respect to PJC: this happened in 5 cases out of 6.
It is interesting to remark that for convective precipitation events the scores obtained by OMC forecasted data are clearly better than the PJC ones. For example, for the 18th July 2005 event, OMC produces a result with 8% more "correct" pixels (in the average) with respect to PJC. As shown by Figure 4.5, this difference is far less significant for stratiform precipitations like the one of 3d March 2006, where OMC is just 2% better than PJC in the average.

The tolerance value used in these experiments is of ± 1. Obviously, incrementing the tolerance would produce better scores, up to 90% of correct pixels and more. This study, however, is focused in the comparison between OMC and PJC products, rather than on the analysis of absolute values obtained by both of them. With regard to this result, it is important to note that PJC images have usually more background (black) pixels than OMC images. The PJC product is the best estimation of ground precipitation and it passes through a sequence of algorithms that eliminate the noise, which is not the case for OMC [AAV98]. The lower number of non-background pixels in PJC images could have an impact on the value produced by our quality function.

According to these first observations, OMC has been chosen for the following experiments.

4.6.2 Forecast tuning

The patternSize tuning has been carried for both satellite and radar forecasted data. The results are quite interesting. Smaller pattern sizes (between 30 and 40 pixels) produce better results for the forecast of satellite images, then, with the increase of the pattern

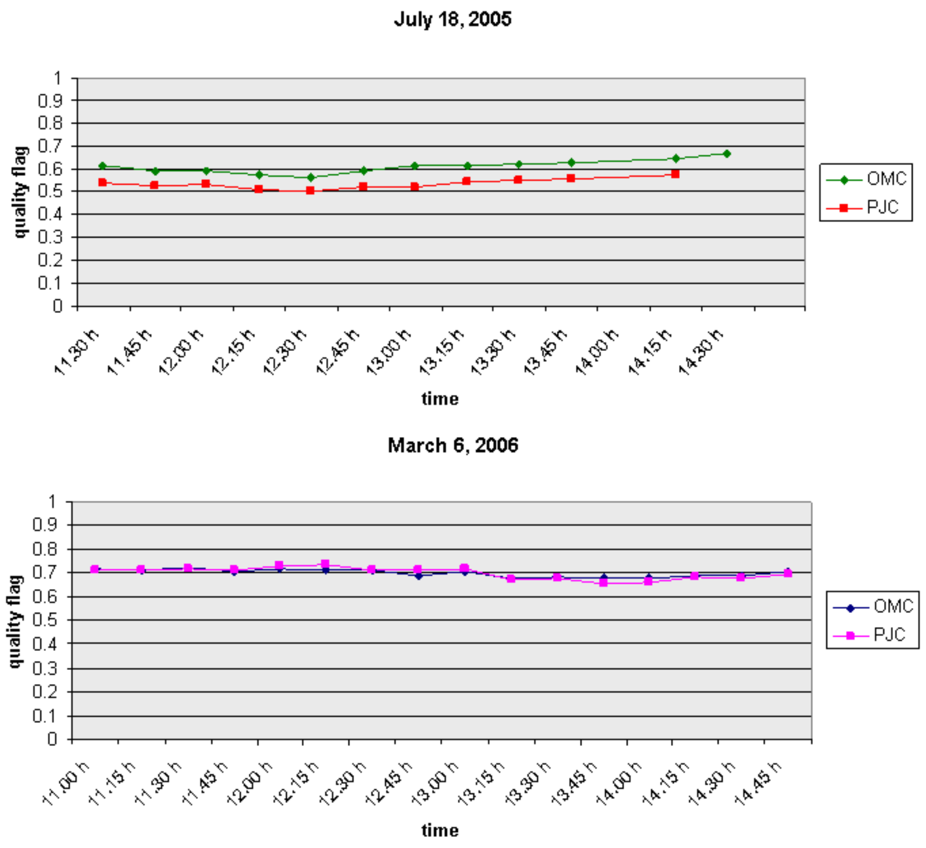


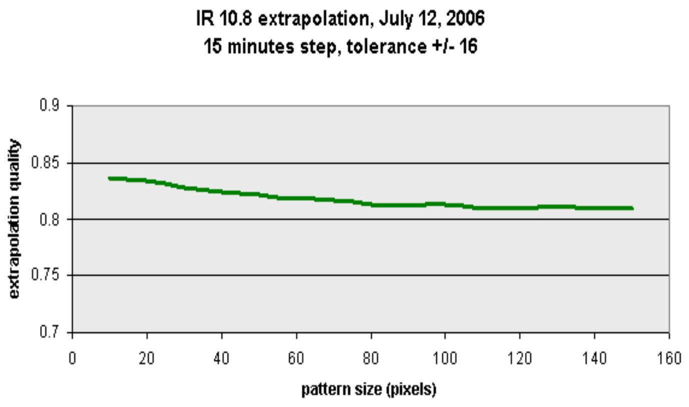
Figure 4.5: Quality assessement over time; 2 products (OMC, PJC) observing 2 events.

size, the quality diminishes. On the other hand, the trend is different for radar forecasted data: small patterns give worse results than larger ones. A comparison between satellite and radar data forecast is illustrated by Figure 4.6.

It is important to point out that the relation between pattern size and result quality is not always so clear as for the 12th July 2006 event. Furthermore, the gain in term of absolute value for the quality flag is small: for radar data, the difference between the worst and the best score is of about 1%. Despite this fact, this analysis can play an important role if considered under the **cost - benefit** point of view. The patternSize parameter plays a major role in the running time of the AMF algorithm when it is run using the pattern matching modality (see [Sch06]). The pattern matching lookup is repeated for all the gridpoints of the image (the grid is also a parameter of the algorithm, as shown by Listing 4.6). For each grid point, it is realistic to suppose that an operation is executed for every pixel of the pattern [Ric94]. If the pattern size is n and the number of gridpoints is m, the computational complexity of the algorithm is therefore

$$O(n^2 \cdot m)$$

The size of the pattern has an important weight in this relation. As a consequence, discovering that very large patterns produce only slightly better results can help in optimising the use of computational resources.



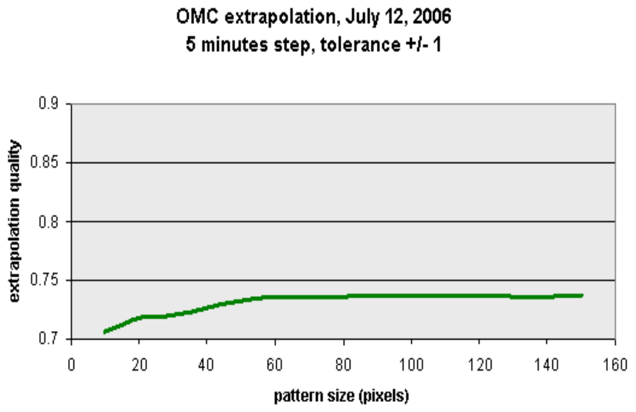


Figure 4.6: Upper image: satellite forecasted images based on satellite retrieved AMF. Lower image: radar forecasted images based on satellite retrieved AMF.

Concerning the OMC product forecast, the considered events showed that a good value for patternSize is 60 pixels. This result, however, is mitigated by the fact that this observation is not always so clear and the results vary from one event to another.

4.6.3 Forecast quality decay over time

The last question this work deals with is how forecast quality decreases over time. The result obtained is coherent for all the considered events.

Figure 4.7 refers to the event of 8th March 2006 and shows how quality is related to the forecast time. The blue dots represent the forecast quality, while the orange line is a *logarithmic regression* over the same data set. The logarithmic regression fits the quality decay with high correlation ($R^2 > 0.9$).
The logarithmic regression function does not look like a "usual" logarithmic function. A general logarithmic function is of the form $y = a \cdot ln(x) + b$; the one we are considering here, however, has a negative value for the coefficient a, which explains its position in the first quadrant of the cartesian coordinates system.

In all the studied events, the threshold of 50% "correct" pixels is reached for forecasts of about 15 minutes. However, remember that different tolerance values would strongly influence this figure. As for the previous results, quality values do not have to be

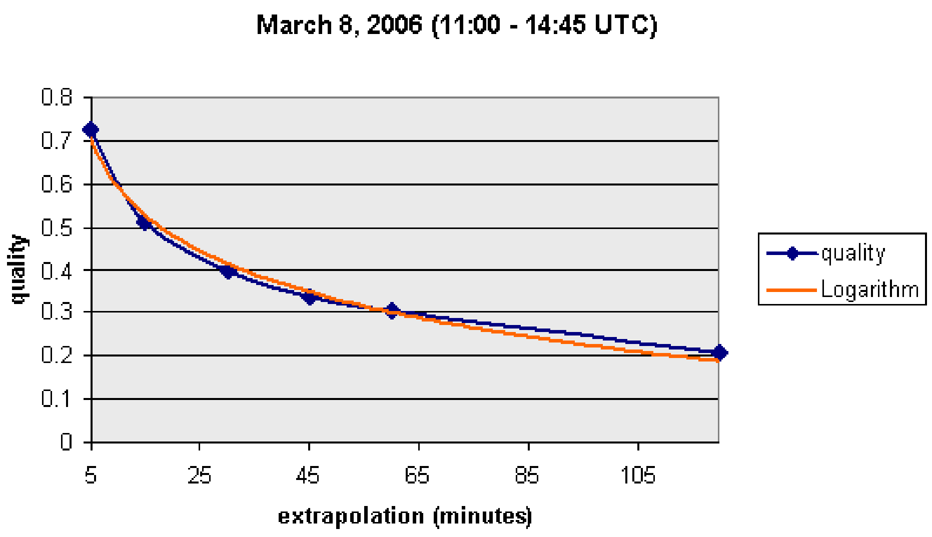


Figure 4.7: Quality assessement against forecast period for one event and one product (OMC).

considered as absolute values but, rather, they provide an indication on the observed trends.

5 Conclusions

5.1 Results

A flexible framework to characterise meteorologically relevant data in order to support R&D activities in the field of nowcasting has been designed and implemented. COMETA consists in an XML database and a set of Java classes implementing the framework's logic. Metadata files describe the features of different data types and processing information; functionalities are provided to verify the compatibility of data (*metadata combination* phase) and to combine the data (*data combination* phase).

The implementation of a use case led to the following results. It has been shown that this framework is a valuable tool for characterising and combining remote sensing data. COMETA produced some first interesting results concerning instantaneous radar retrieved rain field forecasts by means of satellite derived displacement vectors. Experiments exploiting the *parameter perturbation* facility have shown that, for the six case studies considered, the forecast of radar product Overview Max Composite (OMC) produces better results than the Rain Composite product (PJC), especially for convective precipitation events. Furthermore, the AMF algorithm seems to attain a good cost-benefit compromise for pattern sizes around 60 pixels. The behaviour of the quality decay against the increasing forecast time has been studied.

5.2 Outlook

Chapters 2 and 3 illustrated how the COMETA framework has been conceived and implemented, while Chapter 4 presented a concrete use case. Similarly, this Section first discusses some possible developments concerning the framework itself and, afterwards, it exposes a few considerations about the radar and satellite data combination use case.

5.2.1 Framework extensions

eXist XML database has proven valuable for storing metadata files. However, the creation of metadata to describe product features and processing information has still to be done manually, by writing new metadata files (as described in Section 3.2). A very useful extension would consist of a guided procedure, possibly accompanied by a graphical user interface, to ease the generation of metadata files.

An additional direction for future works is the creation of a specific syntax (e.g. a dedicated XML namespace) to describe the steps composing a metadata combination method. At the moment, the implementation of a new combination method requires the writing of a new Java class. Ideally, it should be possible to describe a method without requiring the user to have Java knowledge. This extension, although fairly difficult to realise, would largely increase the user friendliness of COMETA.

Another improvement deals with the framework's parameter perturbation feature. As long as one or two parameters are perturbed, their complexity can be handled by the programmer who writes the method's code. However, it is often interesting to perturb several parameters within the same experiment in order to study their correlation. This approach leads to a *combinatorial problem* where all the varying parameters have to be combined with each others. The combinatorics nature of the problem poses two great challenges: first, the management of several perturbed parameters cannot be done manually by the programmer and the framework should provide a way to help the developer in this task. Second, a combinatorial problem has a factorial computational complexity, which could require too much time to be treated.

5.2.2 Use case: improving first results

Predicting the displacement of a radar detected rain field in the alpine region is an ambitious task. The approach proposed in Chapter 4 is meant to provide an example of how COMETA can be used with a multi-sensor approach. First results show consistent signals and are encouraging.

The definition of a more adequate quality flag is important for absolute assessments. This could still be image based or entity based, like CRA [1]. Although very robust and much more refined, CRA is complex to implement. An easier way to improve the quality flag accuracy would be to examine the properties of the difference image, like histogram mean and standard deviation.
A following step would be the RMS analysis of the residual displacement vectors computed on forecasted and measured images and accounting for parallax induced errors.
Another improvement would be to apply a smoothing filter on the difference image, before computing the quality flag. This operation would reduce the influence of shift errors, in case of strong gradients affecting the radar image.

Additional work would involve the study of how radar derived displacement vectors can predict the motion of radar detected rain fields, although orographic effects and a limited correlation length have not to be neglected. Due to the presence of the Alps, the results we would expect from such a method are limited; nevertheless, it is important to study and quantify the outcomings of this approach.

An interesting further development could derive from the combination of the forecast technique presented in Chapter 4 with the Thunderstorm Radar Tracking (TRT), an algorithm developed at MeteoSwiss. TRT is based on the OMC product and it is able to detect severe thunderstorm cells and, by examining their motion in the past, it predicts their displacement in the future. Various kinds of cooperation between the two algorithms can be foreseen.

A
Common Acronyms

AMF Atmospheric Motion Field

API Application Programming Interface

CineSat A real-time short-range weather forecast software.

CRA Contiguous Rain Areas.

DOM Document Object Model

ECMWF European Centre for Medium-Range Weather Forecasts (Reading, UK)

FGDC Federal Geographic Data Committee, USA

JAXB Java Architecture for XML Binding

JDK Java Development Kit

JVM Java Virtual Machine

MeteoSwiss Federal Office of Meteorology and Climatology MeteoSwiss, Switzerland

MSG Meteosat Second Generation

NOAA National Oceanic and Atmospheric Administration, USA

NWG Nowcast Working Group

OMC Overview Max Composite (radar product)

PJC Ground precipitation estimate, composite image (radar product)

RASA Radar and Satellites team of MeteoSwiss

SAF/NWC Satellite Application Facility on support to Nowcasting and Very Short-Range Forecasting

SAX Simple API for XML

TRT Thunderstorm Radar Tracking

VSRF Very Short-Range Forecasting

WMO World Meteorological Organisation

WWRP World Weather Research Program

XML Extensible Markup Language

B
License of the Documentation

C
CD-ROM and website of the project

On the CD-ROM of the project you will find:

- The source code of COMETA.
- The API (Javadoc) of COMETA.
- The binaries and sources of this documentation, as well as the pdf version.

Figure C.1 provides a tree view of the CD-ROM.

The content of the CD-ROM can also be downloaded from the official website of the project: `http://www.sosto.net/cometa`.

```
|-- Cometaproject                       // Framework NetBeans project,
|                                       // API, Javadoc
|-- Metadata                            // Contents of the Metadata DB
|
|-- Documentation
|   `-- FirstMidTermReport              // First report (pdf)
|                                       // and slideshow
|   `-- SecondMidTermReport             // Second report (pdf)
|   `-- FinalPresentation               // Final presentation
|                                       // (slideshow)
|   `-- Thesis                          // Binaries (pdf, ps, etc.) and
|       |-- appendix                    // sources of this document
|       |-- figures                     // Sources of the figures.
|       |-- chapters
|
|-- Software                            // The software used.
    |-- JavaJDK                         // JavaJDK 1.5
    |-- Netbeans                        // Netbeans IDE
    |-- eXist                           // XML database
```

Figure C.1: Tree view of the content of the CD-ROM

References

[AAV98] AAVV. *Operational Use of Radar for Precipitation Measurement in Switzerland*. VDF Hoschschulverlag AG der ETH Zürich, 1998.

[AAV03] AAVV. Nowcasting and the WWRP Report of WWRP Ad Hoc Nowcast Working Group. Technical report, NWG, WWRP, 2003.

[AAV04] AAVV. A short introduction to Meteosat Second Generation (MSG). Technical report, Eumetsat, Darmstadt, Germany, 2004.

[AAV07] AAVV. Atmospheric Motion Vectors and IWWG Matters. *Report of the 35th Meeting of the Coordination Group for Meteorological Satellites*, 2007.

[AS00] J. T. Anderson and M. Stonebraker. Sequoia 2000 metadata schema for satellite images. Technical report, EECS Departement, University of California in Barkeley, USA, 2000.

[Cle07] L. Clementi. MEMcaf: Meteorological Metadata Combination Framework, First mid-term report. Technical report, University of Fribourg and MeteoSwiss, 2007.

[Cle08] L. Clementi. MEMcaf: Meteorological Metadata Combination Framework, Second mid-term report. Technical report, University of Fribourg and MeteoSwiss, 2008.

[GG07] G. Galli and I. Giunta. Inter-comparison of remote-sensing observations: a new relational metadata framework on support to meteorological R&D, work proposal for the Master thesis of Lorenzo Clementi. Technical report, MeteoSwiss, 2007.

[KKE05] S. Q. Kidder, J. A. Kankiewicz, and K. E. Eis. Meteosat Second Generation Cloud Algorithms for Use at AFWA. Technical report, DoD Center for Geosciences and Atmospheric Research, Colorado State University, Colorado, USA, 2005. [Retrieved February 22nd, 2008, from `http://amsu.cira.colostate.edu/kidder/BACIMO_2005.pdf`].

[LZT04] C. Lin, I. Zawadzki, and B. Turner. Precipitation forecast based on numerical weather prediction models and radar nowcasts. Technical report, Departement of Atmospheric and Oceanic Sciences, McGill University, Montréal, Québec, 2004.

[NIS04] National Information Standards Organization NISO. *Understanding metadata*. NISO Press, USA, 2004.

[OM03] E. Ort and B. Mehta. Java Architecture for XML Binding (JAXB). Technical report, Sun Microsystems, 2003. [Retrieved February 20th, 2008, from `http://java.sun.com/developer/technicalArticles/WebServices/jaxb/`].

[Ric94] John A. Richards. *Remote Sensing Digital Image Analysis, An introduction.* Springer-Verlag, 1994.

[Sch06] J. Scheibler. *CineSat A real-time short-range weather forecast system. Software user manual.* GEPARD, Austria, 2006.

[TK04] U. D. Turdukulov and M.-J. Kraak. Visual exploration for Nowcasting of precipitating convective clouds in time series of remote sensing data. *ESA-EUSC 2006 Workshop Proceedings [online]*, 2004. [Retrieved February 29th, 2008, from `http://earth.esa.int/rtd/Events/ESA-EUSC_2006/Poster/Po71_Turdukulov.pdf`].

[Wil04] J. W. Wilson. Precipitation nowcasting: past, present and future. Technical report, National Center for Atmospheric Research, Boulder Colorado USA, 2004. [Retrieved February 1st, 2008, from `http://www.bom.gov.au/bmrc/basic/old_events/hawr6/qpf/WILSON_KEYNOTE.pdf`].

[WMO05] WMO. A Proposal for Changes to the WMO Core Metadata Profile. Technical report, WMO, CBS Inter-programme expert team on metadata implementation, Beijing, China, 2005.

[Zaw05] I. Zawadzki. Nowcasting of precipitation. Technical report, Departement of Atmospheric and Oceanic Sciences, McGill University, Montréal, Québec, 2005.

Referenced Web Resources

[1] CRA (entity-based) verification. http://www.bom.gov.au/bmrc/wefor/staff/eee/verif/CRA/CRA_verification.html (accessed March 11th, 2008).

[2] Eumetsat. http://www.eumetsat.int/ (accessed February 18th, 2008).

[3] eXist XML database. http://exist.sourceforge.net/ (accessed December 7th, 2007).

[4] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (accessed February 19th, 2008).

[5] NetBeans IDE. http://www.netbeans.org/ (accessed February 21st, 2008).

[6] Solaris 10 Operating System. http://www.sun.com/software/solaris/index.jsp (accessed February 21st, 2008).

[7] Ultra 25 SUN workstation. http://www.sun.com/desktop/workstation/ultra25/specs.xml (accessed February 21st, 2008).

[8] Geospatial Metadata. http://www.fgdc.gov/metadata (accessed December 5th, 2007).

[9] JAXB Reference implementation. https://jaxb.dev.java.net/ (accessed February 19th, 2008).

[10] A website about Alpine meteorology. http://www.meteomontebaldo.it/ (accessed February 28th, 2008).

[11] Consulting, writing, and research in XML and databases. http://www.rpbourret.com/ (accessed December 7th, 2007).

[12] SAFNWC. http://nwcsaf.inm.es/ (accessed March 6th, 2008).

[13] Wikipedia. http://www.wikipedia.org/ (accessed February 18th, 2008).

[14] eXist XML database. http://xerces.apache.org/xerces-j/ (accessed December 5th, 2007).

[15] XML:DB Initiative for XML Databases. http://xmldb-org.sourceforge.net/ (accessed February 19th, 2008).

[16] XML-RPC Home Page. http://www.xmlrpc.com/ (accessed February 21st, 2008).

[17] XML Schema recommendation from W3C. http://www.w3.org/TR/xmlschema-0/ (accessed November 17th, 2007).

[18] Introduction to XML Schema. `http://www.w3schools.com/schema/schema_intro.asp` (accessed November 22nd, 2007).

Index