Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Mercury: Using the QuPreSS reference model to evaluate predictive services

S. Martínez-Fernández [a,*], X. Franch [a], J. Bisbal [b]

[a] GESSI Research Group, Universitat Politècnica de Catalunya – Barcelona Tech (UPC), C/Jordi Girona 1–3, 08034 Barcelona, Spain
[b] INS Manuel de Cabanyes, Av. Francesc Macia 110–114, 08800 Vilanova i la Geltru, Barcelona, Spain

A R T I C L E   I N F O

Article history:
Received 12 March 2014
Received in revised form 16 November 2015
Accepted 17 November 2015
Available online xxxx

Keywords:
Tool development
Predictive services
Reference model
Forecast verification
Service monitoring

A B S T R A C T

Nowadays, lots of service providers offer predictive services that show in advance a condition or occurrence about the future. As a consequence, it becomes necessary for service customers to select the predictive service that best satisfies their needs. The QuPreSS reference model provides a standard solution for the selection of predictive services based on the quality of their predictions. QuPreSS has been designed to be applicable in any predictive domain (e.g., weather forecasting, economics, and medicine). This paper presents Mercury, a tool based on the QuPreSS reference model and customized to the weather forecast domain. Mercury measures weather predictive services' quality, and automates the context-dependent selection of the most accurate predictive service to satisfy a customer query. To do so, candidate predictive services are monitored so that their predictions can be eventually compared to real observations obtained from a trusted source. Mercury is a proof-of-concept of QuPreSS that aims to show that the selection of predictive services can be driven by the quality of their predictions. Throughout the paper, we show how Mercury was built from the QuPreSS reference model and how it can be installed and used.

1. Introduction

The success of service technologies has boosted a big offer of services covering many domains. Service customers do not need to worry about the development, maintenance, infrastructure, or any other issue related to the service operation. Instead, they only have to find and choose the most appropriate service offered by some service provider [1]. Therefore, it becomes necessary to assess which service is the most appropriate for fulfilling the customer's needs. Examples of such needs are the quality of service, reputation, cost, security, personalization, and locality.

Among all kinds of services, we focus on predictive (or forecasting) services. We define *predictive services* as those services whose main functionality is to show in advance a condition or occurrence about the future. Predictive services emerge in many domains such as stock market prices, bookmaker results, election polls, and sales forecasting. The quality of their predictions is of obvious importance to:

- citizens, because reliable predictions may significantly improve their decision-making;

* Corresponding author.
E-mail addresses: smartinez@essi.upc.edu (S. Martínez-Fernández), franch@essi.upc.edu (X. Franch), jbisbal@xtec.cat (J. Bisbal).
URL: http://www.essi.upc.edu/~smartinez/ (S. Martínez-Fernández).

http://dx.doi.org/10.1016/j.scico.2015.11.009
0167-6423/© 2015 Elsevier B.V. All rights reserved.

- service providers, because it affects their reputation and potentially their revenues;
- developers, if the prediction algorithm is embodied as a software component ready to be integrated into other systems (e.g., offered as a web service);
- and domain specialists (e.g., meteorologists, brokers, etc.), who want to understand when their prediction models behave better or worse.

An example that is really familiar to all of us is weather forecast. Weather conditions affect our decisions in daily routines such as deciding what to wear first thing in the morning or when to travel. To make these decisions, different services could be consulted. Examples of weather forecast services are the weather section on TV news or specialized websites that provide predictions over specific data (such as forecastadvisor.com) [2]. However, sometimes their predictions do not match or change over time as the date of interest approaches. In this context, a software engineering challenge arises: *given a portfolio of candidate predictive services, which one is expected to be the most accurate to satisfy the customer needs?*

Bearing this challenge in mind, we decided to create an academic software tool to experiment with the problem of selecting the "best" predictive service. Our goal was to prove that such problem could be solved by instantiating QuPreSS, a reference model for predictive service quality assessment [3]. Therefore, this paper presents *Mercury*, an academic tool based on QuPreSS that assesses weather predictive services based on the quality of their predictions. Among the variety of existing predictive domains, we choose weather forecast because its services are daily used by many people.

Throughout the paper, we show how Mercury was built. We focus on its requirements, the main design decisions, and how it is installed and used. Besides, Mercury is available on a virtual machine (see details on Section 5). By distributing Mercury on a virtual machine, the software can be reviewed and used, and other researchers can see how to instantiate QuPreSS for other predictive domains. It is important to remark that although Mercury is focused on a single predictive domain, it is the first step of the creation of a general validation framework for the problem of prediction quality in many predictive domains.

This paper is an extension of the paper presented in the 4th International Workshop on Academic Software Development Tools and Techniques (WASDeTT-4) [4]. This work additionally presents: a background on reference models, forecast verification tools and service monitoring; a five-step process to create tools based on QuPreSS; the libraries of Mercury in order to allow its installation and usage; access to complete documentation of Mercury; and improved lessons learned on how to avoid several problems that we faced during Mercury development, deployment and maintenance.

The paper is structured as follows. Section 2 addresses relevant theory about reference models, forecast verification, related tools, and service monitoring. Section 3 motivates the problem that QuPreSS solves and shows how QuPreSS can be instantiated. Section 4 describes the development as well as the main design and implementation decisions made in Mercury. Section 5 shows the technologies used in Mercury, its installation process, and several demonstration scenarios to learn how to use it. Section 6 shows lessons learned from the experience of Mercury building. Finally, Section 7 summarizes the paper and identifies a number of future directions.

2. Background

In this section, we respectively present a background of reference models and the two techniques that enable predictive service selection: forecast verification and service monitoring.

2.1. Reference models

"A reference model is a division of functionality together with data flow between the pieces. A reference model is a standard decomposition of a known problem into parts that cooperatively solve the problem" [5]. Reference models arise in mature domains in which experience has lead to a standard solution for the problem, e.g., the standards parts of a compiler, a database management system or a service-oriented environment, and how such parts work together to accomplish their collective purpose. Well-known examples of reference models are the OASIS reference model to develop service-oriented architectures [6] and FORMS, which is a reference model for distributed self-adaptive systems [7]. As it has been mentioned, Mercury is based on the QuPreSS reference model [3].

It is important to note that a reference model is a more abstract concept than a reference architecture. A reference architecture is "a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them" [5]. Thus, a reference model is not directly tied to any standards, technologies or other concrete implementation details [6]. Summarizing, a reference architecture is a set of domain concepts mapped onto a standard set of software components and relationships [8]. There are processes to create reference architectures, such as ProSA-RA [9].

2.2. Forecast verification

Forecast verification is the process of assessing the quality of a prediction by comparing it with its corresponding observation [10]. *Forecast quality* is the correspondence between forecasts and observations [10]. A forecast has high quality if it predicts the observed conditions well according to some aspect such as accuracy and skill.

The next subsections respectively explain in which predictive domains forecast verification has been applied, which methods and metrics exist to calculate forecast quality, and existing software and tools.

2.2.1. *Forecast verification in diverse predictive domains*

As discussed by Jolliffe et al. [11], forecast verification has been applied in diverse predictive domains, such as *weather forecasting* (a field of atmospheric science), *statistics*, *finance and economics*, *environmental and earth sciences* (e.g., prediction of earthquakes), *election forecasting*, and *medical and clinical* studies (e.g., drug design, and prediction of protein's issues). This situation has lead to several terms for the concept of forecast verification in different disciplines. In the weather forecasting context, it is referred to as forecast verification, whereas in other disciplines it is referred to as *forecast evaluation* [11] or even *ex post evaluation* in financial and economics forecasting [12].

In weather forecasting, forecast verification is "an indispensable part of meteorological research and operational forecasting activities" [13]. The awareness of this problem by the weather forecast community is increasing and even dedicated events are being organized (See 6th International Verification Methods Workshop[1]). Forecast verification is used to check that forecasts are improving over time, to determine when upgrades are needed and to help companies make decisions [13]. It has been reported its usage to measure the quality of algorithms in weather forecasting for big events such as the Sydney 2000 and Beijing 2008 Olympic Forecast and Research Demonstration Projects, and Sochi-2014 Winter Olympic Games [14–16].

In economics, *forecast evaluation* has been used not only to evaluate a single forecast, but also evaluate competing forecasts and even combine them to produce a superior composite forecast [17].

In election forecasting, rules to evaluate scientific approaches (e.g., polls, political stock markets and statistical models) have been designed and applied to gauge their accuracy [18]. Examples of election forecasting models evaluation have been reported in the United States, France and the United Kingdom [19].

In medicine, prognostic models should be clinically credible. They need to be demonstrated to be effective and forecast verification provide means to evaluate them. Several case studies have discussed how to *validate* a prognostic model [20,21].

Finally, it is worth to note other relevant predictive domains that have not reported the measurement of forecast quality, but perform predictions [3]: *quality of service prediction* with the final goal of improving composition of services and service's reliability, *automotive forecasting*, *prediction of flight delays*, *sales forecasting* for the calculation of safety stock and *results in betting shops* [22].

2.2.2. *Measuring forecast quality*

Forecast quality consists of the statistical description of how well the forecasts match the real observations. It provides important feedback on the predictive service. Forecast quality has many different attributes that can provide useful information: bias, reliability/calibration, uncertainty, sharpness/refinement, accuracy, association, resolution and discrimination [10]. Murphy defines these attributes important for the weather forecasting domain and indicate how to calculate them [10]. It must be noted that these attributes may slightly differ depending on the predictive domain. For instance, Lewis-Beck considers that an election forecasting instrument should be evaluated by four criteria: accuracy, lead, parsimony and reproducibility [19]. All these attributes provide useful information about the performance of a predictive service. No single measure is sufficient for judging and comparing forecast quality [11].

There are two main approaches to measure forecast quality: measures-oriented and distributions-oriented [10,23]. The former includes several metrics such as the mean absolute error, the mean-square error, and various skill scores. They only focus in one or two attributes of forecast quality (e.g., accuracy, which is the average correspondence between individual pairs of forecasts and observations). On the other hand, the latter is based on the joint distribution of forecast and observations (see the Murphy–Winkler framework [24]) and contain all attributes of forecast quality.

2.2.3. *Tools and software for forecast verification*

Pocernich [25] has recently made a survey about software and tools to support the forecast verification process. Among the most popular forecast verification tools are spreadsheets, statistical programming languages (e.g., R, SPSS, MATLAB), and institutional supported software. Next, we pay special focus on tools that allow their integration with other software.

The use of R packages (*verification*, *ensembleBMA*, *nnclust*, *pROC*, and *ROCR* [25]) to support forecast verification is becoming popular. For instance, the R package *ensembleBMA* [26] includes functions to calculate mean absolute error, continuous ranked probability score, and Brier score in a spatial context.

With regard to institutional supported software, the Hydrological Ensemble Prediction group of the US National Weather Service's Office of Hydrologic Development has developed the Ensemble Verification System (EVS) [27]. The source code of the EVS is available online [28]. Its metrics library enables forecast verification for the ensemble mean and forecast probabilities by providing multiple metrics (e.g., mean absolute error, relative mean error) that are specially interesting because they can be reused in other software.

Existing software and tools are useful to evaluate the quality of one single predictive service, whose predictions have been previously collected and observations are available. A next step, towards real-time automated forecast verification was

[1] http://www.ncmrwf.gov.in/verif2014/.

previously performed by Domenico [29]. Domenico proposed geosciences web service to integrate sources of data in order to perform later forecast verification with observations. However, only QuPreSS allows: automatically collecting forecasts from several heterogeneous predictive services and observations from a ground truth service; performing real-time forecast verification; and, supporting service customers to analyze which one is the best predictive service for them (see Section 3).

2.3. Service monitoring

Although forecast verification is a mature field and there have been many research efforts over the past decades, the proliferation of services has lead to a new use of forecast verification: the selection of the predictive service that provides the highest forecast quality. As a consequence, service monitoring becomes fundamental for forecast verification. Service monitoring consists of using a monitoring infrastructure that observes the behavior of (ground truth/predictive) services.

Monitoring is the way to provide users and system integrators the means to build confidence that a service, which is used and not owned and runs on a machine out of the users' control, delivers a function with the expected quality of service [30]. With the help of monitoring we can succeed in useful tasks such as verifying that a service invocation meets certain conditions, and offering access to monitored data for tools.

SALMon is a service-oriented system that has two services: the monitor service and the analyzer service [31]. The monitor service measures the values of dynamic quality attributes (such as response time and availability) in order to retrieve quality of service of a service-oriented system. SALMon supports passive monitoring and testing, it supports any type of service technology, it can be integrated in service-oriented systems, and new ways to measure quality of service can be added [31]. Up to our knowledge, SALMon is the only monitoring tool with the aforementioned advantages and that obtains real-time quality of service in service-oriented systems [31].

3. The QuPreSS reference model

This section presents the QuPreSS reference model. First, we show the prediction problem of selecting the predictive service that is expected to be the most accurate one to satisfy some given customer needs [3]. Second, we define a five-step process to develop a tool based on QuPreSS.

3.1. Problem statement

Given a portfolio of candidate predictive services, we aim to find the most accurate to satisfy some given customer needs. To this end, QuPreSS requires the following four inputs (see Fig. 1):

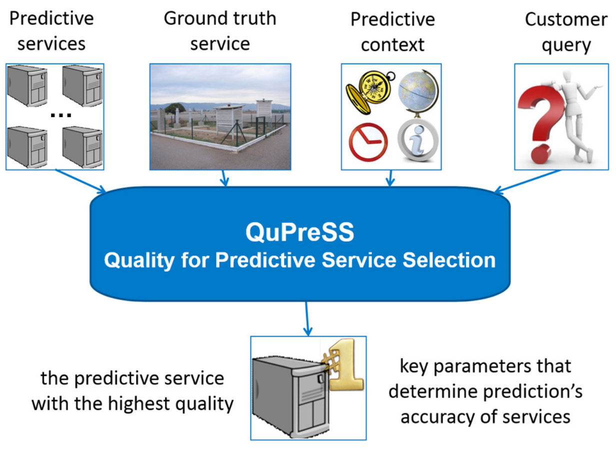

Predictive services
Ground truth service
Predictive context
Customer query
...
QuPreSS
Quality for Predictive Service Selection
the predictive service with the highest quality
key parameters that determine prediction's accuracy of services

Fig. 1. Inputs and outputs of the QuPreSS reference model.

- *Predictive services*. The portfolio of services that offer predictions to the customer. These services need to be identified in a service directory.
- *Ground truth*. Trusted information that is the object of prediction. It is collected from one single source (e.g., service providing real observations once they happen), which is trusted and reliable. Hence, only one ground truth service is needed.
- *Predictive context*. Context conditions that may influence the predictions (e.g., date and customer location).
- *Customer query*. The specific customer need that is required to be satisfied by a prediction given by some predictive service.

To get the first two inputs, it is necessary to use a monitoring infrastructure able to capture data from services. The last two inputs come from the customer, indirectly (information about the context gathered, e.g., from her GPS location) or directly (text of the query). These inputs are present in all the predictive domains that we referred to in Section 2.2.1.

As output of QuPreSS, the predictive service with the highest quality is recommended to the customer after the forecast verification process. These inputs and output have been formally defined in Martínez-Fernández et al. [3].

3.2. Steps to develop a tool based on QuPreSS

QuPreSS is based on the experience with developing Mercury and a careful study of the existing literature through a systematic literature review [3]. An exhaustive reference model covering all of the predictive domains found in the literature is beyond the scope of this article, and perhaps infeasible to achieve. Instead our intention has been to establish a reference model covering the core parts of such domains, while remaining extensible for future refinements. To that end, we defined the requirements of Table 1 for the specification of tools that cope with the prediction problem stated above. QuPreSS provides a standard decomposition into parts that cooperatively solve the problem of verifying and selecting predictive services. These parts are depicted in Fig. 2.

Table 1
Summary of requirements of the target tools of QuPreSS. Detailed requirements and fit criteria can be found in [22].

Number[a]	Requirement
FR 1	QuPreSS-based tools shall compare the predictions from predictive services with real observations in order to make a ranking of a set of predictive services based on forecast quality.
FR 2	QuPreSS-based tools shall read prediction data from several predictive services.
FR 3	QuPreSS-based tools shall read real observations coming from a trusted source (i.e., ground truth).
FR 4	QuPreSS-based tools shall monitor and parse data from both types of external sources: predictive services and ground truth service.
FR 5	QuPreSS-based tools shall save data from both types of external sources: predictive services and ground truth service.
FR 6	QuPreSS-based tools shall be able to offer data to external systems.
NFR 1	QuPreSS-based tools shall be extensible.
NFR 2	QuPreSS-based tools shall be developed as a service.
NFR 3	QuPreSS-based tools shall be adhered to standards.
NFR 4	QuPreSS-based tools shall be able to work with continuous data flows.
NFR 5	QuPreSS-based tools shall work when external sources are unavailable.

[a] *Note*: Functional Requirement (FR), Non-Functional Requirement (NFR).

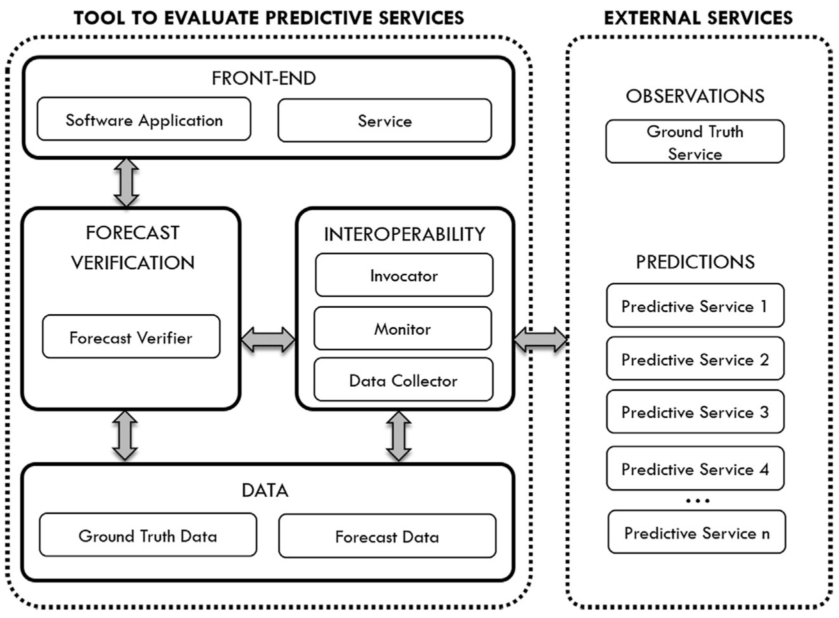


Fig. 2. The QuPreSS reference model.

To develop a tool based on the QuPreSS reference model, developers should conduct the next steps to map the parts defined in Fig. 2 into software elements and implement them:

1. *Selection of external sources*. QuPreSS-based tools need to have access to a ground truth service and a portfolio of predictive services. They are external and could be potentially heterogeneous. They can be implemented by web services

or other technologies. If an external source is not exposed as a web service, it can be wrapped into a web service proxy. These proxies also take care of parsing the different formats into a unified form. To integrate all these different technological styles, QuPreSS is organized around a service-oriented architecture [1]. A service-oriented architecture is essentially a collection of services that are able to communicate with each other, and therefore it can integrate heterogeneous systems [32].

2. *Selection and management of a monitoring tool.* In the heart of QuPreSS lies a monitor. It saves in a systematic manner the quality of service of each ground truth/predictive service and the response given to every periodical request launched to these services. An existing monitoring tool could be used when possible. In order to manage the monitor (i.e., to collect both the ground truth and the predictions gathered by the monitor service), it becomes necessary to read, parse and save the data collected. The management of the monitor is done by the forecasting data collector.
3. *Creation of two databases*. A wide variety of forecast verification procedures exist (see Section 2.2.2), but all involve measures of the relationship between a forecast or set of forecasts, and the corresponding observation(s). Thus, any forecast verification method necessarily involves comparisons between matched pairs of forecasts and the observations to which they pertain [33]. As a consequence, QuPreSS prescribes the use of two databases: one in charge of saving observations (Ground Truth Database) and another one to save predictions (Forecast Data Database). In those prediction domains in which predictions change very frequently, it would be better to use data stream management systems instead of a database management system to process continuous data flows [34]. Finally, for the design of the conceptual model of the databases, a previous study of the predictive domain becomes necessary.
4. *Creation of a forecast verifier service*. The Forecast Verifier component handles customer queries. It has two functions. First, it supports the decision-making process of choosing the most accurate predictive service from the available portfolio. To this end, a forecast verification procedure or metric needs to be implemented. Such procedure can be implemented from scratch or reusing existing software (e.g., R packages or the metrics library of the EVS, see Section 2.2.3). Moreover, the quality criteria (e.g., attribute of forecast quality and amount of days analyzed) needs to be defined in order to rank the available predictive services. Second, it handles the query over the chosen service (i.e., the “best” predictive service for the customer context). For this second functionality, it relies on the Invocator component, which invokes the chosen service to return its predictions to the customer.
5. *Creation of a front-end*. QuPreSS allows two different front-ends. First, a web client application that can be used by customers to directly enter their queries. Second, a web service that allows external services interoperating with QuPreSS-based tools. In both cases, predictive domain analysis needs to be performed to identify the queries that are meaningful for the customers.

In Section 4, we describe the instantiation of all parts of QuPreSS by following these five steps to build Mercury.

4. Developing Mercury: a QuPreSS-based tool to verify and select weather predictive services

Mercury is an instantiation of the QuPreSS reference model for the weather forecast domain. Therefore, it needs to fulfill the requirements defined in Table 1. Additionally, Table 2 shows the specific requirements of Mercury. In this section, we present how Mercury was built to accomplish all functional requirements by following the five-step process presented in Section 3.2. The resulting Mercury concrete software architecture after instantiating QuPreSS is depicted in Fig. 3.

Table 2
Summary of specific requirements of Mercury (extension of Table 1). Detailed requirements and fit criteria can be found in [22].

Number	Requirement
FR 1.1	Mercury considers the weather forecasting predictive domain.
FR 2.1	The weather predictive services currently considered by Mercury are: RSS Yahoo! Weather, Meteocat, and AEMET.
FR 3.1	The trusted source (i.e., ground truth) currently considered by Mercury is: AEMET.
FR 7	Mercury shall give current weather forecast from the most accurate predictive service for a specified city.
NFR 6	Mercury shall use existing components when possible.
NFR 6.1	Mercury shall be able to connect to SALMon to monitor web services.
NFR 7	Mercury shall obey legal statements from external web services.

4.1. External sources

Mercury deals with a simplified scenario in terms of service portfolio and context variables. The reason is that we encountered difficulties when looking for free predictive services [22]. On the one hand, the ground truth is obtained from the Spanish Meteorological Agency (AEMET[2]). This agency provides timely observations with the help of more than 700 stations with sensors measuring weather conditions. On the other hand, for demonstration purposes, the portfolio

[2] http://www.aemet.es/es/eltiempo/observacion/.

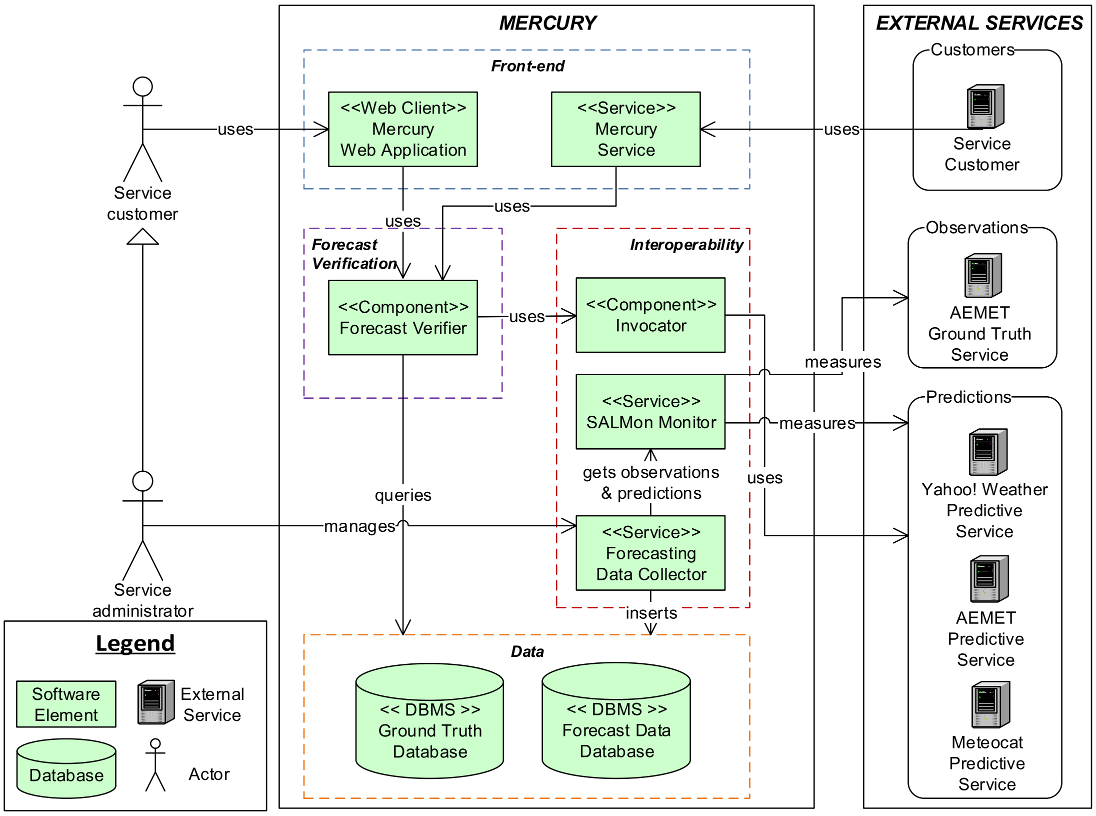

MERCURY
EXTERNAL SERVICES
Front-end
<<Web Client>> Mercury Web Application
<<Service>> Mercury Service
uses
Service customer
Customers
Service Customer
Forecast Verification
<<Component>> Forecast Verifier
Interoperability
<<Component>> Invocator
<<Service>> SALMon Monitor
gets observations & predictions
<<Service>> Forecasting Data Collector
Observations
AEMET Ground Truth Service
measures
Predictions
Yahoo! Weather Predictive Service
AEMET Predictive Service
Meteocat Predictive Service
queries
manages
inserts
Service administrator
Data
<< DBMS >> Ground Truth Database
<< DBMS >> Forecast Data Database
Legend
Software Element
External Service
Database
Actor

Fig. 3. Mercury concrete software architecture: instantiation of the QuPreSS reference model.

of predictive services is composed of: AEMET itself,[3] Meteocat,[4] and Yahoo! Weather[5]. These services are continuously monitored by Mercury in order to be able to infer in which contexts they are more adequate by comparing their predictions with the ground truth over time.

The three predictive services are wrapped into a web service with a pre-defined format that consists of an array of elements of type *ApiForecastData*. This type includes a superset of elements (which can be null) with information about a weather forecast for a date: *ConditionID*, *Description*, *Icon*, *Image*, *IsNight*, *Prediction*, *ShortPrediction*, *ShortTitle*, *TempHigh*, *TempLow*, *TempUnit*, *Title* and *WebUrl*. For more information about these elements, the reader is referred to [22].

Related requirements (see Table 1 and Table 2): FR 2, 2.1, 3, 3.1; NFR 1.

4.2. The monitor and the forecasting data collector services

We use the SALMon monitor infrastructure to implement this service [31]. SALMon was chosen given its adaptability and performance exhibited in previous uses [35–37]. The data collection process works as follows.

1. *Setup:* the parameters of the system are initialized (e.g., which are the cities put under the control of the monitor and which is the portfolio of predictive services considered).
2. *Ground truth collection:* the Forecasting Data Collector service daily saves a summary of the real observations (e.g., high and low temperatures of the day) once they happen for each city. These observations come from the trusted source (i.e., AEMET sensors).
3. *Prediction collection:* First, SALMon gets the response (with predictions) that every predictive service gives for each city. This operation is repeated several times per day to cope with temporal unavailability (every 6 hours by default). At the end of the day, the Forecasting Data Collector service gets the last response that SALMon obtained for each predictive service. The data provided by the monitor is treated for three reasons: identifying null values or errors of the predictive

[3] http://www.aemet.es/es/eltiempo/prediccion/municipios.
[4] http://dadesobertes.gencat.cat/ca/cercador/detall-cataleg/?id=16.
[5] http://developer.yahoo.com/weather.

services; checking the information provided in that forecast (several predictive services may give disparate information, e.g., the number of days in advance of the forecasts, and extra information such as meteorology alerts); and parsing the data to translate it into a uniform format. The goal of this treatment is to guarantee that the predictions stored in the databases are congruent.

Related requirements: FR 4; NFR 1, 2, 3, 5, 6, 6.1, 7.

4.3. Databases

Mercury includes two (relational) databases, instead of stream management systems, since for weather forecasting oscillations are seldom dramatic.

To determine the data model of Mercury, we consolidated the information given by the considered sources (i.e., all three predictive services and the ground truth service). These sources give different information about weather conditions (e.g., wind speed, humidity...). Still, a subset of information is common to all of them: maximum temperature, minimum temperature, unit of temperature, date of forecast/observation and location of forecast/observation.

Fig. 4 shows the conceptual model describing the data collected. The two main entities are *Forecast* and *Observation*. Since both of them share most of the attributes, we define these common attributes in an abstract entity *WeatherData*.

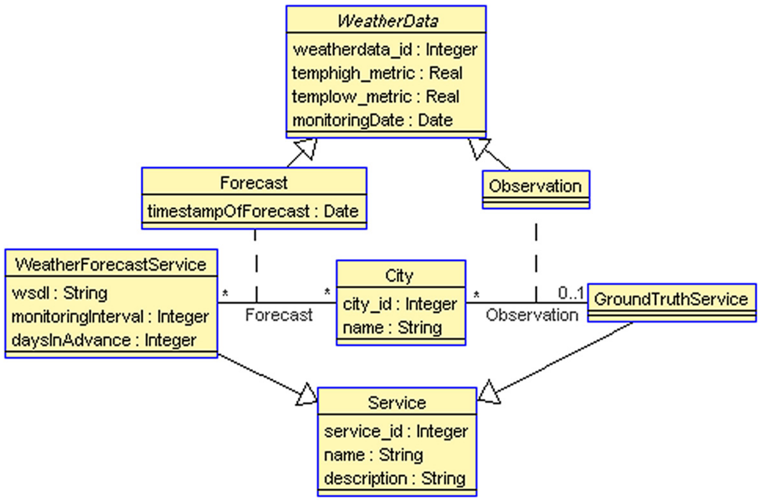


Fig. 4. Data conceptual model for weather forecast.

Observations are data points given by the *GroundTruthService*, whilst Forecasts are data points that a *WeatherForecastService* makes in advance. In both cases, they refer to a particular location (*City*).

The temporal dimension plays a fundamental role in the model. We have three temporal variables to reconcile: 1) the number of days predicted by a predictive service (attribute *daysInAdvance* in *WeatherForecastService*), 2) the date in which a prediction was made (*timeStampOfForecast* in *Forecast*), and 3) the date of the prediction (*monitoringDate* in *WeatherData*).

Related requirements: FR 5; NFR 4.

4.4. Forecast verifier

The quality parameter defined in this proof-of-concept was the accuracy of high and low temperature forecasts, because it is intuitive for non-experts in forecast verification. More precisely, we have implemented two measures, the mean-squared error and the approximation error. Both of them quantify the difference between values implied by an estimator (i.e., predictive services) and the truth values.

The mean-squared error (MSE) of a predictive service *PS* is defined as:

$$MSE_{PS} = \frac{\sqrt{\sum_{i=1}^{m} (\mu_{Tgt} - T_{PSi})^2}}{n} \tag{1}$$

where μ_{Tgt} is the average of temperatures (high or low) of the corresponding ground truth observations ("corresponding" means the observations that have the same value for the *monitoringDate* attribute as the *monitoringDate* attribute of the predictions of T_{PSi}); T_{PSi} is a temperature (high or low) as predicted by the *PS*; *i* is an index (which refers to a prediction made for a date interval); *n* is the total amount of observations from the *gt* (ground truth) and *m* the total amount of predictions made by the *PS* for a date interval. When the *PS* gives predictions more than one day in advance, *m* is greater than *n* because there are several predictions for the same date.

The approximation error (AE) is calculated as follows:

$$AE_{PS} = \frac{\sum_{i=1}^{n} |T_{GTi} - T_{PSi}|}{\sum_{i=1}^{n} |T_{GTi}|} \tag{2}$$

where *T* indicates a high or low temperature that is either a prediction from the *PS* or its corresponding observation from the ground truth (*GT*); *i* is an index (which refers to a date); and *n* is the total amount of observations/predictions compared.

A quality-based ranking of predictive services can be done by using any of these two measures. The default ranking is made with the mean-squared error of predictions for the last fifteen days, in which predictive services with lower mean-squared error are more accurate. This allows to select the predictive service with the highest likelihood to be right. This "best" service is the one with the highest quality in the past under the variables (e.g., a city) of a customer query.

Related requirements: FR 1, 1.1, 7; NFR 2.

4.5. *Front-ends*

Mercury provides two different front-ends. First, a web client application that can be used by customers to directly enter their queries. The parameters of the query are the date and the location in which the customer is interested, and the kind of response (report or redirection). Second, a web service that allows external services interoperating with Mercury via a "web service description language" (WSDL) interface. Section 5.1 provides more details on this part.

Related requirements: FR 6; NFR 2.

5. Installation and use of Mercury

Mercury is available on a virtual machine[6] inside the WaSDeTT 2013 bundle of SHARE [38], and on its own web page [39]. The virtual machine of Mercury contains a fully self-contained demo without Internet access. For testing purposes, it includes the Forecast Verifier web client and a historic data set collected from July 2011 to March 2014. Thus, it consists of an offline version of Mercury. For a running version of Mercury, which collects data from external sources every day, the reader is referred to Mercury's web page [39]. Both Mercury's virtual machine and web page also include installation documentation, databases' installation scripts, binaries of the web services and web client application, videos with example applications and user documentation.

Mercury has been implemented in Java. Mercury services run under the Tomcat 6 application server[7] and the web service engine Axis2 version 1.3[8]. MySQL has been chosen as database management system. After installing Tomcat 6, Axis 2 and MySQL, the installation of Mercury mainly consists of three tasks: creation of the databases with scripts, deploying the .war file of the web client front-end to Tomcat 6, and deploying the web services in Axis 2. The installation guide details these tasks.

To implement web services, Eclipse 3.3 J2EE version[9] has been used. In order to create and test web services, we have used the Web Service Tutorials for WTP 1.5[10]. The web client application uses jQuery[11] and Highcharts javascript libraries for graphics[12].

5.1. *Demonstration scenarios*

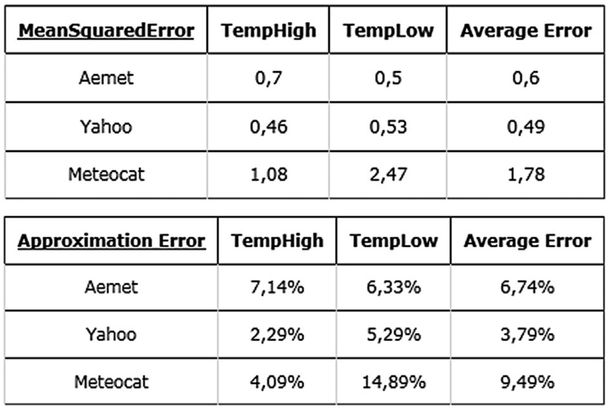
We have designed two demonstration scenarios to show the functions and usage of the web client graphical user interface and the web service. They are explained below and in the demonstration video accessible at the following URL: http://youtu.be/XE58jSwcIic.

6 http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=Ubuntu12LTS_MercuryQuPreSS.vdi.
7 https://tomcat.apache.org/download-60.cgi.
8 http://axis.apache.org/axis2/java/core/download.cgi.
9 http://www.eclipse.org/downloads/moreinfo/jee.php.
10 http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/.
11 http://jquery.com/.
12 http://www.highcharts.com/.

5.1.1. Analysis scenario

To analyze under which circumstances prediction models perform better or worse, Mercury can be requested to generate a report with the ranking of predictive services ordered by their mean-squared error or approximation error. This ranking is generated given a specified city and a period of time. There are two kinds of reports:

1. *Report with predictions made* `in` *a specific date:* It analyzes all weather forecasts made in the chosen date for a specific city. For instance, if the customer asks which service made better forecasts on August 4th 2011 in Tarragona, she gets the mean-squared errors of predictions made by all predictive services when forecasting next days' weather (5th, 6th...). As we can see in Fig. 5, the service that made better forecasts in that date was Yahoo!.

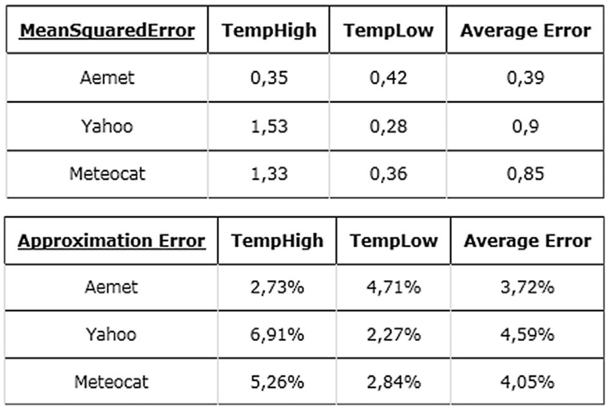


MeanSquaredError	TempHigh	TempLow	Average Error
Aemet	0,7	0,5	0,6
Yahoo	0,46	0,53	0,49
Meteocat	1,08	2,47	1,78

Approximation Error	TempHigh	TempLow	Average Error
Aemet	7,14%	6,33%	6,74%
Yahoo	2,29%	5,29%	3,79%
Meteocat	4,09%	14,89%	9,49%

Fig. 5. Errors in the "predictions made *in* a specified day" page.

2. *Report with predictions made* `for` *a specific date:* It analyzes all weather forecasts made for the given date and a specific city. For example, if the customer asks which service gave better forecasts for July 31st 2011 in Lleida, she gets mean-squared errors of predictions previously made (on 30th, 29th...) by all services for July 31st. The best service was AEMET (Fig. 6).



MeanSquaredError	TempHigh	TempLow	Average Error
Aemet	0,35	0,42	0,39
Yahoo	1,53	0,28	0,9
Meteocat	1,33	0,36	0,85

Approximation Error	TempHigh	TempLow	Average Error
Aemet	2,73%	4,71%	3,72%
Yahoo	6,91%	2,27%	4,59%
Meteocat	5,26%	2,84%	4,05%

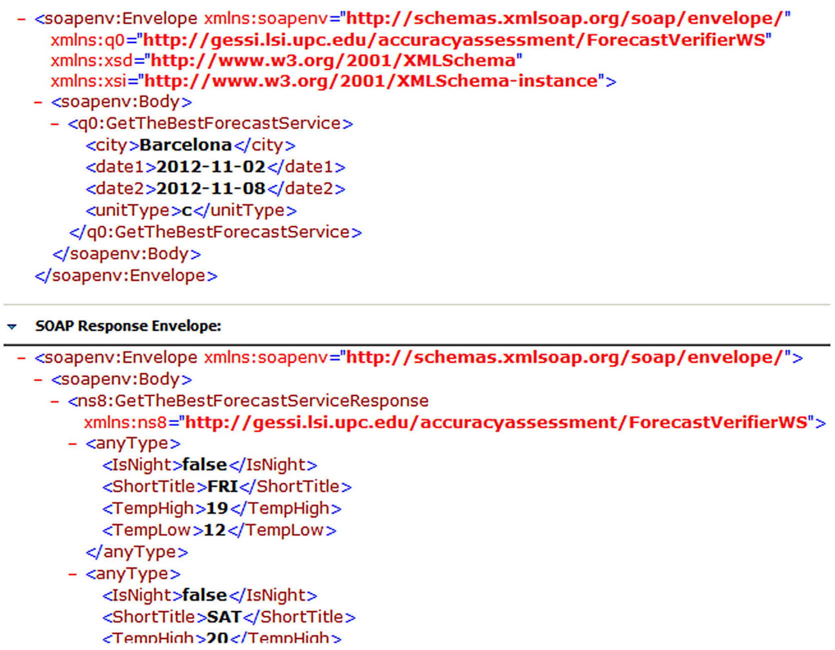
Fig. 6. Errors in the "predictions made *for* specified day" page.

The demonstration video shows the graphical representation of the results as displayed by Mercury.

5.1.2. Prediction scenario

In this scenario, the customer specifies a city and she gets the forecasts (with the highest likelihood to be right) for the next days. To do so, Mercury compares the errors that predictive services made in the predictions for that city in the last fifteen days, and returns the current forecasts from the predictive service that has been more accurate for that city in the past.

This functionality is not only available in the web client interface, but also as a web service. Fig. 7 shows an excerpt of the response given by the web service when asking for forecasts in Barcelona at November 2nd in 2012.



```
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:q0="http://gessi.lsi.upc.edu/accuracyassessment/ForecastVerifierWS"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <soapenv:Body>
    - <q0:GetTheBestForecastService>
        <city>Barcelona</city>
        <date1>2012-11-02</date1>
        <date2>2012-11-08</date2>
        <unitType>c</unitType>
      </q0:GetTheBestForecastService>
    </soapenv:Body>
  </soapenv:Envelope>
```

SOAP Response Envelope:

```
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  - <soapenv:Body>
    - <ns8:GetTheBestForecastServiceResponse
        xmlns:ns8="http://gessi.lsi.upc.edu/accuracyassessment/ForecastVerifierWS">
      - <anyType>
          <IsNight>false</IsNight>
          <ShortTitle>FRI</ShortTitle>
          <TempHigh>19</TempHigh>
          <TempLow>12</TempLow>
        </anyType>
      - <anyType>
          <IsNight>false</IsNight>
          <ShortTitle>SAT</ShortTitle>
          <TempHigh>20</TempHigh>
```

Fig. 7. Request (above) and response (below) of the ForecastVerifierWS.

6. Discussions and lessons learned

This section discusses the difficulties that we had to overcome while building Mercury, so that readers who embark on a similar project can learn from this experience. We could classify these problems in five categories: problems of integration; difficulties with testing a distributed system; problems of deployment; the need of experts in the domain; and, not preparing the academic tool to be commercially successful.

Problems of integration and evolution. The requirements basically dictated the integration and monitoring of various web services, and to progressively add new predictive services and other ways to measure their quality. As a result, we focused on developing a tool with a scalable service-oriented architecture whereas we simplified the predictive service portfolio and context variables. We found two major problems during the development of the tool, both related with the evolution of services or tools used. First, we needed to perform adaptations to the current version of SALMon, namely: adding the functionality of saving the whole response of the service which is monitored; and increasing the longitude of a varchar field to support longer soap actions. In next versions of SALMon, these functions should also be present to ensure backward compatibility. Second, in the beginning of the project a fourth weather predictive service was monitored, but after two months it became a pay-per-use service, so we stopped monitoring it. Also, the ground truth service has been evolved by the provider, what has implied new changes.

To improve the interoperability of a software system, we found that the service-oriented architecture pattern is a good solution. However, developers should be aware that the services that are not under their control may change and evolve, especially when there are no contracts. To mitigate this issue, it is recommended to use services with long term support.

Difficulties with testing a distributed system. Regarding testing, we started the process at the lowest level: unit testing of services that we developed and our web application. Later, we continued the testing process at the integration level to test the working of the forecasting data collector service. For details about how testing was performed, the reader is referred to [22]. The major problem that we faced was at the integration level. We realized that the most crucial non-functional requirement to integrate services that are not under your control reliability. External sources were distributed over the network and even developed and hosted by different organizations. Thus, an important requirement is that the tool should work when the response of external services is either expected or unexpected.

To avoid problems, it is important that the system is designed and tested considering that an external source may be unavailable (for reasons out of our control) or may give an unexpected response (for instance, because it evolved). In addition, it is recommendable to specify the actions to undertake when this type of problems occur (e.g., trying to reconnect for a specific number of times, or stopping monitoring).

Problems of deployment. We coped with several undesired issues while uploading the services to the production environment. First, the network of the university had the port 500 closed, which we were using to read observations from AEMET ftp server. To solve this problem, we opened this port and made a cron job to download the observations. Second, in the development environment, we were working with the latest version of Axis2 (1.5.4, released on December 2010) whereas in the production server the version of Axis2 was 1.3. The web service clients generated by version 1.5.4 were not compatible with version 1.3. Therefore, we needed to generate them again with the version 1.3 to solve this problem. Third, in order to make debug and testing easier, we had to ask for permission to access the log of Tomcat server.

In our case, Mercury was developed in a local environment by students (as many other academic tools). In this context, it is always necessary to plan some time to deploy the solution in a production environment, because the likelihood of having deployment problems is high (e.g., problems with the university network, technology versions, and permissions). Besides, version control is appropriate for teams with many developers.

The need of experts in the domain. We faced problems related to the need of having an expert in the predictive domain for the development of these tools, since it is needed to perfectly know the complex predictive context to assess the quality of predictions. For instance, non-expert stakeholders may have difficulties to properly cope with trivial problems (e.g., if an observation says that it rained just a few drops, can we consider as correct a forecast that predicted the rain?) or to design more complete validation plans.

To instantiate the QuPreSS reference model in tools for any predictive domain, it is advisable to look for experts in the domain. It is not desirable to make assumptions that something will work in a certain way if you have never done something similar. This is important, because many decisions related to the domain (e.g., metrics to measure forecast quality) have a lot of implicit assumptions that might not be true (e.g., utility of that metric for practitioners). Experts have necessary knowledge, such as what metrics are useful for the forecast verifier service.

Not preparing the academic tool to be commercially successful. As an academic tool, Mercury was created to prove the feasibility of the QuPreSS reference model and to offer tool support for the predictive service selection problem. However, as we could see in Table 1, none of the requirements was related to being commercially successful. Hence, technical and business problems arise when the tool is widely used. On the one hand, the tool is not well prepared to have many users (for instance, handling peak loads on the servers or malicious users). On the other hand, the tool is not prepared to be profitable, maintained and evolved in case of success.

We learned that academic tools should also consider how to be commercially successful. Then, it is recommended to create a business model for academic tools. The business model might help to determine the real gap in the market and to establish a unique business goal. This helps to make the tool simpler and congruent to such business goal. By having a business model, the academic tool is ready to be released to the market if it is a good idea.

Next, we show the strengths and weaknesses of this research.

6.1. Strengths and weaknesses

Mercury has allowed to assess the feasibility of the QuPreSS reference model and to understand the complexity of the prediction problem. We experienced the successful selection of predictive services by instantiating QuPreSS for the weather forecast domain. Moreover, QuPreSS may be applied for all the prediction domains that we have found, and the knowledge acquired during the development of Mercury could be transferred to the development of QuPreSS-based tools for other predictive domains.

Another strength of this project is that in our research group there were already people with experience in service-oriented architecture. As a result, Mercury development has benefited from the already defined service-oriented infrastructure implanted for SALMon, and knowledge about standards and de-facto technologies that had been gathered in previous experiences. It contributed to efficiently build Mercury and reduce the learning curve in the beginning. In that direction, we consider positive to develop academic tools based on previous efforts when possible.

The biggest strength of Mercury is its service orientation. It is scalable and allows the easy addition of new predictive services in the service directory, and new quality metrics to perform forecast verification.

On the other hand, among the weaknesses we should remark the following. First, weather forecasting is not as critical as other predictive domains. For instance, the value of rainfall forecast is not as high as other type of predictions, such as tropical cyclones and earthquakes in atmospheric, environmental and earth sciences. Also, as an academic tool, we have had limited resources (e.g., we implemented this first proof-of-concept for the weather forecast domain because it was the predictive domain for which more free services were available).

7. Conclusions and future work

We have presented Mercury, an instantiation for the weather forecast domain of the QuPreSS reference model. QuPreSS addresses the challenge of selecting the most accurate predictive service from a given portfolio to satisfy the customer's needs in a certain domain. The idea is that there are predictive services all across the web and tools become necessary to wire them up. Mercury enables to spawn a set of predictive services. Its main contribution is to make the interconnection of such predictive services possible in order to assess their predictions' quality.

There are four kinds of envisioned users for any instantiation of QuPreSS such as Mercury: individual citizens who want to obtain the best possible prediction for a particular query given their current context; service providers because it affects their reputation, and potentially their revenues; service designers who can integrate the tool in the core of self-adaptive service-oriented systems to guide its evolution; and, domain specialists (e.g., meteorologists, brokers, etc.) who want to understand when their prediction models behave better or worse. The current implementation of Mercury has been validated with three real weather predictive services. The main goal of the validation has been to assess the feasibility of the approach and to understand its complexity for designing more complete validation plans.

Future work spreads in three directions. First, analyzing the weather forecast domain to identify advanced characteristics to evaluate weather forecast quality. Specifically, performing an ontological analysis of the domain of Mercury to identify the relevant domain concepts and their relationships, and using data-mining to identify the current knowledge about key parameters that determine predictive service quality. Second, we encourage researchers in other predictive domains to create Mercury-like instantiations of QuPreSS to evaluate the forecast quality of their prediction models. Third, we recognize that the QuPreSS reference model could be refined to address more functionalities, quality attributes, and facilities to software developers. With regard to new functionalities, several forecasts from predictive services could be combined in order to create a superior composite forecast, and service customers could upload their own data and use the forecast verifier service. Concerning quality attributes, many ground truth services could be integrated to evaluate their quality of service (e.g., availability) and reliability. Regarding the facilities of QuPreSS to software developers in any predictive domain, the reference model could be mapped to software components to ease the connectivity of services, provide guidelines to reuse such software components, and offer an implemented software component with a collection of forecast quality measures.

Acknowledgements

This work has been supported by the Spanish project TIN2013-44641-P, and the Spanish grant FPU12/00690.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.scico.2015.11.009.

References

[1] M. Papazoglou, Web Services: Principles and Technology, Addison–Wesley, 2008.
[2] K. Nikolopoulos, K. Metaxiotis, V. Assimakopoulos, E. Tavanidou, A first approach to e-forecasting: a survey of forecasting web services, Inf. Manag. Comput. Secur. 11 (2003) 146–152.
[3] S. Martínez-Fernández, J. Bisbal, X. Franch, QuPreSS: a service-oriented framework for predictive services quality assessment, in: 7th International Conference on Knowledge Management in Organizations: Service and Cloud Computing, KMO 2012, Springer, Berlin, 2013, pp. 525–536.
[4] S. Martínez-Fernández, X. Franch, J. Bisbal, Verifying predictive services' quality with Mercury, in: 4th International Workshop on Academic Software Development Tools and Techniques (WASDeTT-4 2013).
[5] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 3rd edition, Addison–Wesley Professional, 2012.
[6] OASIS, Reference model for service oriented architecture 1.0, https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, 2006 [Accessed February 21st, 2014].
[7] D. Weyns, S. Malek, J. Andersson, FORMS: unifying reference model for formal specification of distributed self-adaptive systems, ACM Trans. Auton. Adapt. Syst. 7 (2012) 1–61.
[8] P. Clements, R. Kazman, M. Klein, Evaluating Software Architectures, Addison–Wesley Reading, 2001.
[9] E.Y. Nakagawa, M. Guessi, J.C. Maldonado, D. Feitosa, F. Oquendo, Consolidating a process for the design, representation, and evaluation of reference architectures, in: IEEE/IFIP Conference on Software Architecture, WICSA, 2014, pp. 143–152.
[10] A.H. Murphy, What is a good forecast? An essay on the nature of goodness in weather forecasting, Weather Forecast. 8 (1993) 281–293.
[11] I.T. Jolliffe, D.B. Stephenson, Forecast Verification: A Practitioner's Guide in Atmospheric Science, John Wiley & Sons, 2012.
[12] J.S. Armstrong, Principles of Forecasting: A Handbook for Researchers and Practitioners, vol. 30, Springer, 2001.
[13] B. Casati, L. Wilson, D. Stephenson, P. Nurmi, A. Ghelli, M. Pocernich, U. Damrath, E. Ebert, B. Brown, S. Mason, Forecast verification: current status and future directions, Meteorol. Appl. 15 (2008) 3–18.
[14] E.E. Ebert, L.J. Wilson, B.G. Brown, P. Nurmi, H.E. Brooks, J. Bally, M. Jaeneke, Verification of nowcasts from the WWRP Sydney 2000 forecast demonstration project, Weather Forecast. 19 (2004) 73–96.
[15] X. Yu, Overview of Beijing 2008 Olympics WWRP Nowcast FDP and implementation plan, in: WWRP Symposium on Nowcasting and Very Short Range Forecasting, Toulouse, 5–9 September 2005.
[16] A. Murav'ev, A.Y. Bundel, D. Kiktev, D. Blinov, A. Smirnov, Verification of mesoscale forecasts in the 2014 Olympic Games region for the first test period. Part I: verification techniques and polygonal quality assessments of the COSMO model forecasts, Russ. Meteorol. Hydrol. 38 (2013) 723–734.
[17] F.X. Diebold, J.A. Lopez, Forecast evaluation and combination, http://www.nber.org/papers/t0192, 1996 [Accessed February 24th, 2014].
[18] J.E. Campbell, Introduction – the 2004 presidential election forecasts, PS Polit. Sci. Polit. 37 (2004) 733–735.
[19] M.S. Lewis-Beck, Election forecasting: principles and practice, Br. J. Polit. Int. Relat. 7 (2005) 145–164.
[20] D.G. Altman, P. Royston, What do we mean by validating a prognostic model?, Stat. Med. 19 (2000) 453–473.
[21] S.C. Johnston, P.M. Rothwell, M.N. Nguyen-Huynh, M.F. Giles, J.S. Elkins, A.L. Bernstein, S. Sidney, Validation and refinement of scores to predict very early stroke risk after transient ischaemic attack, Lancet 369 (2007) 283–292.
[22] S. Martínez-Fernández, Accuracy assessment of forecasting services, http://upcommons.upc.edu/pfc/handle/2099.1/13105, 2011 [Accessed October 15th, 2015].
[23] H.E. Brooks, C.A. Doswell III, A comparison of measures-oriented and distributions-oriented approaches to forecast verification, Weather Forecast. 11 (1996) 288–303.
[24] A.H. Murphy, R.L. Winkler, A general framework for forecast verification, Mon. Weather Rev. 115 (1987) 1330–1338.

[25] M. Pocernich, Appendix: verification software, in: Forecast Verification: A Practitioner's Guide in Atmospheric Science, second edition, 2012, pp. 231–240.
[26] C. Fraley, A.E. Raftery, T. Gneiting, J. Sloughter, V.J. Berrocal, Probabilistic weather forecasting in R, R J. 3 (2011) 55–63.
[27] J.D. Brown, J. Demargne, D.-J. Seo, Y. Liu, The ensemble verification system (EVS): a software tool for verifying ensemble forecasts of hydrometeorological and hydrologic variables at discrete locations, Environ. Model. Softw. 25 (2010) 854–872.
[28] NOAA, The ensemble verification system (EVS) – national weather service – office of hydrologic development, http://amazon.nws.noaa.gov/ohd/evs/evs.html, 2013 [Accessed February 21st, 2014].
[29] B. Domenico, Forecast verification with observations: use case for geosciences web services, http://www.unidata.ucar.edu/projects/THREDDS/GALEON/Phase2Connections/VerificationUseCase.html, 2007 [Accessed April 1st, 2013].
[30] G. Canfora, M. Di Penta, Service-oriented architectures testing: a survey, in: Software Engineering, Springer, 2009, pp. 78–105.
[31] M. Oriol, X. Franch, J. Marco, Monitoring the service-based system lifecycle with SALMon, Expert Syst. Appl. 42 (2015) 6507–6521.
[32] U. Zdun, C. Hentrich, W.M. Van Der Aalst, A survey of patterns for service-oriented architectures, Int. J. Internet Protoc. Technol. 1 (2006) 132–143.
[33] D.S. Wilks, Statistical Methods in the Atmospheric Sciences, vol. 100, 2011, Access Online via Elsevier.
[34] L. Golab, M.T. Özsu, Issues in data stream management, ACM SIGMOD Rec. 32 (2003) 5–14.
[35] C. Muller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortes, M. Rodriguez, Comprehensive explanation of SLA violations at runtime, IEEE Trans. Serv. Comput. 7 (2014) 168–183.
[36] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, K. Pohl, Usage-based online testing for proactive adaptation of service-based applications, in: 2011 IEEE 35th Annual Computer Software and Applications Conference, COMPSAC, 2011, pp. 582–587.
[37] A. Kertesz, G. Kecskemeti, M. Oriol, P. Kotcauer, S. Acs, M. Rodríguez, O. Mercè, A.C. Marosi, J. Marco, X. Franch, Enhancing federated cloud management with an integrated service monitoring approach, J. Grid Comput. (2013) 1–22.
[38] P. Van Gorp, S. Mazanek, SHARE: a web portal for creating and sharing executable research papers, Proc. Comput. Sci. 4 (2011) 589–597.
[39] S. Martínez-Fernández, Mercury: a tool for selecting the most accurate weather predictive service based on QuPreSS reference model, http://www.essi.upc.edu/~smartinez/?page_id=131, 2014 [Accessed February 24th, 2014].