Enterprise Service Bus Solution for an Efficient Development of Geodesic Monitoring Systems

A. Itu
Department of Automation and Information Technology, Transilvania University of Brasov, Brasov, Romania
(alina.itu@unitbv.ro)

Abstract – **Modern constructions are typically monitored with geodesic monitoring systems. One the most important challenges for these systems is represented by the complexity of developing new instance from the ones already set up. The solution introduced herein is based on a service oriented architecture, which in turn relies on an enterprise service bus as its principal component. The monitoring system consists of three principal components: measurement system, data cleaning and integration, and the deformation analysis tool. By employing the herein proposed approach these three components become loosely coupled. Herein, the focus lies on the component performing data cleaning and integration, consisting of three principal chains of services linked either serially or sequentially. The first chain manages the files containing the observations recorded by the tachymeters which perform the object monitoring. The second chain manages the data sent by the humidity and temperature sensors. The third chain performs preliminary analysis functions, and also makes the results public on a website. Error management at bus level is performed by a fourth chain in the ESB: it manages the issues of language level error handling and improves the transparency. Due to the loose coupling characteristics of the system, the components can be developed in parallel. Together with the improved flexibility and the greater reusability, this is a key aspect of a service oriented architecture.**

Keywords - **Geodesic monitoring system, data integration and cleaning, error management, ESB, SOA**

I. INTRODUCTION

Geodesic monitoring systems are employed to monitor over time the structural changes of an object. The monitoring consists in observing, determining and analyzing these changes. The principal application area of such systems is the monitoring of water storing dams [1], [2]. However, they may also be used for analyzing bridges or buildings.

Typically, geodesic monitoring systems rely on a fixed reference frame. This means that the observations are taken from a static point of view, and all measurements are performed in relation to the static point.

To globally assess the spatial displacements of an object, a set of points is chosen, and their movement is analyzed in time. This means, the locations of the points are determined periodically. As a result, the tachymeter is the principal building block of a geodesic monitoring system, which performs measurements using laser technology [4]. The tachymeter measures both distances and angles pertaining to a certain point on the considered object. The principal difficulty in performing the periodic measurements is to identify and aim at the exact same location at different time points.

Hence, if the location of a point changes, and a new measurement is performed, two different angles (noted as α and β in fig. 1) are employed, to be able to correctly position the system [5]. One simple solution is to place a reflecting surface, e.g. a Lüneburg lens, at each location which is monitored. This means, that the tachymeter has to also process the reflected beam, i.e. to determine its intensity. Furthermore, a position control system needs to be put in place, to control the two degrees of freedom [6], to maximize the perceived intensity of the reflected beam.

Once the locations of the points can be reliably determined in time, a deformation analysis needs to be performed, to evaluate whether significant structural changes have taken place, i.e. evaluate whether significant displacements have been observed. This analysis is performed using a software tool.

A drawback of current geodesic monitoring systems is their tightly coupled nature. It is not possible to simply reuse existing components when setting up a new monitoring system (fig. 2 displays the principal components of the device [7]).

The focus of the work reported herein has been mainly to decouple the components depicted in figure 2, using a solution based on a service oriented architecture [8] and an enterprise service bus (ESB).

The ESB is designed to handle interoperability issues [9]. Specifically, it provides an infrastructure which allows for a simplification of the integration and which also enables the reutilization of the various components in the above mentioned service oriented architecture. This is performed by enabling any type of data transport and / or transformation. An ESB is event-driven, service- and message-oriented, and these properties are combined to

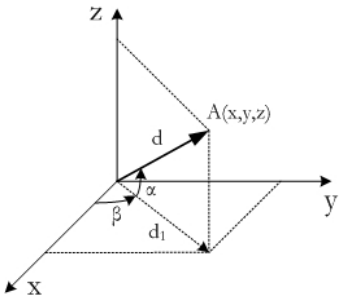

Figure 1. Point in space defined in polar coordinates.

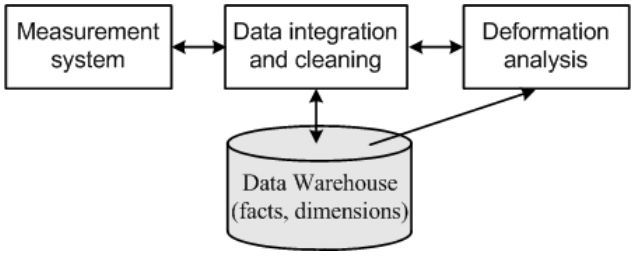

Figure 2. Principal components in a geodesic monitoring system.

generate a secure and scalable infrastructure, capable of integrating scattered IT resources and applications. The messages submitted on the bus have a header (containing key-value pairs representing various parameters) and a body, which for Java implementations as used herein, may rely on any type of class (object) [10].

To allow for interoperable software components, which would otherwise be incompatible, ESBs can execute data format transformation, communication protocol transformation, and interaction model transformation [11]. As a result, loosely coupled systems are obtained, capable of communicating, and which can be adapted and improved with little effort.

The objective of this manuscript is to showcase the implementation of an ESB, designed for a geodesic monitoring system, which provides all the principal advantages of a service oriented architecture: reusability, smaller reaction time and flexibility. The Spring Integration portfolio has been employed for the implementation of the ESB introduced in the paper, but all of the concepts described herein apply for any service bus [12]. The ESB enables data transformation and validation, a decoupling of the principal system components, it stores the data in the database, and allows for calling preliminary analysis workflows.

In the following section the monitoring system is described in detail, with emphasis on the data format. Next, the ESB solution is described with its components and chains, the bus level error management, and with a basic version of the database design. Section three describes the results, and the last section presents the conclusions of the work.

II. MATERIALS AND METHODS

A. Geodesic monitoring systems

Herein we employed a geodesic monitoring system composed from three tachymeters, each of them being responsible for the monitoring of fifteen points. As described above, a fixed reference frame is considered, and all positions are related to it. To be able to correlate the measurements with the weather conditions, additionally, humidity and temperature sensors have been mounted.

The focus of the paper does not lie on the measuring system, hence in the following we address the system outputs, and the processing operations which are then performed by the middleware.

The outputs of the geodesic monitoring system are three file types:

- An xml file, named observation file, which specifies the tachymeter position, and the positions of the points which are monitored by the tachymeter
- A text file specifying the current humidity value
- A text file specifying the current temperature value

The structure of the xml is defined by an xsd schema file. The file specifies the precise date and time of the observation, the precise date and time of the time point at which the point location was determined (these time points are separately specified since the point locations are measured sequentially). The exact time point is not crucial, but the differences in time between the different measurements are processed by a filter implemented in the ESB framework for validating the measurement.

A single measurement is stored in each text file. The humidity and the temperature are measured more often than the point locations. Hence, several text humidity / temperature files will be linked with a single observation file. The correlation of the measurements will be performed by the ESB solution.

B. Enterprise service bus solution

The principal part of the ESB designed for the herein described solution is depicted in fig. 3 [13], [14]. It is composed from three principal chains, where two of them are merged while the processing is being performed. For each ESB component a Java class has been defined and implemented (class name is depicted right on top of each ESB component). The three chains are briefly described in the following, with emphasis on the principal bus parts. An XML file (esb-config.xml) contains the definition of the ESB configuration.

The first chain polls the file system at a pre-specified time intervals, to access newly available xml observation files. A file-adapter is used at the start of the first chain, and the files are sent as message bodies through the output channel. In the following, a transformer generates a Java class instance from the XML file, by employing the DOM API. Next, a filter is applied, which dismisses the observation if the time interval between measurements

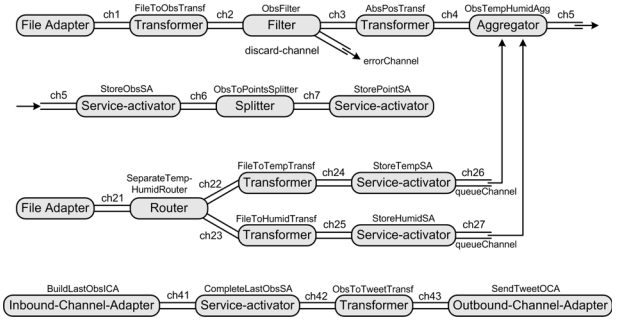

Figure 3. Structure of the ESB.

performed for two points is too long. The threshold has been set at 400 seconds, which is equivalent to a mean time interval of approx. 28.5 seconds between consecutive measurements. The threshold value was defined by a specialist in geodesic monitoring systems: it is of high importance for an accurate deformation analysis to have short measurement intervals, because, otherwise the accuracy of the analysis decreases. In the following, a transformer is employed, which performs the translation of the point locations from the local tachymeter coordinate system to the global coordinate system (defined by the reference frame). Before addressing the following aggregator, the second ESB is described.

A file adapter is also used at the start of the second chain. It polls the file system for obtaining access to newly available humidity or temperature text files, which are then sent as message bodies to a router. The router differentiates the humidity and temperature files, which are then sent through different channels. Hence, the processing is then performed in parallel. While performing the processing, the files are transformed into Java class instances, the information is stored in the database, and the instances are placed on queue channels, where they are stored until a different component or thread reads them (up to 100 messages can be stored at a time). The aggregator employed in the first chain processes all messages placed in these queue channels, and then determines which are the most suitable humidity and temperature samples for each tachymeter measurement (basically, the values which are closest in time to the measurement time point). A part of the aggregator code is displayed in figure 4. In the following, the observation is saved (containing the tachymeter location, and the humidity and temperature values, together with the time and date information). Afterwards,

```
public class ObsTempHumidAggreg {

    QueueChannel qcTemp;
    QueueChannel qcHumid;

    @ServiceActivator
    public Observation aggregateObsTempHumid(Message<Observation> observation){

        Observation obs=observation.getPayload();

        //Calculating the observation in seconds
        int secunde;
        Date date_obs=obs.getDate();
        secunde=date_obs.getHours()*3600+date_obs.getMinutes()*60+date_obs.getSeconds();

        Humidity minH=null,h;
        Temperature minT=null,t;

        //Reading the messages from the queue channels
        List temperatures = qcTemp.clear();
        List humidities = qcHumid.clear();

        //Iterating through the temperatures to determine the appropriate one
        Iterator i= temperatures.iterator();
        while(i.hasNext())
        {
            //Verifying the time difference between the observation date and
            //the temperature date
            //..................
        }


        //Iterating through the humidities to determine the appropriate one
        Iterator l= humidities.iterator();
        while(l.hasNext())
        {
            //Verifying the time difference between the observation date and
            //the humidity date
            //..................
        }

        obs.setTemp(minT);
        obs.setHumid(minH);

        return obs;
    }
```

Figure 4. Aggregator service.

a splitter divides the payload, i.e. the points list, into distinct messages, where each one contains a single point.

The final ESB chain submits messages at periodic time intervals to twitter, where a specific account was created. Messages are being sent every six hours, describing the monitoring system status. Each message is composed from:

- Latest locations of the three tachymeters
- Greatest displacements of the locations being monitored
- Humidity and temperature corresponding to the last observation

Since the messages sent to twitter are limited to a length of 140 characters, the transformer divides the message into two separate messages, which are then sent at an interval of 60 seconds by the outbound channel adapter. The communication with the twitter account is handled through the Twitter4J API, available under the BSD license [15].

Another important observation is that in an ESB solution, errors cannot be managed by employing only a basic try-catch mechanism. The general rule states that all errors should be collected on an error channel, and then they should be submitted to distinct processing blocks, which can take the corresponding actions [12], [16]. Herein, a router has been used to differentiate between the four remaining error cases, which cannot be handled at language level. A separate Java class has been implemented, specialized for handling a specific error type. For most of the errors depicted in fig. 5, detailed messages are being sent to one or several maintenance / service engineers. Specifically:

- a problem in the measuring system may appear if there is no correspondence between the schema file and the xml file
- if the measurement is too long, the tachymeter may have encountered an issue
- either the humidity or the temperature sensor may not function properly
- the twitter communication may be interrupted

The component performing the data cleaning and integration saves the results in a database, whose structure is depicted in figure 6. Globally, the tables may be categorized into dimension and fact tables [17]. To perform a comprehensive deformation analysis, all the information collected in the past for an object has to be

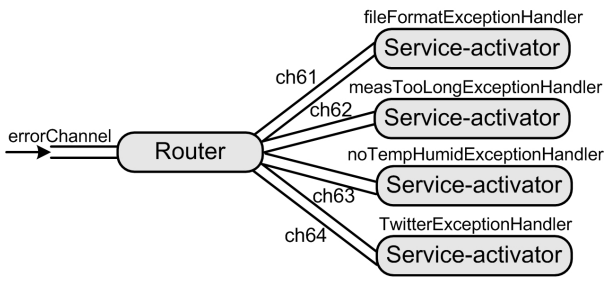

Figure 5. Chain performing error management.

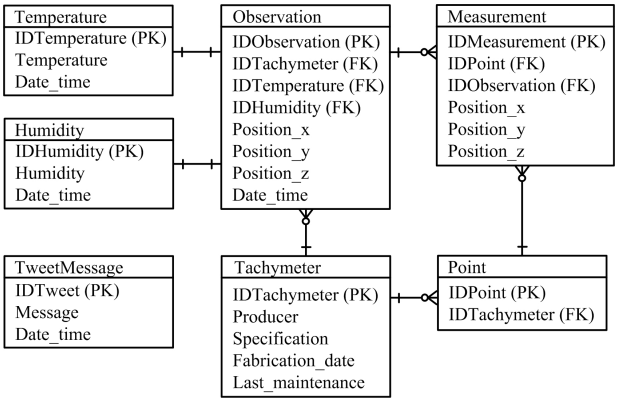

Figure 6. Database structure.

accessed from the database. The third normal has been used for normalizing the structure of the tables, guaranteeing that no redundant data is stored and that maintenance can be very easily performed. There are numerous connections between the tables, which can be categorized as being either one-to-many or one-to-one. Finally, the Hibernate framework has been employed as a persistence management solution and for object relational mapping.

III. RESULTS

The system was tested continuously for approximately 72 hours. At intervals of six hours, a message pair has been sent:

- message 1: tachymeter locations and the corresponding humidity and temperature measurements
- message 2: locations with the greatest displacements during the last six hours

The location of the first tachymeter for all three axes is displayed in fig. 7. The actual data displayed in the figure was measured while running one of the numerous test sessions in our laboratory.

We note that no exceptions or errors were recorded while running the test phase. The processing times of the three chains depicted in fig. 3 have been stored during the test runs (the queue channels pair was considered as the terminal point of the second chain), and are displayed in fig. 8.

No considerable delays were observed during the processing steps: all differences from case to case were very small. We note that the runtimes are the largest for the first chain. This is caused by the large number of services in this chain.

The final chain actually leads to comparable runtimes, although it is based on only four services (whereas the first chain contains seven services). This aspect is caused by the greater complexity of the services in the third chain. Finally, we note that the second chain has the shortest runtimes.

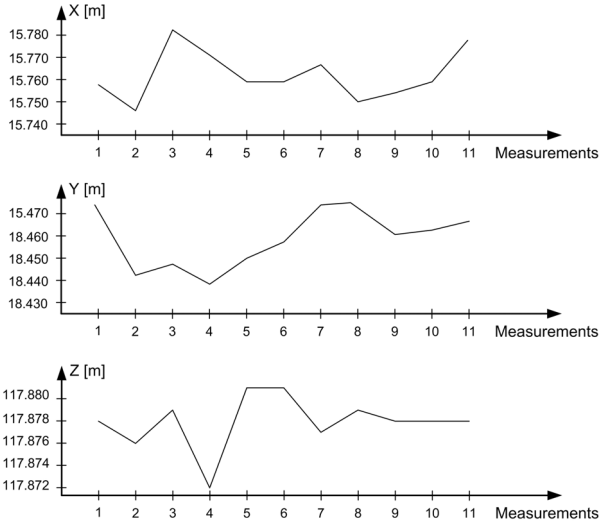

Figure 7. Location of the first tachymeter during the tests.

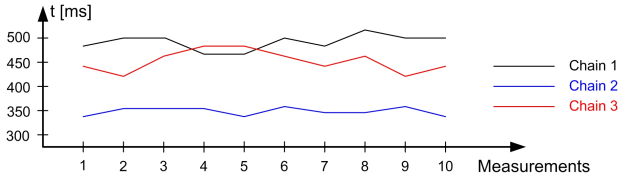

Figure 8. Runtimes of the three chains during the tests.

IV. CONCLUSIONS

A novel approach for setting up geodesic monitoring systems has been introduced in the paper. The service oriented architecture based solution enables a decoupled development of the principal components in the system (fig. 2). The main advantage is that both the development and the testing times can be reduced, given the loosely coupled system configuration.

Secondly, the software components can be reused with minimal effort. When a new geodesic monitoring system needs to be set up, the various building blocks can be integrated with minimal or even no changes. Thus, both new systems can be configured easily, and existing systems can be quickly modified if desired.

A first conclusion would be that the proposed system provides all the principal advantages of a service oriented architecture. These were mentioned in the first section: smaller reaction time, flexibility, reusability. We also note that the application area of geodesic monitoring systems is well suited for such a methodology, given the clearly separable but still interconnected building blocks. A narrow part of the processing is performed by each software service depicted in fig. 5 [18]. Hence, once the requirements were clear and set, it was possible to develop all components in parallel.

The solution based on the enterprise service bus is the simplest one for setting up an end-to-end application (see the data cleaning and integration component in fig. 2). The Spring Integration ESB was considered because it

can be configured very fast and efficiently through an xml file. The services have been implemented as POJO classes, based on specific annotations. Reusability is obtained at different levels:

- chain level: each series of services can be reused without requiring details regarding the services in the chain
- service level: each service can be reused independently from the context

We conclude that another significant advantage of the approach is the varying reusability granularity.

Another very useful aspect is the error management capability of the ESB based approach. Since errors cannot be managed always at language level, the fact that they can be managed also at bus level, relying on specialized services, represents an important advantage.

Finally, not only the use of the Spring Integration portfolio has provided important advantages, but also the usage of the Spring framework [19]. For example, for the aggregator depicted in fig. 4, the dependency-injection capability enables the access to the second chain queue channels.

ACKNOWLEDGMENT

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CCCDI – UEFISCDI, project number ERANET-FLAG -RoboCom++ (2), within PNCDI III.

REFERENCES

[1] A. Sortisa, P. Paoliani, Statistical analysis and structural identification in concrete dam monitoring, Engineering Structures, vol. 29, pp.110-120, 2007.

[2] J. Jeon, J. Lee, D. Shin, H. Park, Development of dam safety management system, Advances in Engineering Software, vol. 60, pp. 54-563, 2009.

[3] W. Stempfhuber, Online Monitoring historischer Kirchen mit einem Präzisionstachymeter mittels reflektorloser, direkter oder indirekter Winkel- und Streckenmessung, XIV. International Course on Engineering Surveying, Zürich, 2004.

[4] W. Stempfhuber, Genaue Positionierung von bewegten Objekten mit zielverfolgenden Tachymetern, XIII. International Course on Engineering Surveying, München, 2000.

[5] F.Y. Leu, G.C. Li, A scalable sensor network using a polar coordinate system, Signal Processing, vol. 87, pp.2978-2990 , 2007.

[6] S. Dejima, W. Gao, H. Shimizu, S. Kiyono, Y. Tomita, Precision Positioning of a five degree-of-freedom planar motion stage, Mechatronics, vol. 15, pp. 969-987, 2005.

[7] B. Reuter, D. Henrici, A model for service-oriented communication systems, Journal of Systems Architecture, vol. 54, pp. 594-606, 2008.

[8] T. Erl, Service-Oriented Arhitecture: Concepts, Technology, and Design, Indiana, Prentice Hall PTR, 2005.

[9] D. A. Chappell, Enterprise Service Bus – Theory in Practice, New York, O'Reilly Publishing, 2004.

[10] E. Hewitt, Java SOA Cookbook, Sebastopol, O'Reilly Media, 2009.

[11] G. Decker, O. Kopp, F. Leymann, M. Weske, Interacting services: From specification to execution, Data & Knowledge Engineering, vol. 68, pp. 946–972, 2009.

[12] J. Long, G. Mak, Spring Enterprise Recipes: A problem-solution approach, New York, Springer-Verlag, 2009.

[13] M. Rosen, Applied SOA: Service-Oriented Architecture and Design Strategies, Indianapolis, Wiley Publishing, 2008.

[14] J. Lawler, Service-oriented arhitecture: SOA strategy, methodology, and technology. New York, Auerbach Publications, 2008.

[15] http://yusuke.homeip.net

[16] C. Riedl, T. Bohmann, M. Rosemann, H. Krcmar, Quality management in service ecosystems, Inf. Syst. E-Bus Manage, vol. 7, pp. 199–221, 2009.

[17] F. Silvers, Building and Maintaining a Data Warehouse, New York, CRC Press, 2008.

[18] M. Turner, D. Budgen, P. Brereton, Turning software into a service, IEEE Computer, vol. 36, pp. 38-45, 2003.

[19] D. Minter, Beginning Spring 2: From Novice to Professional, New York, Springer-Verlag, 2008.