Design of a service oriented architecture for a geodesic monitoring system

Alina Itu[1,2]
[1]Siemens Corporate Technology, Siemens SRL
[2]Transilvania University of Braşov
Braşov, Romania

***Abstract*— Geodesic monitoring systems are used for the surveillance of modern constructions. They have evolved significantly over the past years due to several breakthroughs in engineering. One of the significant issues for such systems is the difficulty of developing new systems based on the existing ones. The solution presented in this paper relies on a service oriented architecture having an enterprise service bus as its core component. By using this approach, the three main components of the monitoring system (measurement system, data integration and cleaning, and deformation analysis) become loosely coupled, and, thus, more independent. The paper focuses on the data integration and cleaning component for which three main chains of sequential or parallel services have been developed. The first chain processes the observation files provided by the three tachymeters which monitor the object, the second one processes the data provided by the temperature and humidity sensors, and the third chain provides preliminary analyses functions, and publishes the results on a website. A fourth chain of the ESB focuses on the error management at bus level, which handles the issues of language level constructions and provides greater transparency. Thus, the ESB decouples the main components of the system, it provides data transformation and validation, and it inserts the data into the data warehouse. The loosely coupled nature of the system facilitates parallel development of the components, enhanced reusability, and greater flexibility, all of which are key aspects for a service oriented architecture. Another advantage of our approach is the varying granularity of reusability, which can be adopted at service level, but also at chain level.**

Keywords*— *geodesic monitoring system, SOA, ESB, data integration and cleaning, error management

I. INTRODUCTION

A geodesic monitoring system is a system employed to observe, determine and analyze or assess the geometrical changes which an object goes through in time. One of the main areas where these systems are used is the monitoring of dams, which are used to store water [1], [2], but also for buildings, bridges, etc.

As described in [3], in classical geodesic monitoring systems the measurements are performed from a static point of view, which means there is a fixed reference frame to which all of the measurements are related to.

To determine the displacement in space of an entire object, the displacements of a set of points are determined. As a result, the positions of the points are measured periodically. Thus, the main component of a geodesic monitoring system is a tachymeter (a distance measuring laser device [4]) by which the distance and the angles corresponding to a certain point on the monitored object can be measured continuously. The use of a distance measuring laser device, though, has a major difficulty: the aiming of the exactly same point at consecutive measurements.

As a result, if the point moves, and a new measurement is undertaken, the positioning must be performed using two different angles α and β (fig. 1) [5]. A possible approach for handling this issue is to mount a reflecting object based on a Lüneburg lens. To employ this technique, the tachymeter should be able to measure the intensity of the reflected beam. This intensity represents the input of the position regulating system which possesses two degrees of freedom [6] and which orientates the laser device.

The other main component of a geodesic monitoring system is a software package which performs a so-called deformation analysis. This software tool analyzes the positions of the points and draws conclusions regarding the displacement of the entire object.

One issue of existing geodesic monitoring systems is that their components are tightly coupled, and, thus, a significant part of the effort put in building these systems is lost when trying to set up a new monitoring system. Fig. 2 displays the main components of the system [7].

Our research has focused on decoupling the components displayed in fig. 2 by providing a service oriented architecture solution [8] with an enterprise service bus (ESB) as its core component.

ESB is a service bus which solves interoperability problems [9]. ESBs provide a software infrastructure which simplifies the integration and allows for a flexible reutilization of components in a service-oriented architecture, by facilitating data transport and transformation of any kind. ESBs are message-oriented, service-oriented and event-driven, and these characteristics are combined to provide a scalable and secure infrastructure, which is able to integrate scattered applications and IT resources. Each message sent on the bus has a header which contains a series of parameters as key-value pairs, and a payload which in this case (a Java

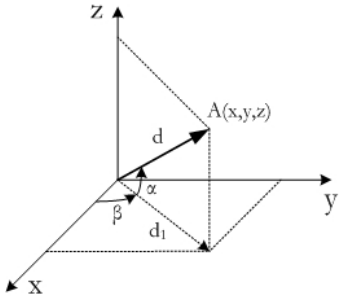

Figure 1. Polar coordinates for a point in space.

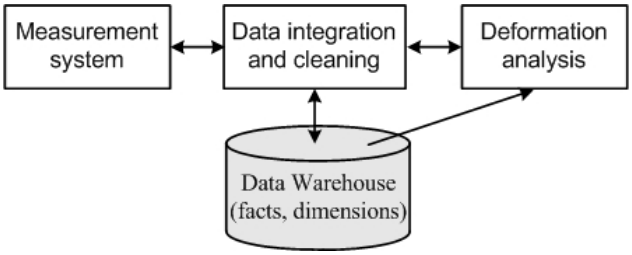

Figure 2. Main components of a geodesic monitoring system.

implementation) may be any type of object (class) [10].

ESBs are capable of executing communication protocol transformation, data format transformation and interaction model transformation to ensure the interoperability of the software components, which otherwise would be incompatible [11]. This aspect leads to loosely coupled systems which are able to communicate, and which can be modified and extended with minimal effort.

The goal of this paper is to present the development of an ESB for a geodesic monitoring system which provides all the main advantages of a service oriented architecture: flexibility, smaller reaction time and reusability. The ESB described in this article has been implemented using the Spring Integration portfolio but the concepts presented are valid for any service bus [12]. It does not only decouple the main components of the system, but it also provides data transformation and validation, it inserts the data into the data warehouse, and provides even preliminary analysis functions. The preliminary analysis results are published on a free accessible website (twitter).

In the next section we present in greater detail the monitoring system, emphasizing its data formats. Next, we will describe the ESB solution with its chains and components, the error management at bus level, and a simplified version of the data warehouse structure. Finally, we will present the results, and draw some conclusions on our work.

II. METHODS

A. Geodesic monitoring systems

The geodesic monitoring system considered for our research contains three tachymeters, each of them monitoring fifteen points. The positions of the tachymeters are related to a fix reference frame. As for every such system, the atmospheric conditions can have a significant influence on the results, and, hence a temperature and a humidity sensor have also been installed.

As the structure and the working principles of the measuring system do not represent the goal of this article, we will concentrate on the outputs of the system, and on the further processing steps performed through the middleware solution.

The geodesic monitoring system outputs three types of files:

- a *.xml* file, called observation file which contains the positions of a tachymeter and of the points monitored by it;
- a *.txt* file which contains the current temperature;
- a *.txt* file which contains the current humidity.

Fig. 3 displays the *.xsd* file (schema file), which describes the structure of the *.xml* file containing the positions of the points. Beyond the exact time and date of the observation, the file also contains the exact time and date of the moment in time when the position of each point was determined (these moments in time are also included because the positions of the points are determined sequentially). The absolute moment in time is not crucial, but, as described below, the time differences between the measurements are used by a filter built inside the ESB solution in order to validate the observation.

Each of the text files contains a single measurement. The temperature and the humidity are determined more often than the positions of the monitored points and as a result there will be several such files corresponding to a single observation file. Again the ESB will provide a solution for this, and correlate all of the measurements.

B. Enterprise service bus solution

Fig. 4 displays the main part of the enterprise service bus designed for this project [13], [14]. It contains three main chains, two of them being combined into a single one during the processing. A Java class is associated to each ESB component (name of the corresponding Java class is

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="
                    xmlns:tns="http://www.example.org/Observa

<xs:element name="observation">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="date_time">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="day" type="xs:integer"/>
      <xs:element name="month" type="xs:integer"/>
      <xs:element name="year" type="xs:integer"/>
      <xs:element name="hour" type="xs:integer"/>
      <xs:element name="minute" type="xs:integer"/>
      <xs:element name="second" type="xs:integer"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="tachymeter">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="pos_x" type="xs:decimal"/>
      <xs:element name="pos_y" type="xs:decimal"/>
      <xs:element name="pos_z" type="xs:decimal"/>
     </xs:sequence>
     <xs:attribute name="id" type="xs:integer" use="required"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="point" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="pos_x" type="xs:decimal"/>
      <xs:element name="pos_y" type="xs:decimal"/>
      <xs:element name="pos_z" type="xs:decimal"/>
      <xs:element type="date">
       <xs:complexType>
        <xs:sequence>
         <xs:element type="day" type="xs:integer"/>
         <xs:element type="month" type="xs:integer"/>
         <xs:element type="year" type="xs:integer"/>
         <xs:element type="hour" type="xs:integer"/>
         <xs:element type="minute" type="xs:integer"/>
         <xs:element type="second" type="xs:integer"/>
        <xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="id" type="xs:integer" use="required"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</schema>
```

Figure 3. XSD file for the observations.

displayed above each ESB component). Next, we will briefly describe the three chains emphasizing the main parts of the bus.

The configuration of the ESB is defined in an XML file, called esb-config.xml (fig. 5 displays a small part of the file).

The first chain starts with a file-adapter, which periodically polls the file system for new *.xml* observation files, and sends the files as payloads of messages through the output channel. Next, there is a transformer which turns the *.xml* file into an instance of a Java class (using the DOM API). Afterwards, the message arrives at a filter which discards the observation if the greatest time difference between the measurement of two points is too large (if the time difference is greater than 400 seconds, i.e. the average time interval between two measurements is greater than around 28.5 seconds). The exact value of 400 seconds has been chosen together with a specialist in geodesic systems and is important for the deformation analysis whose accuracy decreases if the specified time is exceeded. Afterwards, a transformer translates the positions of the points from the local coordinate system of the tachymeter to the absolute coordinate system (the reference frame). To explain the aggregator which follows, we first describe the second chain in the ESB.

The second chain starts also with a file adapter which polls the file system for new *.txt* temperature or humidity file, and sends them as payloads of messages to the router. The router separates the temperature and humidity files, and sends them through separate channels. As a result, the processing which follows, tales place in parallel. During the processing the files are turned into instances of Java classes, the data are stored into the database, and the instances are then sent out to queue channels, which store them until another thread or component reads them from there (the maximum storing capacity is of 100 messages). The aggregator of the first chain reads all messages from the queue channels and associates the most appropriate temperature and humidity samples with the tachymeter observation (the samples which are closest in time to the observation moment). Fig. 6 displays a part of the code of the aggregator. Further on, the observation is stored (i.e. the tachymeter position together with the date and time, and the temperature and humidity samples). Next, there is a splitter which is divides the large payload (the list of points) into separate messages, each containing a single point. Thus, the processing which follows can be more concentrated.

The last chain of the ESB displayed in fig. 3 sends messages periodically to twitter, where we have created an

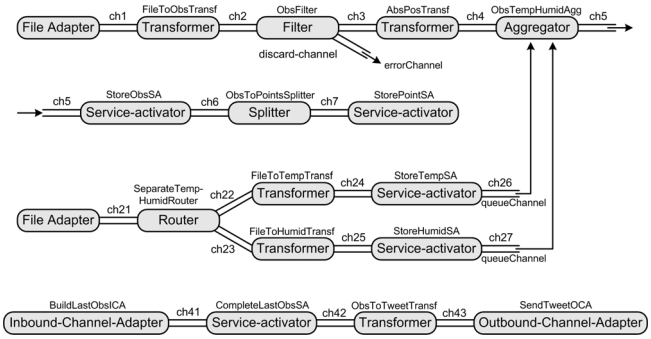

Figure 4. ESB Structure.

```
<channel id="file_obs_c1"/>
<channel id="obs_c2"/>
<channel id="filtered_obs_c3"/>
<channel id="abs_obs_c4"/>
<channel id="complete_obs_c5"/>
<channel id="stored_obs_c6"/>
<channel id="splitted_obs_c7"/>
<channel id="discarded_obs_c8"/>
<file:inbound-channel-adapter directory="c:/GeoMonSys/Observations/"
    channel="file_obs_c1"
    filename-pattern="([a-z])*.xml">
    <poller default="true">
        <interval-trigger time-unit="SECONDS" interval="5"/>
    </poller>
</file:inbound-channel-adapter>
<transformer input-channel="file_obs_c1"
    ref="file_to_obs_transformer"
    output-channel="obs_c2"/>
<filter input-channel="obs_c2"
    ref="obs_filter"
    output-channel="filtered_obs_c3"/>
<transformer input-channel="filtered_obs_c3"
    ref="absolute_pos_transformer"
    output-channel="abs_obs_c4"/>
<service-activator input-channel="abs_obs_c4"
    ref="obs_t_h_aggregator"
    output-channel="complete_obs_c5"/>
<service-activator input-channel="complete_obs_c5"
    ref="store_obs"
    output-channel="stored_obs_c6"/>
<splitter input-channel="stored_obs_c6"
    ref="obs_to_points"
    output-channel="splitted_obs_c7" />
<service-activator input-channel="splitted_obs_c7"
    ref="store_point"/>
```

Figure 5. XML configuration of the ESB.

account. Twitter allows users to broadcast messages (a status or a tweet) to all people who have subscribed to these status updates (the updates are limited to 140 characters). We have chosen to send messages every six hours regarding the status of the monitoring system. Every message contains the temperature and humidity of the last observation, the last known positions of the three tachymeters and the three greatest displacements of the monitored points.

Because the updates are limited to 140 characters the transformer splits the message into two parts and the outbound channel adapter sends them at an interval of one minute (actually the outbound channel adapter will be blocked for 60 seconds but this has no effect on the rest of

```
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 6. The Aggregator service.

the ESB because of its multi-threaded setup). To be able to communicate directly with the twitter account we have used the Twitter4J API which has been developed by Yusuke Yamamoto and is available under the BSD license [15].

Inside an ESB solution, the error management cannot always be accomplished through simple try-catch, language-level constructions. It is advisable to collect all of the errors on an error channel, and then send them to separate processing blocks, which can act correspondingly [12], [16]. In fig. 7 one can see that we have used a router to distinguish between four error cases which could not be solved at language-level (the second error is sent by the filter shown in fig. 7 through the discard-channel and through the error channel). For each of these errors a separate Java class has been written which concentrates solely on handling the specific error. In response to most of the errors indicated in fig. 7, messages will be sent to one or more maintenance engineers. There may be a problem in the measuring system because the *.xml* file does not correspond to the schema file, the tachymeter may have a problem if the measurement time is too long, one of the temperature or humidity sensors may not work properly, or the connection to twitter may be interrupted.

The data integration and cleaning component stores the processing results in a database (fig. 8) which operates as a data warehouse. Thus, once the data of the observations and of the points are stored, they will never be deleted and the tables can be divided into fact and dimension tables [17]. Thus, the deformation analysis process, which is accomplished based on the data in the database, has access to the all information gathered during the monitoring of the object. The structure of the tables has been normalized to the third normal form, thus guaranteeing the lack of redundancy and very easy maintenance. One can see that there are several connections between the tables, either one-to-one or one-to-many. For object relational mapping, and as a persistence management solution, we have used the Hibernate framework.

II. Results

The testing phase lasted around 72 hours. A pair of messages has been sent every six hours: the first one contains the positions of the tachymeters, and the latest temperature and humidity, and the second one displays the points with the greatest displacements since the previous message was posted. Fig. 9 displays the position of tachymeter one on the three axes during the testing phase. The data for these measurements have been recorded during one of the various test sessions carried out in our laboratory.

Further on, no errors or exceptions were found during the testing phase. During this period of time we have also recorded the processing times for the three processing chains displayed in fig. 4 (the end of the second chain has been considered to be the pair of queue channels). They are displayed in fig. 10.

One can see that there are no remarkable delays during the processing, and the differences from case to case are very small. The processing times of the first chain are the greatest, which is mainly caused by the great number of services in the first chain. The third chain, although consisting of only four services (opposed to the seven services of the first chain), leads to very similar times. This is due to the greater computational complexity of the services in this chain. Finally, the second chain requires the least processing time.

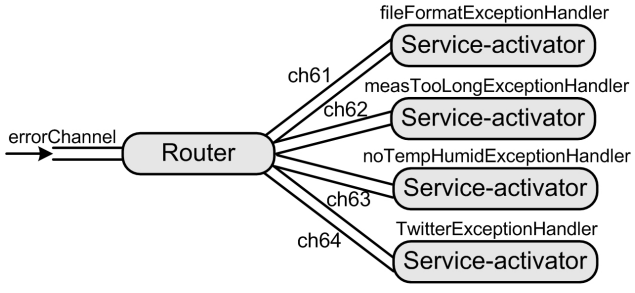

Figure 7. The error management chain.

IV. Conclusions

Throughout the paper we have presented a new approach for a geodesic monitoring system. The solution, which is based on a service-oriented architecture, allows for a separate development of the main components in the system (fig. 2). Hence, a first advantage of this solution is the shorter development and testing time, observed due to the loosely coupled nature of the system.

A second important advantage which we have emphasized is the improved software reusability. If one wants to build a geodesic monitoring system for another object, most of the components can be reintegrated without changes, or with only minor modifications. As a result not only new systems can be set up very fast but also existing ones can be simply adapted to new requirements.

Thus we can draw a first conclusion, namely that our system provides all the main advantages of a service oriented architecture, advantages which were presented in the introductory section: flexibility, smaller reaction time, and reusability. The field of geodesic monitoring, with its easily distinguishable but yet interconnected components, is perfectly suited for such an approach. Every service displayed in fig. 7 performs a small part of the processing [18]. Thus, even the development of these components has taken place in parallel, once the tasks for each component were precisely defined.

To combine the developed services into an end-to-end application (the data integration and cleaning component displayed in fig. 2) the easiest way is to build an enterprise service bus solution. We have chosen to use the Spring

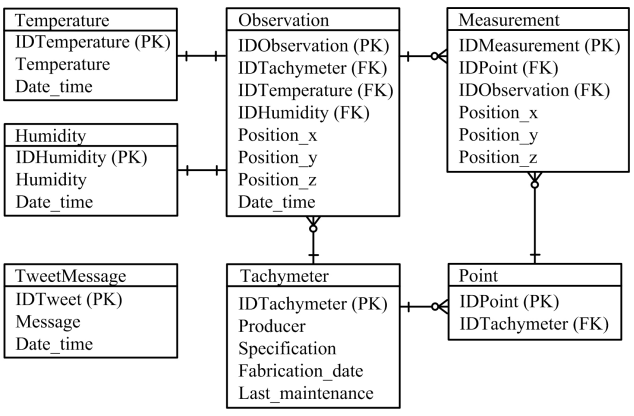

Figure 8. The database structure.

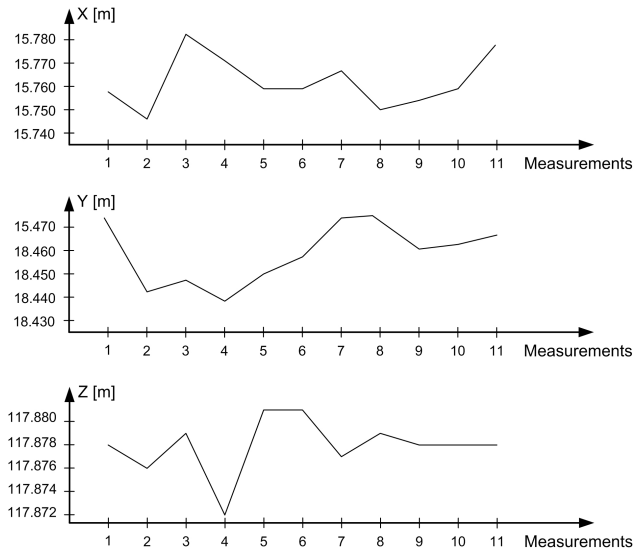

Figure 9. The position of tachymeter 1 during the testing phase.

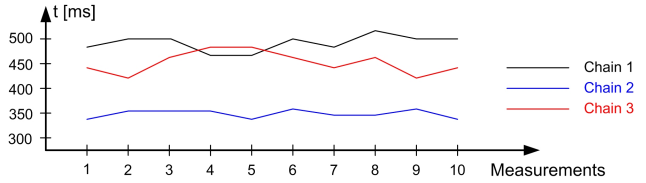

Figure 10. The processing times of the three chains.

Integration ESB because of its simple and straightforward configuration through an xml file (fig. 5). Furthermore, the services have been set up as simple POJO classes with specific annotations. In our solution reusability is adopted at different levels:

- service level: every service can be reused regardless of context;
- chain level: every sequence of services can be reused without needing to take into consideration the services which appear in the chain.

Thus, another advantage of our approach is the varying granularity of reusability.

What we also found to be very interesting and useful is the error management in an ESB solution, because the errors cannot always be treated at language level, they can also be treated at bus level through special services.

Finally, we have to mention that in the development phase we have not only profited from the facilities provided by the Spring Integration portfolio, but also in general from the Spring framework [19]. A good example is the aggregator shown in fig. 6, where the dependency-injection facility provided easy access to the two queue channels of the second chain.

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