Reusing Existing Object-oriented Code as Web Services in a SOA

Harry M. Sneed
ANECON GmbH
Vienna, Austria
Harry.sneed@t-online.de

Chris Verhoef
Free University
Amsterdam, NL
x@cs.vu.nl

Stephan H. Sneed
MetaSonic AG
Pfafenhofen an der Ilm, Germany
Stephan.sneed@metasonic.de

Abstract: This paper describes the reuse of existing classes and methods in an existing object-oriented system as web services in a service-oriented architecture. The approach presented here identifies the interfaces and public methods which can be invoked from outside and generates a WSDL interface to access them. It is a bottom-up approach to creating web services which allows business processes to reuse existing functionality. The tool SoftReuse described in this paper not only generates interfaces to existing public methods in Java and C# code, but also generates a visual documentation of those interfaces and test scripts for testing them. The test scripts use assertions to generate service requests as well as to validate the service responses. The goal is to build the reused services into new S-BPM business processes. This is an extension of the work already made with procedural languages and presented at a previous MESOCA Workshop.

Keywords: *Reverse engineering, Software Reuse, Object-oriented Systems, Web-Services, SOA, Business Processes, Testing Web Services, Java, C#, BPEL.*

I. BACKGROUND OF THIS WORK

In the paper entitled "Linking Legacy Services to a Business Process Model" H. Sneed, S. Sneed and S. Schedl describe how legacy COBOL programs can be reverse engineered to a business process model. The purpose of that work was to model the legacy programs in a business process notation for better understanding them. The emphasis there was on program comprehension, i.e. presenting the functionality of the legacy programs within the context of a business process model. To achieve that the legacy programs had to be wrapped and a web service interface description generated. The biggest barrier to comprehending the code was the naming of the procedures and their parameters. In COBOL and PL/I mnemonic labels are often used to identify procedures and data variables. The meaning of these abbreviated labels is not obvious to a person unfamiliar with their origin. So in the solution proposed, renaming tables were used to convert short name into long ones to appear in the business model. The WSDL interface definition extracted from the legacy program was then imported by the S-BPM tools for modeling business processes. In this way the interfaces to the legacy COBOL code were exposed and represented in a business process model. These exposed interfaces could then be used to incorporate their functionality into the existing business process model. As such this was a pure reverse engineering project, going back from the code to the model [1].

The work described here serves a different purpose, namely to reuse legacy code as executable web services. In this respect, it is more a reengineering project. It also deals with programming languages other than COBOL and PL/I, namely the object-oriented languages C++, C# and Java. The work goes back a long way to the very beginning of the object-oriented movement. At that time developers were faced with the problem of what to do with the existing procedural code. It was Wally Dietrich who suggested at an OOPSLA conference in 1988 to wrap it [2]. By wrapping it behind an object interface, a legacy program could be reused as an object among other objects in an object-oriented architecture. Of course this was not so simple. The legacy source code had to be altered to fit to the interface. In a paper entitled "Encapsulating Legacy software for Reuse in Client/Server Systems" published in 1996, H. Sneed described a source reengineering approach to replace file and screen interfaces by call interfaces in order to access the code from another program [3]. This approach was refined and used in later projects to reuse COBOL programs as web services [4].

II. TRANSITION TO SERVICE-ORIENTATION

Now, 20 years after the object-oriented transition, the software world is faced with another transition, that from object-orientation to service orientation. Objects should now be converted to services which can be accessed from anywhere within an enterprise architecture. Besides the fact that services are universally accessible, they are also at a higher level of granularity than objects which encapsulate single data elements or small groups of data such as an address. Services can contain whole applications such as a travel booking with many points of entry. Each entry is defined as an operation. Booking a flight is only one of many travel operations that can be invoked. Each operation has its own parameters, rules and return values.

The problem with objects is that they have a too fine a level of granularity so that there are too many of them, which makes it hard to manage. Typical middle sized applications with less than 200.000 lines of code may have more than 2000 classes. Larger systems have over 5000 classes. On top of that come the many interdependencies between classes – not only the inheritance relations but also the associations – which amount

to several hundred thousand in a larger system. Considering the fact that complexity is a matter of the number of relationships relative to the number of elements of a system, the complexity of object-oriented systems became higher than that of previous procedural systems even though their size was less. Object-Orientation leads to less code, but to greater complexity [5]. That again leads to higher maintenance costs. Every middle-sized application with 200 to 500 kilo statements requires at least 2 to 4 maintenance technicians to maintain it. That is more than what most enterprise can afford. Therefore, there is a need for standard services which can be reused in different contexts.

The high costs of maintaining individual object-oriented systems are the driving force behind the move to service-oriented architectures. The ultimate goal is to have general purpose services which can be shared by many users in multiple applications. That means the service should be constructed from the beginning with this in mind. However, in the transition phase it may be expedient to use existing objects when constructing the services. The user of a service only sees the interface to that service. He does not see what is behind it. Therefore, in constructing a service-oriented architecture it is recommended to reuse existing software objects as services when getting started [6]. Later these services can be replaced by newly constructed ones or by standard services taken from the cloud, but as a starting point users can reuse their already existing software objects to fulfill the functionality offered by a service.

III. APPROACHES FOR MIGRATING TO SOA

A service-oriented architecture is intended to support the enterprise business process. It is the task of the IT department to provide the services. It is up to the user business departments to model their processes. In the end the two worlds should meet. There are two ways of reaching this end:

- Top down from the process model to the services
- Bottom-up from the services to the process model.

The top-down approach implies that first the business processes are modeled and then the services are implemented to fit to that model. The service developers customize their software to satisfy the requirements of the model. Any time the model is changed the underlying services are changed to follow the model. This can also be referred to as the model-driven approach [7].

The bottom-up approach assumes that first the services exist and then the business processes are modeled in a way that they fit to the existing services. This way the same common services can be reused to satisfy the requirements of different business processes. Not only can standard services be used but also existing legacy services. When the standard or legacy services change the business process model has to be adapted to reflect that change. This is called the service-driven approach. For this approach, the services must first be collected and made available. A prime source for web services is as has been pointed out in previous papers the existing code base. [8]. Web Services can be identified in and extracted from existing source code, whether it is procedural or object-oriented (see Figure 1).

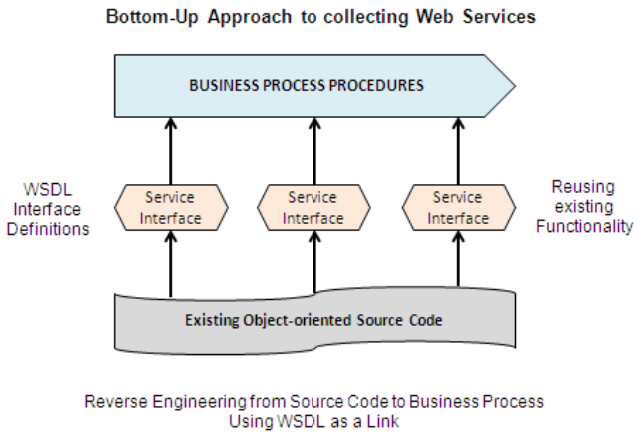

Figure 1: Bottom-Up Approach to collecting Web Services

Of course business analysts prefer the top-down model as long as they are not paying for the development of the services, since this approach gives them the maximum freedom to shape the business processes any way they want. The bottom-up approach is preferred by the IT departments since it allows them to optimize their service offer. In both cases the process model and the services have to be joined together both physically and logically. Physically they are joined by a message sending mechanism or a service invocation. The business process sends a request to the desired service and receives back a response. This communication is normally handled by an enterprise service bus. Logically they are joined by connecting the business process model with the model of the underlying services. By linking the two model descriptions together it becomes possible to trace changes from one to another and to make an impact analysis on the effects of change to one model on that of the other model. The linkage of the two models is an essential prerequisite to the maintenance and evolution of the service architecture as a whole.

Business process models are expressed in some modeling language such as EPC, BPMN, S-BPM or a domain specific language [9]. Business process modeling languages are conceived to depict subjects, objects and events which occur within the business world. Their entities are the business processes, the business actors, the business objects, the business rules and the business events or process steps. With these entities they express the view of the business analyst on the SOA system. From the business process model executable processes are generated in a language such as BPEL or Java Script. This process code is intended as a driver and integrator of the underlying services.

The language of the services is the programming language with which they are implemented in. It could be Java, C#, COBOL, PL/I or any other programming language. It could also be in a unified modeling language like UML. Newer applications may be depicted in UML but that is seldom the case. Older applications are described only by the programming language they are written in. That means, in order to obtain a description of the existing software it is necessary to extract it from the code by means of reverse engineering. As once noted by Allan Perlis the only true description of a program is the code itself [10].

IV. WRAPPING OBJECT-ORIENTED CODE

The difference between the techniques used here and those described in previous publications lies in the way the service interfaces are constructed and made available. In the wrapping of COBOL programs the COBOL code has to be reengineered to provide callable entry-points. In the case of object-oriented code, the entry-points are already there. They are the public methods which have an API type interface and can be invoked from outside with the required input parameter types. If constructed properly these public methods will have a single point of return where they give back a data result. The result may be a single variable or a group of variables, an object. It even may be the reference to another method. Each returned result has its own type, specified in the method declaration.

The approach pursued here foresees three language transformations. In the first transformation the method interfaces are transformed into a relational table. In the second transformation a WSDL interface definition is generated from that relational table. In the third transformation, both a BPEL business process to use the service and a test script to test the service are produced from the WSDL service interface definition (see Figure 2).

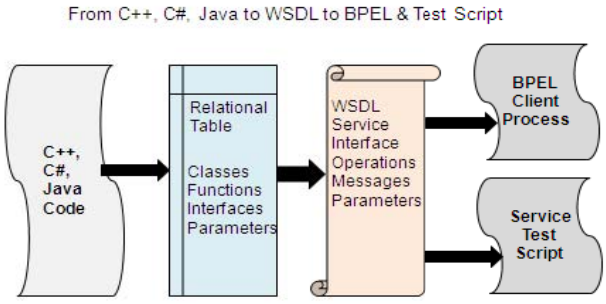

Figure 2: Interface Code Transformations

In automatically generating web-service interfaces from existing object-oriented code, it is necessary to parse that code to identify all publically declared methods. It is relatively easy to convert these Java C++ or C# method interfaces over into a WSDL operation interface. The name of the method becomes the name of the operation and the names and types of the arguments become the input parameters in WSDL. Problematic is the conversion of the return results. Since there may be several return statements within a single method, each returning another result of the same type, these have to be collected together and converted into a set of alternate output parameters in WSDL. The caller will recognize which the actual result is, since the others will be empty. The data type declarations in the WSDL schema will be taken from the argument and result types defined in the method declaration.

<resultType> <methodName> (<arguments>)

Another problem comes up with the polymorphic methods. Methods with the same name can appear in many classes. Therefore it is necessary to qualify the operation names in WSDL with the names of the classes to which the methods belong. Thus, from the viewpoint of the service user they are different operations. For the service user it is only necessary to assign the arguments and to invoke the operation as is done in BPEL. Of course the service user must first invoke the constructor operation before starting to invoke the others. For this, a special constructor operation for each reused object is inserted into the service interface. This is another difference to procedural wrapping where the program state is singular and static. In the case of objects, there may be multiple instances and the service user must know which instance is currently active. For that an object identifier is required as a prefix to the operation name. If the object is derived from a higher order object, i.e. it is inherited; the qualifying name must include the full path to that operation, e.g.

<class_A.class_A1.class_A11.Operationname >

The names of the superordinate classes are collected by tree walking up the hierarchy of classes contained within the system. This may result in very long operation names, but it is necessary to uniquely identify each and every operation. Identifying all of the outputs of an operation presents another challenge. As already mentioned before, the return statements may appear anywhere within the method being reused. They must be collected together with their types to define a union of alternate output variables. It is possible to declare redefined data types in XML using the choice statement meaning that the data element is either one or the other. The data type “return-result” may be defined in the XSD schema as being:

```
<xsd:complexType name = “returnvalue”
        <xsd:choice>
                <xsd:element name = “resulta”>
                <xsd:element name = “resultb”>
                <xsd:element name = “resultc”>
        </xsd:choice>
</xsd:complexType>
```

The invoking business process procedure must know which of the return value types to expect.

V. USING A REPOSITORY TO STORE INTERFACES

Rather than generate service interface definitions directly from the source code, the method used here is to first create a repository from that source code. The repository contains an entry for every significant element of the code – components, sources, classes, methods, interfaces and attributes. The entries are actually rows in a relational database. Each row depicts a binary relationship between two elements. Elements are identified by type and by name. There is a fixed set of predefined relationship types such as "owns", "uses", "calls", "inherits" etc. Having the contents of the code in this format makes it much easier to process them, especially the tree walking up the inheritance hierarchy [11].

```
CLAS;ThisClass;owns;FUNC;ThisMethod
FUNC;ThisMethod;uses;DATA;ThisData
FUNC;ThisMethod;call;FUNC;ThatClass.ThatMethod
FUNC;ThisMethod;retr;DATA;ReturnValue
```

When a method is defined as public, its name and arguments are stored as an interface in the repository. The same applies to publically declared interfaces. When a return statement occurs, it is stored as a return from that method in which it occurs together with the name of the returned value.

```
returnTypeZ Method1 (typeA Argument1, typeB Argument2)
{
if (Argument1 > Argument2)
        return resultX;
else
        return resultY;
}
```

will result in no less than ten entries to the database table.

```
CLAS;ThisClass;owns;FUNC;Method1
FUNC;Method1;uses;INTR;Method1
INTR;Method1;recv;PARM;Argument1
PARM;Argument1;uses;TYPE;typeA
INTR;Method1;recv;PARM;Argument2
PARM;Argument1;uses;TYPE;typeB
INTER;Method1;send;PARM;resultX
PARM;resultX;uses;TYPE;returntypeZ
INTER;Method1;send;PARM;resultY
PARM;resultZ;uses;TYPE;returntypeZ
```

Method1 will become an operation with the input parameters Argument1 and Argument2, and the output parameters resultX and resultY.

There are many entries in the code repository for all methods, attributes, classes, etc. but in generating a service interface only a subset of those entries are required, namely the classes, the interfaces, the parameters and the returns. This is enough to create the operation definitions. On top of that the data types are required to create the data definition schema with the variables referred to in the operations as inputs and outputs. Since inputs and outputs may be data structures, the entire data hierarchy must be depicted in the repository including arrays and unions.

Capturing the code structure in a repository not only supports the creation of interfaces to access the code. It also supports the reengineering and redocumentation of the code. So there are many reasons for preserving the code structure in a repository. Each time the code is changed the repository is updated to reflect that change. The code repository is a prerequisite for many kinds of activities including wrapping. It should be noted that Riebisch and his team at Ilmenau also built up a repository to support impact analysis [12].

VI. TOOLS FOR MINING WEB SERVICES

There are four different tools involved in mining object-oriented code to extract web services for reuse in a SOA.

- SoftRedoc for processing the C++, C# and Java Code
- WSDLGen for generating WSDL interface definitions
- SoftReuse for visualizing the service interfaces
- BPELGen for generating business process code.

SoftRedoc parses the code to identify the public methods and interfaces and to convert them into a table of cases and operations with their parameters and return types. It creates a CSV table for each component.

WSDLGen processes the CSV tables and creates a WSDL service interface definition for each component which includes the data declarations, the operations, the messages and the parameters for that component.

SoftReuse processes the WSDL interface definition to create an interface repository for viewing and editing the service interface elements. For this it has a graphical user interface to a relational database containing the services, operations, messages and parameters.

BPELGen processes the interface repository to locate a particular service and to generate for that service a BPEL process definition to invoke it as well as a test script procedure to test it.

The tools make up a chain from the source code up to the business process model and the corresponding test procedures. The input is the source code, the outputs are the BPEL procedures and the test scripts for testing web services (see Figure 3).

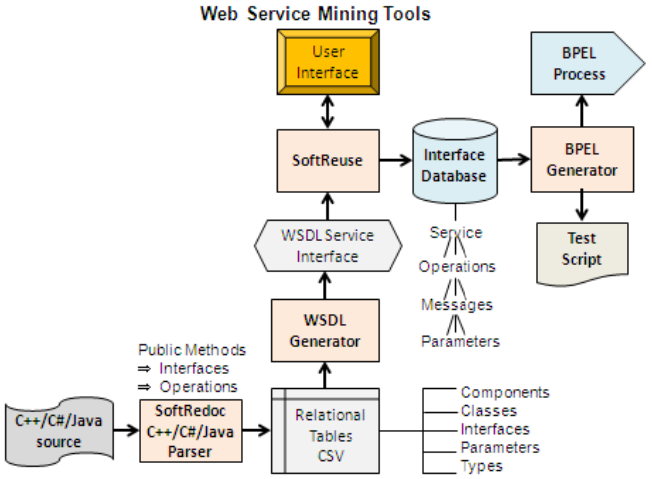

Figure 3: Web Service Mining Tools

VII. STEPS TO CREATING AN INTERFACE REPOSITORY

There are in all eight steps to wrapping existing code for reuse as web services:

a) Parsing the code to populate a repository
b) Processing the repository to collect all data types referred to
c) Processing the repository to collect all entries to and exists from the code
d) Generating the type definitions of the message data
e) Generating the WSDL operations and messages
f) Documenting the interfaces to the code
g) Generating a BPEL procedure
h) Generating a test script.

A. Populate the Code Repository

The first step is to parse the code to populate a repository. Every statement needs to be analyzed whether it is relevant to the structure of the code. If so, it becomes an entry in the repository table. In the end the repository is a mirror of the code structure, an abstract syntax tree without all of the details.

```
public  void Bausparer ( VarChar   cRole,
                         Nummer    nVersion,
                         TimeStamp tsReadTime,
                         int       iSeqNumber,
                         Nummer    nGrdNr,
                         Nummer    nBBZS,
                         VarChar   cNachname,
                         VarChar   cVorname,
                         VarChar   cKzAnrede,
                         VarChar   cKzTitel,
                         Datum     dGeburtsdatum,
                         boolean   bKzArchiv,
                         Adresse   oAdresse
                         ) {
```

B. Process the Repository in the First Pass

The second step is to go through the repository and collect all of the entries = public interfaces, and exits = returns. These are marshaled into an internal operation table for further processing.

```
CLAS;Bausparer      ;OWNS;FUNC;Bausparer
FUNC;Bausparer     ;USES;INTR;Bausparer
INTR;Bausparer      ;RECV;PARM;cRole
PARM;cRole           ;USES;TYPE;VarChar
INTR;Bausparer      ;RECV;PARM;nVersion
PARM;nVersion        ;USES;TYPE;Nummer
INTR;Bausparer      ;RECV;PARM;tsReadTime
PARM;tsReadTime   ;USES;TYPE;TimeStamp
INTR;Bausparer      ;RECV;PARM;iSeqNumber
PARM;iSeqNumber  ;USES;TYPE;int
INTR;Bausparer      ;RECV;PARM;nGrdNr
PARM;iGrdNr          ;USES;TYPE;Nummer
INTR;Bausparer      ;RECV;PARM;nBBZS
PARM;iBBZS          ;USES;TYPE;Nummer
```

C. Process the Repository in the Second Pass

The third step is to go through the repository and collect all of the data types referred to in the interface and return statements, i.e. the elementary variables, structures, arrays and unions. These definitions are needed to create the WSDL data schema.

D. Generate the XSD Data Schema

The fourth step is to generate the XSD data schema from the data types in the internal data table. If the references are made to data types in external schemas, then these schemas are copied into the interface definition source, so that in the end the source includes all data definitions. If the external schemas are changed they will not effect this interface definition.

```
<complexType name = "Person_Input_Params">
  <sequence>
    <element name = "cRole " type = "ns:string"/>
    <element name = "nVersion " type = "ns:iNummer"/>
    <element name = "tsReadTime " type = “TimeStam"/>
    <element name = "iSeqNumber " type = "ns:int"/>
    <element name = "nBBZS " type = "ns:string"/>
    <element name = "dGeburtsdatum " type = “Datum"/>
    <element name = "nGrdNr " type = "ns:Number"/>
    <element name = "cNachname " type = "ns:string"/>
    <element name = "cVorname " type = "ns:string"/>
    <element name = "cKzSparer" type = "ns:string"/>
    <element name = "bKzArchiv " type = "ns:boolean"/>
    <element name = "cOrt " type = "ns:string"/>
    <element name = "cPlz " type = "ns:Nummer"/>
    <element name = "cStrasse" type = "ns:string"/>
  </sequence>
 </complexType>
```

E. Generate the WSDL Operations and Messages

The fifth step is to generate the messages and operations. First the messages are created with parts referring to the data types defined in the data schema or to elementary XML types. Then the operations are created with inputs and outputs. The inputs refer to the messages taken from the arguments of the function called. The outputs refer to the predefined return results. The operation itself bears the name of the qualified method, i.e. with the concatenated names of the classes to which it belongs. Finally the SOAP binding is generated to complete the interface definition and a service name assigned, the service name is the name of the component or package in which the operations are located.

```
<portType name="BAUSPAR_DBBSAG">
  <operation name="Bausparer">
    <input message="tns:Bausparer_Input"/>
    <output message="tns:Bausparer_Output"/>
  </operation>
  <operation name="BausparerMarshalling">
    <input message="tns:BausparerMarshalling_Input"/>
    <output message="BausparerMarshalling_Output"/>
  <operation name="Person">
    <input message="tns:Person_Input"/>
    <output message="tns:Person_Output"/>
  </operation>
</portType>
```

F. Document the Interface Definitions

The sixth step is to generate a documentation of the generated web service interface. The interface is displayed in a tabular form with four columns. In the first column are the service names which are equivalent to the interface names. To view the contents of the interface, the user has to select one. In the second column are the names of the operations belonging to the service selected. Here too, the user can select one. In the third column are the input and output parameters of the operation selected. When the user selects one, the data elements of the input or output are displayed in the fourth column with level, name and type.

In each column the user has the possibility of viewing all entries in that column, e.g. all operations or all data elements, from which he can select one. The user also has the possibility here to change the names. If he alters a name, that name will be replaced in all instances of that column. This is particularly important for renaming the data elements from old procedural programs. To be useful in a business process model the names should be speaking ones.

This hierarchical view of the service interfaces is very important to the reuse of the services, since it gives the potential user an insight into the contents of the service. He can readily see what goes in and what comes out. This makes it easier to select, which existing services he wants to use. Other WSDL interfaces not taken from the existing code may also be documented in this form, thus giving the user an overall view of all services available to him in a service-oriented architecture (see Figure 4)

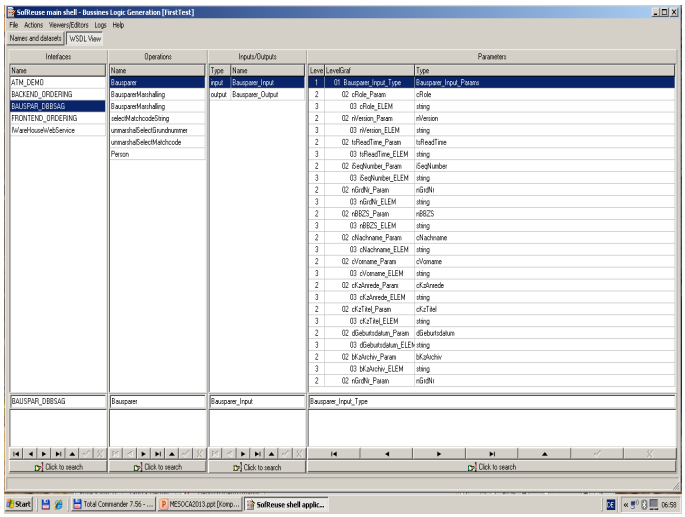

Figure 4: View of Interface Definition

G. Generating a BPEL Procedure

The seventh step of this reverse engineering process is optional. It is to generate a BPEL procedure for accessing the recaptured web services. Not all users will want to use BPEL4WS to implement their business processes, but that language is a widely used standard which can act as a model for other solutions [13]. Therefore, the user can have a BPEL framework procedure created for each service he wants to use.

The generated BPEL code has four parts:

- partner links
- variables
- assign inputs
- invocation statement
- assign outputs.

The partner links are created from the roles defined by the user. The variables are declared using the data elements from the interface definition. The assignment of data to the operation inputs can only be partially generated since the tool only know the data in the interface and not the data in the user process. For the source of the inputs, dummy names are used which the user has to replace with his own names. The invocation statement is generated in full to invoke the selected operation. After the return from the service the BPEL procedure has to take over the return results. For this additional assign statements are required to copy the service outputs to the local variables of the BPEL procedure. With this procedure, the user can see exactly how the web service is to be called.

```
process name = “Bausparen"
<PartnerLink name = “BausparerUser"
    partnerLinkType = "calender:User"
    myRole = "Provider"
    partnerRole = "User" />
</partnerLinks>
 <variables>
    <!-- inputs to Bausparer Functions -->
        <variable name = " cRole" Type = “string"/>
        <variable name = “nGrdNr" Type = “Nummer"/>
        <variable name = “nBBZS” Type = “Nummer"/>
    <!-- outputs from Bausparer Functions -->
        <variable name = "ResponseCode" Type = “int"/>
        <variable name = “ThisPerson" Type = “Person"/>
    </variables>
    <assign>
      <copy>
        <from variable = “Role" part = “Client" />
        <to variable = “cRole" part = “Bausparer" />
      </copy>
      <copy>
        <from variable = “GrundId" part = “Client" />
        <to variable = “nGrdNr" part = “Bausparer" />
      </copy>
   </assign>
   <!-- call Bauparer Service to provide Person Data -->
    <invoke partnerLink = “BausparerUser"
                    portType = “BausparerStatusPT“
                    operation = “Bausparer"
                     inputVariable = “BausparerRequest"
                    output Variable = “BausparerResponse" />
```

The generation of the invoking business procedure terminates the reverse engineering process which started with the mining of the code for all possible functions. Such a process to reuse existing code makes the transition to a service-oriented architecture faster and cheaper than if the user were to implement the services from the scratch.

H. Building a Bridge to SoaML

In March of 2012 the Object Mangement Group released the Verion 1.0 of a new Service-oriented Modelling Language – SoaML – to model the semantics of software services [14]. This language has since been built into several UML modeling tools including IBM’s Rational Software Architect. For users of these tools it is very important that their service architecture can be modeled with this new standard notation. The tool SofReuse described here has much of the information required to produce that model. It knows the services, the operations, the messages and the parameters. They can be passed over to the SoA modeling tool in the form of an XMI interface file. There they can be represented as a Service Interface prototype as depicted in Figure 5.

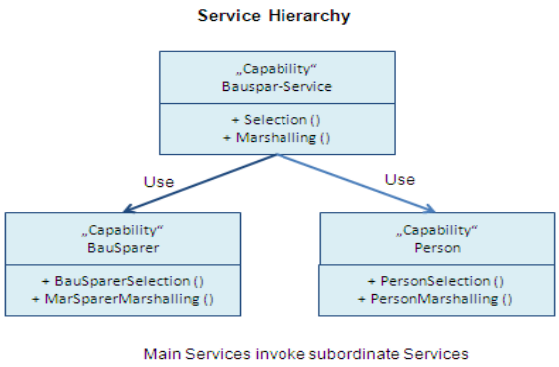

Figure 5: Service Interface Prototype

What is not known is the client side, i.e. what requests will be sent in what order and what callback operations are required. These modeling elements must be filled in by the system architect, but at least the server side of the model is there and can be expanded. This is one of the issues to be dealt with in future work.

I. Creating a Test Script

The final step of the process is to generate a script for testing the encapsulated web service. The script contains assertions for both requests and responses to each of the operations made available. The assertions for the requests assign data to the input parameters. The assertions for the responses validate the output results against the expected values. It is left to the tester to assign specific data values such as strings and numbers. They can of course be randomly generated from the parameter types, but it is better when the test values are taken from the previous tests of the original code. If a test driver such as JUnit was used, the assertions of that test can be taken over. In any case, whether the data is automatically generated, copied from previous tests or assigned manually, it is important to test each and every wrapped operation before making it available in the SOA. There are several tools for testing web services, but the one used here is WebsTest [15].

```
service: BausparerSOAPServerservice;
  if (testcase = „Bausparer001");
//  It should  be possible to select a Bausparer byGrdNr
//  and BBZS
     if ( operation =   „Bausparer");
        if ( request  =   „Bausparer1Request");
           if ( object = "Person" );
              assert inp.CRole   =  “Besitzer";
              assert inp.nGrdNr  = „4711";
              assert inp.bBBZS  = „120036";
              assert inp.tsReadTime  = „201303130700";
              assert inp.iSeqNumber  = "100";
          endObject;
        endRequest ;
        if ( response  =   „Bausparer1Response");
           assert out.$ResponseTime < „1200";
```

```
        if ( object = "return" );
           assert out.ResponseCode  = „00";
           assert out.cNachname = „Schmidt";
           assert out.cVorname  = „Karl";
           assert out.dGeburtsdatum  = „19220420";
         endObject;
      endResponse ;
   endOperation;
endCase;
```

VIII. EXPERIENCE WITH THE REUSE OF OO-CODE

The method described here has been applied to both a DotNet and a Java application. The Java application was for billing the electricity in a large city. The system contained 31 components with 1.654 classes, 15.384 methods and 183.474 statements. Of the 15.384 methods 788 had a public interface. All of these were wrapped and converted to operations in 31 services, one for each Java package. Six of the wrapped operations were invoked to demonstrate the feasibility of the approach. The results returned were correct and the response time was acceptable. Of course it was greater than if the methods were invoked within the Java environment, but that is to be expected.

The DotNet application studied was a system for collecting dues to the Austrian Chamber of Commerce in Vienna. This system had 36 components with 2.606 classes, 16.832 methods and 309.823 statements. Of the 16.832 methods only 492 had a public int4erface and could be accessed from outside. These were converted to operations in 32 WSDL interfaces. All of the interfaces were documented and three operations were tested. Here too, correct results were returned in an acceptable time frame. This demonstrates that the reuse of DotNet code as web services is feasible, provided the methods reused are not dependent on foreign methods in other components. The possibilities of code reuse depend very much on the architecture of the code.

IX RELATED WORK

The wrapping and reuse of existing code as web-services has been a research topic since the emergence of web service technology at the turn of the century. The article by E. Horowitz entitled “Migrating Software to the World Web” which appeared in the IEEE Software Magazine in 1998 marked the beginning of this movement. One of the first studies was made by the RCOST research center at the University of Sannio in the year 2000 and was reported on by Aversano at the CSMR Conference in 2001. The work was devoted to migrating legacy software systems to web applications. [16]

The focus in this early research was on COBOL code. A paper which dealt more with the business benefits of reusing legacy code appeared in the Communications of the ACM in 2002. The authors of that paper –Pinker, Seidmann and Foster – were conceived with strategies for transforming old Economy Firms to e-Businesses. They present several arguments for reusing existing software. [17 = Pinker2002]. A year later the book of Seacord, Plakosh and Lewis was published on the subject of modernizing legacy systems. One of the alternatives for modernizing software was that of wrapping for reuse in a new architecture [18].

In the same year a book appeared in Germany with the title “Web-based Systemintegration” by Sneed and Sneed. It dealt explicitly with the reuse of legacy software components as web services in a service-oriented architecture and presented several early case studies [19]. After that Sneed published one paper after the other on migrating to SOA – “An Incremental Approach to System Replacement and Integration” at CSMR2005 [20], “Integrating Legacy Software into a service-oriented Architecture” at CSMR 2006 [21], “Migrating to Web-Services – A Research Framework” at CMSR 2007 [22], “A Pilot Project for migrating COBOL Code to Web- Services” at WSE 2008 [23], and “SOA Integration as an Alternative to Source Migration” at the SOAME Workshop in 2010 [24]. At the ICSM 2010 Worms reported on the experience of the Swiss Credit Union in migrating to SOA. There too the process involved the reuse of existing PL/I applications. [25]

Ever since the first paper by Aversano in 2001, the RCOST institute had been working on the creation of web services from existing source code.[26] At the CSMR 2006 in Bari, Canfora, Fasolino and Fratollilo presented their study of migrating interactive legacy systems to web services [27]. The University of Salerno took over this topic of web service migration from the RCOST institute after 2008 and has since published several papers and a book on the subject [28]. The U.S. Software Engineering Institute in Pittsburgh has also made a significant contribution to research on migrating to SOA [29]. The ultimate reference for the work in this field is the IGI Global book entitled “Migrating to SOA and Cloud Computing” edited by Litou and Lewis which was published in the last year [30]. Research goes on in this vital area, seen by many as a key technology in moving to SOA and cloud computing.

X. CONCLUSIONS

The conclusion of this paper is that the reuse of existing software components as web services in a service-oriented architecture is essential to the IT field. It must become possible for new business process models to access existing code and to use the functionality embedded therein. The tool supported approach presented here allows business processes to call public methods in Java and C# code by means of a standard WSDL interface automatically generated from the code. Not only does it grant the business process access modeler with a view of the interface, allowing him to select which functions he wants to reuse. The approach has been tested on complex Java and DotNet applications and found to be feasible. The

tools for the reuse approach are available on the ANECON website in Vienna.

REFERENCES

[1] Sneed, H./Sneed, S./Schedl, S.: "Linking Legacy Services to the Business Process Model" in Proc. of MESOCA2012, Ed. Ionita, A. and Lewis, G., Riva de Garde, Sept., 2012

[2] Sneed, H.: "SOA Integration as an Alternative to Source Migration" Proc of SOAME Workshop, Timisoare, Sept. 2010.

[3] Dietrich, W.: "Saving a Legacy System with Objects", Proc. of OOPSLA-88, ACM Press, New York, 1989, p. 54

[4] Sneed, H.: "Encapsulating Legacy Software for Reuse in Client Server Systems" Proc. Of 3rd WCRE, IEEE Computer Society Press, Monterey, CA., Nov., 1996, p. 104

[5] Eierman, M./Dishaw, M.: „Comparison of object-oriented and third generation development languages" Journal of Software Maintenance and Evolution, Vol. 19, No. 1, Jan. 2007, p. 33

[6] Bichler,M./Kwei-Jay, L.: „Service oriented Computing" IEEE Computer, March, 2006, p. 99

[7] Winter, A., Ziemann, J.: "Model-based Migration to Service-oriented Architectures", in Proc. of SOAM Workshop, CSMR-2007, Amsterdam, p. 107.

[8] Sneed, H.: "Encapsulation of Legacy Software – A Technique for reusing Software Components" Annals of Software Engineering, Vol. 9, Baltzer A.G. Amsterdam, 2000, p. 113

[9] Börger, E.: "Approaches to Modeling Business Processes. A CriticalAnalysis of BPMN, Workflow Patterns and YAWL",Journal of Software & Systems Modeling 2011, Band 11, Springer, 2011.

[10] DeMillo, R./ Lipton, R./ Perlis, A.: "Social Proceses and Proofs of Theorems and Programs", Comm. Of ACM, Vol. 22, No. 5, May 1979, p.22

[11] Lam, T.C./Ding, J./Liu, J.-C.: "XML Document Parsing – Operational and Performance Characteristics", IEEE Computer, Sept. 2008, p. 30

[12] Lehnert, S./Farooq, Q./Riebisch, M.: Rule-based Impact Analysis for heterogeneous Software Artifacts", IEEE Proc.of CSMR2013, Genova, March 2013, p. 209

[13] Juric, M.: Business Process Execution Language for Web Services", PACKT Pub. Birmingham, UK, 2004

[14] Fischbach, M./Puschmann, T./ Alt, R.: „Service LifeCycle Management", Wirtschaftsinformatik, Nr. 1, Feb.2013, p. 51

[15] Sneed, H. & Huang, S. (2006): *WSDLTest – A tool for testing Web Services*, Proc. of WSE-2006, IEEE Computer Society Press, Philadelphia, Sept. 2006, p. 14

[16] Aversano, L./Canfora, G./deLucia, A.: "Migrating Legacy System to the Web", in Proc. of CSMR-2001, IEEE Computer Society Press, Lisabon, March 2001, p. 148

[17] Pinker E./Seidmann, A/ Foster, R.: "Strategies for transitioning old economy firms to E-Business", Comm.of ACM, Vol. 45, No. 5, 2002, p. 76

[18] Seacord, R./Plakosh, D./Lewis, G.: Modernizing Legacy Systems, Addison-Wesley, Reading, 2003, p. 120

[19] Sneed, H./Sneed,S.:Web-based System Integration, Vieweg Verlag, Wiesbaden, 2003

[20] Sneed, H.: "An Incremental Approach to System Replacement and Integration", Proc. of European Conference on Software Maintenance & Reengineering, CSMR-2005, IEEE Press, Manchester, March, 2005, p. 196

[21] Sneed, H.: "Integrating legacy Software into a Service oriented Architecture", in Proc. of CSMR-2006, IEEE Computer Society Press, Bari, March 2006, p. 3

[22] Sneed, H. "Migrating to Web Services–A research framework" Proc of SOAM Workshop, CSMR, Amsterdam, March 2007, p. 3.

[23] Sneed, H.: "Experience in extracting Web Services from Legacy Code", Proc. of 8th Workshop on Web Service Evolution (WSE), IEEE Computer Society Press, Beijing, Sept. 2008, p. 72

[24] Sneed, H.: "SOA Integration as an Alternative to Source Migration" Proc of SOAME Workshop, Timisoare, Sept. 2010.

[25] Worms, C.: „Web Servives in a Global bank – An Enterprise Architecture Perspective", Proc. of WSE-2010, IEEE Computer Society Press, Timisoara, Romania, Sept. 2010, p. 1

[26] Bodhuin, T./Guardabascio, E./Tortorella, M.: "Migrating COBOL Systems to the WEB", WCRE-2002, IEEE Computer Society Press, Richmond, Nov. 2002, p. 329

[27] Canfora, G./Fasolino, H./ Frattolillo, G.: "Migrating Interactive Legacy System to Web Services", Proc. of CSMR-2006, IEEE Computer Society Press, Bari, March 2006, p. 23

[28] DeLucia,A/Francse, R./Tottora G./ Vitiello, N.: "A Strategy and Eclipse-bases Environment for the Migration of legacy systems to multi-tier web-based Architectures" Proceedings of 22nd International Conference on Software Maintenance, IEEE CS Press,Philadelphia, PA., Sept. 2006, p.438

[29] G. Lewis, E. Morris, D. Smith.: "Analyzing the Reuse Potential of Migrating Legacy Components to a Service-Oriented Architecture". In Proceedings of the Conference on Software Maintenance and Reengineering, 2006, IEEE CS Press, pp. 15-23.

[30] Ed. Ionita, A./Litoiu, M./Lewis G.: Migrating Legacy Applications – Challenges in Service oriented Architecture and Cloud Computing Environments, IGI Global Pittsburgh, PA., 2012