Linking Legacy Services to the Business Process Model

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

Stefan Schedl
Metasonic AG
Pfaffenhofen an der Ilm, Germany
stefan.schedl@metasonic.de

Stephan H. Sneed
Metasonic AG
Pfaffenhofen an der Ilm, Germany
stephan.sneed@metasonic.de

Abstract: The purpose of the work described here is to support the reuse of existing software systems in a SOA environment by linking a description of existing programs to the overlying business processes. It is one thing to technically wrap the legacy code. It is another matter to connect the code interface definition to the business processes. The *SoftLink* tool is under development to bridge that gap between the business model and the code reality. The crux of the solution is to identify the entry points to the application system and their parameters and to link them to the events in the business process via a WSDL interface. The method is illustrated here on a legacy COBOL application for processing customer orders. From that code interfaces to the events within the subject-oriented business process model are created via a semi-automated transformation. Future development will focus on linking Java and .Net systems as well. This will provide a better basis for the maintenance of SOA systems and allow impact analysis to traverse the border between model and code.

Keywords: *Reverse engineering, legacy code reuse, business process modeling, maintaining SOA systems, impact analysis, COBOL, BPM, BPMN,S-BPM, SOA.*

I. PURPOSE OF THIS WORK

There is a need among business organizations to rapidly implement and evolve their ever changing business processes on the basis of existing software. This software may exist in the form of off-the-shelf components, web services, cloud services or legacy systems inherited from past operations. Service oriented Architectures can be put together from a wide range of software building blocks [1]. The common requirement on all of them is that they be readily available and interoperable. It should be possible to incorporate them quickly into the evolving architecture. It should also be possible to maintain them there at a reasonable cost. This presupposes that their relationships, i.e. dependencies, to other components in the service architecture are well documented. If changes or corrections occur to the business processes, it must be possible to trace those changes quickly and accurately to the code units affected by them. This requires links between business process models and the underlying code units. These links should lead the responsible maintenance engineer directly from the point of change in the business process to the points of correction in the underlying services.

In the case of purchased or rented services it is up to the provider of those services to make the necessary changes and corrections. The timeliness and accuracy of the error corrections and change implementations should be governed by the service level agreement. This has been discussed in previous papers [2]. In the case of legacy services it is the task of the user himself to make the corrections and to insert the changes. For that he needs to know what services are affected and to what code components they belong. In short he needs a roadmap of his SOA architecture to trace changes through that architecture. The work described in this paper is aimed at providing such a roadmap. The goal is to trace entities of the business process model such as subjects, objects and events directly to the services, modules and operations within the code where they are implemented. Once inside the code, it is then possible to capture the code dependency relationships such as inheritance and association in object-oriented code and calling chains and data flow in procedural code. The problem dealt with here is to get from the business process to the code.

The research work depicted here deviates significantly from previous work by the author Harry Sneed in that the goals have been altered. Previously the author was concerned with establishing an interface to legacy components to enable wrapping them in new applications. This work began already in 1996 and has continued on into the present [3]. It led to the development of the tool SoftWrap, which has been used in several projects to wrap COBOL and PL/I modules, projects which have been reported on before [4]. Wrapping involves changing the code and is, therefore, to be considered reengineering. In the approach presented in this paper the code is not altered. It is left as it is. The goal is not to wrap the code but to document it in such a way, that it can be linked to the new business processes now being modeled. As such, this approach should be considered as reverse engineering. It is recreating documentation from existing code [5].

The ultimate goal is to generate a business model from the underlying code. That Ist-Model can then be compared with the Soll-Model developed by the Business Analysts to determine how well the services provided in the code match to the services required by the business process model. This should be done prior to wrapping the existing code for reuse as services. One should know beforehand how close the old code comes to satisfying the new requirements. That becomes clear when one compares the model extracted from the code with the model designed by the analysts. If they match the code can be reused as is. If not it may be necessary to reengineer the code or to alter the business model to fit the code. If they are too far apart it will be wiser to implement new services or to take over readymade ones from outside [6]. It any case the reverse engineered model supports the decision making process.

Thus, the research described here has a dual purpose:

1) To assess the degree to which existing software components satisfy the requirements of a new SOA architecture.
2) To establish links between the existing legacy services and the overlying business processes.

II. LINKING TWO WORLDS TOGETHER

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 fit together. 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.

The bottom-up approach assumes that first the services exist and then the business processes are modeled so as to 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.

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 call. 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 affects 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 [7]. 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 of the SOA system.

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.

There have been numerous approaches to reconvert the logic of legacy code back into a design or even into a specification language. The early work was directed toward deriving functional and data flow models [8]. Later work was devoted to obtaining object models from existing object-oriented code. That work lead to the extraction of complete UML models from existing C++, C# and Java code [9]. In recent years the target language has been UML. There now are many tools for converting Java and C type languages into UML. There are however few tools for converting older languages as COBOL, PL/I and RPG into UML. This is because of the semantic gap between the procedural nature of these languages and the object oriented nature of UML [10].

Another problem is that of the limitations of static analysis. If parameter structures are setup at runtime, as is the case with REST interfaces, it is not possible to capture them statically. However, if the user is aiming at using statically defined WSDL interfaces, then it is possible to create them from the legacy code, since COBOL and most PL/I interfaces are defined statically.

An even greater problem is that of the naming. In all of the approaches to modeling legacy code the main obstacle has always been the names used in the code. It has been a habit of programmers to use mnemonic terms to identify both their data and their procedures. These terms are often only understood by the programmer himself and after a while not even by him. To outsiders they are meaningless. Therefore, it makes little sense to generate modeling diagrams out of such code as long as the data names remain incomprehensible. Only the form is documented but not the content.

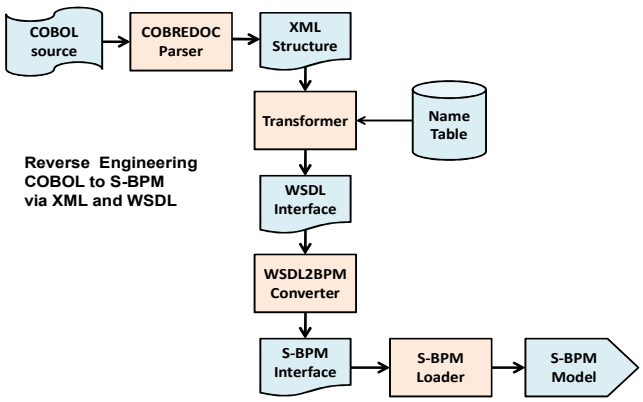
In order to unite the code model with the business process model the names used in the code have to be matched to the names used by the business analysts in their model. This remains the biggest obstacle to a bottom-up approach of logically linking the code with the overlying business model. It

has to be resolved in order to model the interfaces between the two.

III. RESOLVING THE NAMING PROBLEM

Up until now there has been little work in the reverse engineering community on how to alleviate the naming problem. Marcus and Maletic have done pioneering work in using semantic information for promoting program comprehension [11]. This work has been picked up and carried forward by Ducasse and Girba in correlating terms to recognize clusters of information [12]. However, the problem of converting coded labels into meaningful names has yet to be resolved. The key to that resolution seems to lie in recognizing the patterns the original programmers used to assign labels. Normally names are assigned on the basis of some naming pattern [13]. With the help of naming patterns users can be supported in assigning newer meaningful names.

In principle the linking of procedural code like COBOL to a business modeling language in BPMN or S-BPM is a simple mapping process. The entities of the COBOL program are mapped to the entities of the business process. For that a common schema is required. However, in addition to the mapping problem there remains the naming problem. The entity names used in the COBOL code have to be converted into meaningful names and the many constants used in the code have to be assigned symbolic names. Only then can the linking be effective. (see Figure 1: Mapping Process)



In this paper a tool supported five step transformation process is presented which maps COBOL, PL/I and RPG legacy code into the S-BPM modeling language while converting both procedural and data names.

- In the first step the old code is parsed and the syntactic elements stored in symbolic tables.
- In the second step the syntactic elements are converted into an XML document encompassing data types, interfaces and elementary actions. There is one document for every program.
- In the third step the old data and procedural labels are loaded into a term dictionary and the user requested to supply them with new names.
- In the fourth step the renamed elements are converted over into a WSDL service interface definition containing the parameter types, the messages and the operations. There is an operation for every entry in the existing application. In COBOL this corresponds to an online transaction or a batch step. In PL/I and RPG it corresponds to a main procedure [14].
- In the fifth and final step the WSDL interface is converted over into the S-BPM language, whereby the service corresponds to a subject, the messages to messages and the operations to actions.

Since the legacy code entities do not cover all of the entities required by the business process model, the remaining entities have to be inserted by the human user to make the model complete. The end result is a business process model which describes at an abstract level what goes on at the program level.

IV. MODELLING LEGACY CODE

Legacy procedural languages such as COBOL, PL/I and RPG have separately compilable source code members. Their compiled objects are linked together to create a load module or run time unit. One of the modules acts as the main module and calls the others. The linked run time unit is referred to as a program. The parts thereof are referred to as modules. At a program level we have data objects in the form of structures. Structures can be nested, so there are objects within objects, some of which may have multiple occurrences. Some of the objects such as file records, database records and map reports are imported via an input interface. The same or other records of the same type are exported via an output interface. Objects may also be received and returned via a call interface. Objects are as a rule singletons but with an occurrence attribute they can have many instances. Some structures correspond to business objects but most are strictly of a technical nature.

In COBOL and RPG the procedural code is separated from the data descriptions. The procedural part is broken up into procedural blocks, each with its own unique label. Procedural blocks, i.e. paragraphs in COBOL or Begin blocks in PL/I are executed is sequence, but they can also be performed with a return to the next statement or be branched to via a GOTO with no return. In PL/I the procedural blocks may also contain local data which is allocated when the block is invoked and released when the block is terminated. The code within the procedural blocks contains the conditions for executing statements. Some, but not all of these conditions have to do with business logic and can be considered to be business rules. The statements for sending and receiving data, for accessing databases and files and for calling other programs are nested inside the procedural blocks where they can be conditional or non-conditional. Before commencing with the analysis of the code, the access operations to the controlling files and maps are wrapped as depicted below.

WRAP* READ DISPATCH-FILE

```
WRAP* AT END MOVE '09' TO DISPATCH-STATUS
WRAP  MOVE 'RD' TO XML-FUNCODE
WRAP  MOVE ZEROES TO XML-RETCODE
WRAP  MOVE 'DISPATCH' TO XML-FILNAME
WRAP  MOVE 2048 TO X-REC-LNG
WRAP   ENTRY'DISPATCH' USING XML-FUNCODE,
       XML-RETCODE,XML-
       FILNAME,LIEFERPOSTEN
        DISPATCH-STATUS
```

Here the original read statement is converted to an entry statement so as to be able to call this procedure with a stream of data from outside. The type of wrapping differs between online, batch and sub programs. This wrapping of COBOL code has been described in previous papers [15].

The statements within the procedural blocks can be considered to be business dependent if they refer to a business object, otherwise they are implementation dependent. The intertwining of business and technical statements within the same procedural blocks is next to the naming of data the greatest obstacle to mapping the code model on to the business model. It also creates a problem for the extraction of business rules to enhance the business process model [16].

V. MODELLING THE BUSINESS PROCESS

The language used here to model the business process is a subject-oriented process model language referred to as S-BPM. S-BPM was developed by Albert Fleischmann as an alternate to the better known but less precise modeling language ARIS [17].

S-BPM is being used by several large German and Japanese user organizations to model their business processes. The main reason for them to choose this language is that it can be more easily mapped to the underlying code. That makes it possible to generate executable Java systems, something which would not be possible with ARIS. S-BPM has proven to be not only semantically sound, but also practically useful in reflecting existing IT systems since it allows existing software components to be built into its process description.

In recent years BPMN has emerged as an international standard for business process languages [18]. But there is also criticism of the BPMN 2.0 standard. One of the issues is that the standard document itself is incomprehensible and that the language is at some points inconsistent [19]. BPMN also defines many entities that are not always self-explaining, e.g. an “event based exclusive gateway”. Using BPMN to describe COBOL applications only replaces one complex artificial language with another one. S-BPM on the other hand has only five main symbol types and a close relation to natural language. S-BPM’s main entities are subjects, predicates (functions) and objects. There even exists a natural language export for S-BPM [20] which allows the business aspects of the legacy systems to be expressed in natural language.

As a language S-BPM fits better to the concepts of SOA. It allows the business analyst to model distributed, interacting processes built upon a set of common services [21]. The notion of a subject driving a chain of services is more in tune with the service concept. At the same time, S-BPM is closer to the semantics of a procedural programming language like COBOL and can be used as a hinge between existing code and BPMN. This and the fact that S-BPM has utilities for generating executable Java code are the main reasons for using it to model the business processes [22].

Subject-oriented models are built around subjects which are collections of related actions independent of the data those actions act upon. The objects to be processed by a subject can be assigned just in time, i.e. at run-time. A subject is a set of related actions to be performed on an arbitrary set of objects which may not be known when the subject is conceived. Thus, a subject is a container of related actions, just like modules in a procedural system.

S-BPM is based on a simple predicate logic similar to natural language [23]. Subjects act upon objects, but their relation is variable. The objects are defined independently of the subjects and can be joined to them in varying combinations, depending on the predicates. Predicates are verbs which connects the subjects to objects. The predicate “process” connects the subject ”accountant” with the object “invoice” in the statement “accountant processes invoices”. This is referred to in S-BPM as a subjective statement. Every subjective statement is composed of a subject, predicate and one or more objects, e.g.

“accountant processes invoices and accounts”.

The fixed entity is the subject - the accountant. The variable entities are the objects – invoices and accounts and the predicate – processes. This is an inversion of the principle of object-orientation where the fixed entity is the object. There one would say “invoices are processed by an accountant”.

In S-BPM a subject contains a sequence of actions which can be compared to methods in an object model. The difference is that in an object-oriented model, the methods are attached to objects. The methods of a class all refer to the same object. In S-BPM actions are attached to a subject and a subject can process any number of different objects. One function of a subject may be to set the amount due in an invoice and a third may check the customer credit rating. Functions are grouped by subject rather that by object since the subject is the carrier of functionality.

Finally, there are business rules. The actions of a subject are connected via transitions depicted as edges in a directed graph. These transitions can be assigned a business rule. Business rules can be conditional or non conditional. A conditional rule checks the state of one or more objects and returns a true or false answer. Just as there is a 1:*n* relationship between rules

and objects, there is also a 1:*n* relationship between rules and actions and therefore between rules and objects. The rule

> "if Customer.CreditRating = = gold &
> Article.Amount > Order.Amount
> let Article.Status == sold;

refers to the objects "Customer" and "Article". It is attached to the transition from the subject CustomerProcessing to the subject OrderProcessing.

Since actions can take place at different times and at different places, messages are required to connect and trigger subjects. It is interesting to note that there are no events in S-BPM and therefore no use cases just as there are no visible events or use cases in the source code. What starts a chain of actions is the receipt of a message. An event occurs when a message is received, but it is not part of the process description language, since it does not exist as such. For instance, an event "Order Processing" occurs when the subject "OrderProcessor" receives the message "CustomerOrder" and the event "Billing" takes place when the subject "Accountant" receives the message "CreateInvoices". The event itself is only an abstract notion for receiving a message. This set of dependencies leads to the following chain of entities:

Subject=>Message => Subject => Action => Rule => Objects.

VI. MAPPING THE CODE MODEL TO THE S-BPM MODEL

The key to linking the code model to the S-BPM model is the web service definition language - WSDL. The interfaces to the code are mapped to WSDL in order to be mapped further on to S-BPM. The mapping associations are as follows:

COBOL		**WSDL**	**S-BPM**
Program	=>	Service =>	Subject
Procedures	=>	Operations=>	Actions
Parameters	=>	Messages=>	Messages
Maps			
Input Files			
Structures	=>	ComplexTypes=>	Objects
Conditions	=>		Rules

The subjects are the programs, i.e. the run units, triggered by the operating system or by external messages. The actions in S-BPM are the operations in WSDL which are the entry points in the code. Normally a code module will have only one entry point, so that an action corresponds to a module. Should there be more entry points then there will be multiple operations for the same module. The messages in S-BPM match 1:1 to the messages in WSDL, but only 1:n to the messages in the legacy code. Modules not only receive parameters, they are also driven by incoming maps and files. Thus, a message could be a parameter list, but it could also be the next record of a file or the next panel in a dialog transaction. The data structure depicted below contains the parameters to a calendar subroutine in COBOL.

```
WRAP********** Generated Record ******************
WRAP   01  XM059-PARAMS.
               02  P1-DATUM.
                   03 P1-TT              PIC 99.
                   03 FILLER             PIC X.
                   03 P1-MM              PIC 99.
                   03 FILLER             PIC X.
SNEED              03 P1-CE              PIC 99.
                   03 FILLER             PIC X.
                   03 P1-JJ              PIC 99.
               02  P2.
                   03 P2-LANG-CODE   PIC 9.
               02  P3.
                   03 P3-DIRECTION   PIC X.
               02  P4.
                   03 P4-WEEKDAY     PIC X(10).
WRAP**** End of Generated Copy Member *************
```

The objects in S-BPM are the data structures included in the parameters, files and maps converted to messages. Database tables and direct access files are not depicted in S-BPM. The rules are those conditions within the procedural blocks which govern the execution of actions, sends or receives. An exit point, i.e. return statement is considered to be a send action. An entry point is considered to be a receive action.

There is, of course, a control level above the individual programs which regulates their execution sequence. On the main frame it is the job control language, on the Unix it is a script language. The job control invokes the underlying programs and turns over control data to them. In the terms of S-BPM a job control procedure is a super-ordinate subject. BPEL is the job control language of a service-oriented architecture. It is planned to include job control procedures in this reverse engineering process as well, but it would go beyond the scope of this paper to describe that here. This paper restricts itself to the generation of subordinate subjects at the bottom level of the process model hierarchy.

Two different intermediate language levels are produced by the reverse engineering process:

- XML documents
- WDSL interface definitions

Between these two a data description table is generated to convert the legacy type names to more meaningful WSDL type names.

A. The XML document

The first intermediate language level extracted from the code is a domain specific XML document. It depicts the source module from an internal point of view as a hierarchy of data and function entities. At the top level is the module corresponding to an operation in WSDL. The module contains data structures, interfaces and procedural blocks. The data structures and procedural blocks are not mapped to the

business process model since they are at a too low level, however if a user of the process model wants to know more about what is behind a particular action step he can open this XML document and study it. The structures in the XML document contain data attributes and nested structures. The procedural blocks may contain business rules, but they contain mostly code which is only required for the implementation and which is of little interest to business Therefore only business relevant procedures or paragraphs are depicted. The interfaces in the in the XML document can be either global or local. Local interfaces are interfaces within the program. Global interfaces are interfaces to the external world. They include reads, writes, sends, and receives. These interfaces are mapped to the messages of the operations in WSDL.

It is sufficient to use the WSDL interface if one only wants an external view of the legacy program, for instance to include it as a subject in the business process model. However, if one also wants to know how the program works, the XML document should be referred to. It describes what internal data is processed by what procedures under what conditions. The goal is to give the user enough information as to whether he wants to reuse this program or not.

```
<subject name = "XM059" type = "sub" lang = "COBOL">
 <objects>
  <complexType name="XM059-PARAMS">
   <sequence>
    <complexType name="XM059-DATUM">
     <sequence>
      <element name = "P1-TT" type = "xs:decimal"/>
      <element name = "P1-MM" type = "xs:decimal"/>
      <element name = "P1-CE" type = "xs:decimal"/>
      <element name = "P1-JJ" type = "xs:decimal"/>
     </sequence>
    </complexType>
    <element name = "P2-LANG-CODE" type = "xs:char"/>
    <element name = "P3-DIRECTION" type = "xs:char" />
    <element name = "P4-WEEKDAY" type = "xs:string"/>
   </sequence>
  </complexType>
 </objects>
 <actions>
  <action name = "CALC-WEEKDAY" type = "intern"/>
  <action name = "XM059-EXIT" type = "return">
   <interfaces>
    <interface name = "WEEKDAY" type = "return"/>
   </interfaces>
  </action>
 </actions>
</subject>
```

B. The Name Conversion Table

Between the XML document in which the names are still the labels taken from the original program and the WSDL interface is a name table in which the data and procedure labels are listed out with their types. The user is presented this table in the form of an Excel sheet so that he can go through and assign meaningful names if he wants. If not the original names remain. The name tables are retained to preserve the link between the business process model and the original source code [24]. (see Figure 2: Name Conversion Table)

Entity	Type	COBOL Name	WSDL Name
XM059	Struc	XM059-PARAMS	WeekdayRequest
XM059	Struc	P1-DATUM	CurrentDate
XM059	Deci	P1-TT	CurrentDay
XM059	Deci	P1-MM	CurrentMonth
XM059	Deci	P1-CE	CurrentCentury
XM059	Deci	P1-JJ	CurrentYear
XM059	Char	P2-LANG-CODE	Language
XM059	Char	P3-DIRECTION	Alignment
XM059	string	P4-WEEKDAY	Weekday

C. The WSDL interface

The second intermediate language level – the WSDL interface – is generated from the XML document and the name table. There is one WSDL interface for each individual application as defined by the user. The operations are taken from the program entry points. The input and output messages are derived from the global interfaces of the programs in the XML document. The data types are taken from the structures contained within the global interfaces. The business rules for invoking operations are inserted as comments into the WSDL source before the operation definitions. Thus, the WSDL interface definition contains all the information necessary to create the subjects in a S-BPM model – their messages, their operations, their objects and their rules. Further detailed information is to be found in the XML document defined above.

```
<types>
- <complexType name="XM059-PARAMS">
- <sequence>
- <complexType name="CurrentDate">
- <sequence>
<element name="CurrentDay" type="xs:decimal" />
<element name="CurrentMonth" type="xs:decimal" />
<element name="CurrentCentury" type="xs:decimal" />
<element name="CurrentYear" type="xs:decimal" />
</sequence>
</complexType>
<element name="Language" type="xs:decimal" />
<element name="Alignment" type="xs:char" />
<element name="Weekday" type="xs:string" />
</sequence>
</complexType>
</types>
- <message name="WeekDayRequest">
<part name="CurrentDate" type="xs:string" />
<part name="Language" type="xs:integer" />
<part name="Alignment" type="xs:char" />
</message>
- <message name="WeekdayResponse">
<part name="Weekday" type="xs:string" />
</message>
```

VII. CONVERTING WSDL TO S-BPM

The input to the S-BPM process modeling tool is the WSDL interface of each service, i.e. application. From this a business process is created and presented. Any WSDL file describes the service interface by a set of operations with parameters, return values and their data structures. This can well be interpreted as message-based communication. This interpretation maps very well to an S-BPM process model since communication is a central aspect in S-BPM as well. A S-BPM process is depicted by two views.

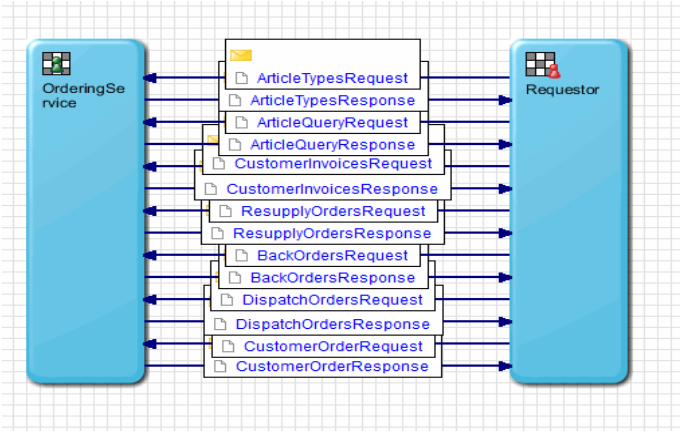
- An external view and
- An internal view.

The external view depicts all of the communication aspects (subjects and the messages the exchange) of a business process while the internal view of a subject shows the internal behavior of each subject.

A. *The external communication view*

The external view describes the exchange of messages between the subjects participating within an S-BPM process. On this level, the web service is represented as a subject. The other subject within this service layer process is the client or the requestor. It is of type "external" which means, it could be any subject in any other process [25]. So there are always two subjects in any of these service process, the requestor and the web service subject that will get the name of the service (the definitions/name attribute). Besides the two subject entities, there are also message entities specified by the message elements within the WSDL file. In S-BPM, a message consists of a sender, a receiver and a message type (the message content).

In the WSDL portType messages are specified as either inputs to or outputs from each provided operation. Hence, each input is interpreted as a request to the web service subject and therefore a message from the external requester subject to the web service subject. On the other hand, each output is interpreted as a message from the web service subject to the external requester subject. The following figure illustrates the communication view of the service process.

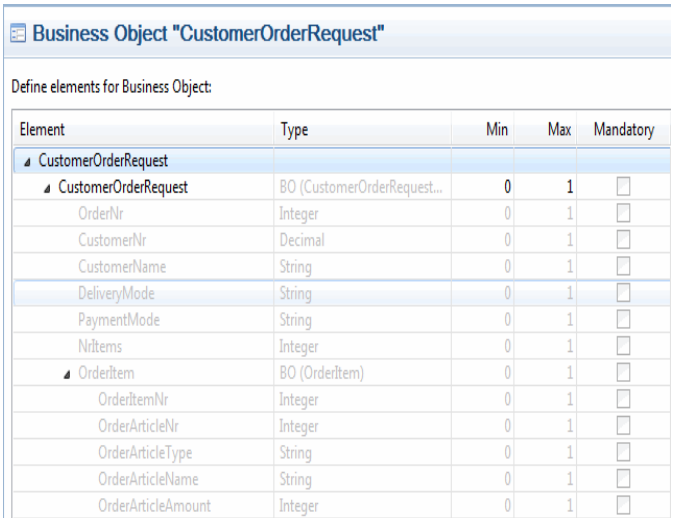


B. *The internal processing view*

As already stated, a subject's internal behavior consists of states and transitions. States can be of three different types: send message, receive message and internal function. In this process, only the web service subject has an internal behavior since the external subject (requestor) is just a place holder for any subject in another process that could be requesting the web service subject. The internal behavior would be the same for all imported web service subjects. There is a receive message state which is the start state and which indicates that the subject is waiting for messages. Depending on the incoming message, the corresponding transition is executed. Each WSDL operation refers to a specific input message. Each operation is here represented as a function state and the corresponding receive message transition leads to this function state. If the model would be executed, this function state would wrap the actual web service operation. If this function returns, the next transition is executed, leading into a send message state. This state sends the message back to the requesting subject. Once the message has been send, the next transition is executed that leads into the start state, waiting for new incoming messages. The internal view of the ordering service is displayed at the end of the paper (see Annex A).

C. *The data structures*

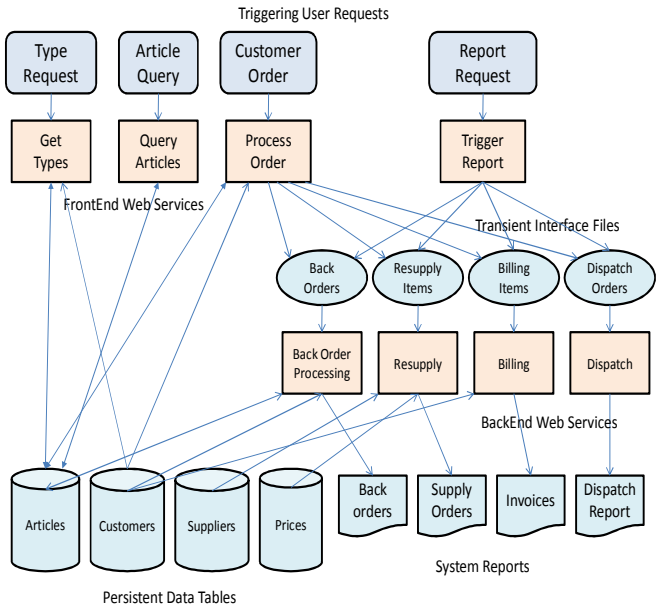
In SOA it is a common practice that the service provider defines the data types that he applies within the offered service. The user has to accept them as part of the interface and needs to provide the proper input data as well as he has to interpret the results for his demands [26]. The same applies to the presented solution. Actually the data import is technically the first step, since the generation of the other entities (like messages) requires these data items. As already mentioned, S-BPM also has objects as modeling entities. Business objects can be associated to messages and processed by internal functions.



These business objects reflect the data structures defined in the WSDL file. They are imported into the process group and made available to all process models within the group. If the service is about to be used, it is up to the technical process modeler to map existing data to the structures demanded by the service.

VIII. MODEL LINKING EXAMPLE

In this section the reverse engineering of an order entry application for processing customer orders is depicted. There were originally eight COBOL programs, four dialog programs and four batch programs (see Figure 3: The Order Entry Application).



A customer first signs in with his password number and name. The first dialog program confirms that the customer is known and returns all article types in the article data base. The second dialog program accepts an article type from the user and lists out all articles of that type with their prices. The third dialog program processes the customer order. It checks the customer credit and, for each item ordered, compares the amount ordered with the amount on stock. If the amount on stock is enough, the program reduces that amount by the amount ordered and generates a dispatch request and a billing item. If the reduced amount on stock falls below the minimum stock level, a resupply order item is created. If the amount on stock is not enough it creates a back order. Each of these dialog steps is taken to be a separation operation.In a separate process the batch programs are invoked to produce the dispatch orders, to produce the invoices, to produce the resupply orders and to process the back orders. Each of these batch programs is considered to be a separate operation.

The XML interface of the order entry service is very large so that only a fragment can be shown here. There are not only modules for each of the seven operations, but also modules to access the four database tables – articles, customers, suppliers and prices., There are in all 12 business objects plus 28 technical objects, plus 16 internal interfaces. There are more than 100 actions with more than 50 business rules. This XML document does not describe everything but it reflects the complexity of the task. For more information the interested viewer would have to look at the code itself. Here the data names are left as they were in the original code.

The name conversion table for this application contains more than 200 data names and some 60 procedure names. Only the parameter and the entry names were converted.

The WSDL interface of the order entry function depicts that function as a complex service with seven operations

- getArticleTypes
- processArticleQuery
- processCustomerOrder
- processDispatchOrders
- processResupplyOrders
- processCustomerInvoices
- processBackOrders

Each operation has a data structure as a request – the input message – and a data structure as a response – the output message. Depicted here is the structure of the customer order.

```
<complexTypename=" OrderItem" minOccurs="1" maxOccurs="10">
  <sequence>
    <element name="OrderItemNr" type="xs:integer" />
    <element name="OrderArticleNr" type="xs:integer" />
    <element name="OrderArticleType" type="xs:string" />
    <element name="OrderArticleName" type="xs:string" />
     <element name="OrderArticleAmount" type="xs:integer" />
  </sequence>
</complexType>
 <complexType name="CustomerOrderRequestType">
  <sequence>
    <element name="OrderNr" type="xs:integer" />
    <element name="CustomerNr" type="xs:double" />
    <element name="CustomerName" type="xs:string" />
    <element name="DeliveryMode" type="xs:string" />
    <element name="PaymentMode" type="xs:string" />
    <element name="NrItems" type="xs:integer" />
    <element name="Items" type="Orderitems" />
  </sequence>
 </complexType>
```

The requests and responses for the dialog operations are taken from the user interface descriptions to these three programs, i.e. from their maps. The requests for the four batch operations are taken from their respective input file records. . Since they are batch operations they have no response except for an artificially created return code.

```
<service = "OrderEntry">
-       <operation name="getArticleTypes">
            <input message="tns:ArticleTypesRequest" />
            <output message="tns:ArticleTypesResponse" />
        </operation>
-       <operation name="processArticleQuery">
            <input message="tns:ArticleQueryRequest" />
```

```
            <output message="tns:ArticleQueryResponse" />
        </operation>
-       <operation name="processCustomerOrder">
            <input message="tns:CustomerOrderRequest" />
            <output message="tns:CustomerOrderResponse" />
        </operation>
-       <operation name="processDispatchOrders">
            <input message="tns:DispatchOrdersRequest" />
            <output message="tns:DispatchOrdersResponse" />
        </operation>
-       <operation name="processResupplyOrders">
            <input message="tns:ResupplyOrdersRequest" />
            <output message="tns:ResupplyOrdersResponse" />
        </operation>
-       <operation name="processCustomerInvoices">
            <input message="tns:CustomerInvoicesRequest" />
            <output message="tns:CustomerInvoicesResponse" />
        </operation>
-       <operation name="processBackOrders">
            <input message="tns:BackOrdersRequest" />
            <output message="tns:BackOrdersResponse" />
        </operation>
    </service>
```

IX. LINKING CODE TO EXISTING BUSINESS PROCESS MODELS

There are two ways for linking models of existing code to a S-BPM process model. One way is to import the code interfaces as a set of related operations into a general repository and include them as internal functions within existing subjects. The other way is to interpret the code interfaces as separate service subjects. That is the approach presented by this paper. There are advantages and disadvantages to both approaches, but the second approach fits better to the concept of a service-oriented architecture and is also more intuitive.

The advantage of the first approach is that foreign functionality, i.e. web service operations, can be inserted directly as function states into the subject that uses it. If the service is represented as a separate subject encapsulating all the operations of the original web service, a send and receive construct is necessary to invoke that service subject from the process subject that wants to use it. Hence, the proposed interpretation requires more modeling entities, e.g. a subject representing the service, send and receive states in the invoking and the invoked subjects, as well as the messages passed between them. At first glance this seems to be a more complex solution accompanied by a bigger overhead. But there are good reasons for accepting this overhead.

Software systems are subject to continuous evolution. Process modeling and service-based architectures are intended to help in managing that evolution. As Josuttis states in his book on SOA practices - “SOA is a paradigm for the continuous evolution of business processes over time and space” [27]. In dealing with BPM and web services, one needs to consider the concepts of SOA and the requirements for flexibility. Integrating service operations directly into business subjects would bring less complexity, but it would constitute a point-to-point connection and violate the most important SOA concept, namely that of loose coupling.

Loose coupling is intended to reduce dependencies between components but it has a price, namely increased complexity. Loose coupling is the best way to achieve the non-functional requirements of flexibility, fault tolerance, and scalability. If a function state representing the web service operation would not be inside a separate service subject but instead be embedded in every subject using that service, changes to that service would have an impact on all of the subject processes which use that service. Should the service web address of an embedded service call be changed, one would have to search through all the subjects using that service, find the function states and change the address there. In the solution prescribed here one could define the web address within the properties of the service subject, reducing the impact domain to exactly one attribute. However, by having it as a separate entity in the process model the number of relationships between processes is increased, thus raising the complexity.

Another aspect to be considered is asynchrony. Within S-BPM, parallel execution is not possible within one subject, because subjects are understood as processing units. If the function state would be included into a real process subject, the process would have to wait for a response. With the proposed solution, the process could proceed and take care of the response later in a true asynchronous mode. In this way, the proposed concept fulfills most if not all of the requirements of loose coupling (distributed control, stand-alone messages, asynchronous communication and simple common data types). The service subject becomes an agent. Its internal behavior can easily be extended by exception paths and links to other services or service instances in case of failure.

With the proposed approach, it is possible to generate a service layer within a BPM suite that links that suite to the underlying code. The actual business process events are taking place above this layer. They are guiding the human users through their tasks, telling them what to do next. These higher level control subjects are equivalent to work flow control procedures written in a job control language (JCL). This way, a mixture of bottom-up and top-down approach to SOA design is supported. First, access subjects are created bottom-up to link the business model to the underlying code base, then process control subjects are defined top-down to depict the actual business work flows, but based on the lower level BPM service layer.

X. CONCLUSION AND FURTHER WORK

This paper has described the status of ongoing work in linking SOA business process models to existing legacy applications. The current starting point of this reverse engineering effort is the source code of the applications. Later the job control sources will be analyzed as well but that is a topic for future work. The source is analyzed to create a set of symbolic tables. From these tables XML documents are generated with the data

structures, the procedures, the conditions and the entry points contained within the code. These documents are then merged to create a WSDL interface definition for the application system as a whole with an operation for each entry point. This WSDL is the input to the process modeling tool which uses it to create a subject-oriented business process model with a subject for every application and an action step for every operation. The input and output messages of the WSDL become the interactions between subjects in the process model. In this way existing legacy IT systems are linked to the business process model to give users an overview of the existing application landscape.

The next step in this research project is to include an analysis of the control procedures which trigger the execution of the individual programs, that is, the level above the programming language. This is necessary to create super subjects, i.e. subjects which govern the behavior of the elementary subjects. It is also intended to export the business rules to the process model. For that the WSDL definition has to be extended to include conditions to the operations. It is planned to use the Schematron schema extension of the OMG for that. The ultimate goal is to provide business users with a semantically sound description of their current IT processes without having to model them manually. The manual modeling should be restricted to the conception of new business processes for which the underlying code has yet to be developed. It is also intended to combine this reverse engineering approach with the reengineering approach already established for converting legacy COBOL and PL/I programs to Java [28]. In this way the user will not only have a Java version of his legacy code but also a business process model which links the code to the view of the business user.

ACKNOWLEDGEMENT

The authors would like to thank their respective employers METASONIC AG and ANECON GmbH for supporting this research work.

REFERENCES

[1] Sneed, H.: "SOA Integration as an Alternative to Source Migration" Proc of SOAME Workshop, Timisoare, Sept. 2010.
[2] Sneed, H. "Migrating to Web Services–A research framework" Proc of SOAM Workshop, CSMR, Amsterdam, March 2007, p. 3.
[3] Sneed, H.: "Encapsulating Legacy Software for Reuse in Client/Server Sysyems" Proc. Of 3rd WCRE, IEEE Computer Society Press, Monterey, CA., Nov., 1996, p. 104
[4] 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
[5] Chikofsky, E.J./Cross, J.H.: Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, Vol. 23, No. 1, 1990, p.13
[6] Winter, A., Ziemann, J.: "Model-based Migration to Service-oriented Architectures", in Proc. of SOAM Workshop, CSMR-2007, Amsterdam, p. 107.
[7] Hasselbring, W., Conrad, S., Koschel, A.: Enterprise Application Integration, Elsevier Akademischer Verlag, Heidelberg, 2006
[8] Canfora, G, Cimitile, A., Munroe, M.: "Reverse-engineering and Reuse re-engineering", Journal of Software Maintenance, Vol. 6, No. 2, March, 1994.
[9] Tonella, P., Potrich, A.: Reverse Engineering of Object-oriented Code, Springer Verlag, New York, 2005.
[10] Sneed, H.: "Transforming procedural Program Structures to object-oriented Class Structures", Proc. of Int. Conference on Software Maintenance (ICSM2002), IEEE Computer Society Press, Montreal, Oct. 2002, p. 286.
[11] Marcus, A., Maletic, J.: "Supporting Program Comprehension using semantic and structural Information", Proc. Of Int. Conference on Software Engineering (ICSE2001), IEEE Computer Society Press, Toronto, June 2001, p. 103.
[12] Kuhn, A., Ducasse, S., Girba, T.: "Enriching Reverse Engineering with Semantic Clustering", Proc. Of Workshop on Reverse Engineering (WCRE2005), IEEE Computer Society Press, Pittsburgh, Nov. 2005, p. 133.
[13] Raja, U., Tretter, M.: "Classification of software patches – A text mining Approach", Journal of Software Maintenance and Evolution, Vol. 23, No. 2, March 2011, p. 69.
[14] Sneed, H.: "Integrating Legacy Software into s service-oriented Architecture", Proc. of European Conference on Maintenance and Reengineering, IEEE Computer Society Press, Bari, March, 2006, p.3.
[15] Sneed, H.: "Wrapping Legacy COBOL programs behind an XML Interface", Proc. of Working Conference on Reverse Eng., IEEE Computer Society Press, Stuttgart, Oct. 2001, p. 189.
[16] Sneed, H., Erdoes, K.: "Extracting Business Rules from Source Code", Proc. of 4th Workshop on Program Comprehension, IEEE Computer Society Press, Berlin, March, 1996, p. 240.
[17] Fleischmann, Schmidt, Stary, Obermeier, Boerger: Subject-Oriented Process Management, Hanser Verlag, Munich, 2011.
[18] OMG Business Process Model and Notation (BPMN), Version 1.2, Object Management group, London, 2009.
[19] 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.
[20] Sneed, S.: "Exporting Natural Language: Generating NL Sentences Out of S-BPM Process Models", Subject-Oriented Business Process Management, Communications in Computer and Information Science, 2011, Volume 138, Part II, Springer, 2011, p. 163-179.
[21] Fleischmann, A.: "Distributed Systems. Software Design and Implementation", Springer-Verlag, Berlin-Heidelberg, 1994.
[22] Sneed, S.: „Mapping possibilities of S-BPM and BPMN2", to be published at S-BPM ONE. Vienna, 2012.
[23] Metasonic, „Internal report on the Generation of Java Sources", Pfaffenhofen , July 2011.
[24] Chomsky, N.: Knowledge of Language, John Wiley & Sons, New York, 1986.
[25] Antoniol, G., Canfora, G., Casazza, G., deLucia, A., Merlo, E.: "Recovering traceability links between Code and Documentation", IEEE Trans. on S.E., Vol. 28, No. 10, Oct. 2002, p. 970.
[26] Yu, E., Mylopoulos, J.: "Understanding Why in Software Process Modelling and Design", Proc.of 16th International Conference on Software Eng.(ICSE1994), IEEE Computer Society Press, Sorrento, Italy, p. 159.
[27] Josuttis, N.: "SOA in der Praxis, System-Design für verteilte Geschäftsprozesse", dpunkt Verlag, Heidelberg, 2008.
[28] Sneed, H.: "Migrating from COBOL to Java – A Report from the Field", IEEE Proc. of 26th ICSM, Computer Society Press, Temesvar, Ro. p. 122.