Linking Business Process and Software System

Lerina Aversano, Marco di Brino, Paolo di Notte, Domenico Martino and Maria Tortorella
Department of Engineering, University of Sannio, P.zza Roma, Benevento, Italy
aversano@unisannio.it, tortorella@unisannio.it

Keywords: Business Process Modelling, Software modelling, Linking Process and software components.

Abstract: Enterprise necessitates to follow the rapid evolution of its business processes and rapidly adapt the existing software systems to its arising needs. A preliminary requirement is that the software subsystems are available and interoperable. A widely diffused solution is moving the adopted software solutions toward an evolving architecture, such as the one based on services. The objective of the research presented in this paper is to support the reuse of the existing software systems in a Service Oriented Architecture. The proposed solution is based on the idea that a Service Oriented Architecture can be obtained from a wide range of existing pieces of software. Such code components can be extracted from the existing software systems by identifying those ones supporting the business activities. Then, the paper proposes an approach for identifying the parts of software candidate to support a business process activity and it is based on the recovering of the links existing between the model of a business process and the supporting software systems. .

1 INTRODUCTION

The continuous changes of business requirements force enterprises to continually evolve the software systems they use for supporting the execution of their business processes. In this context, maintenance activities are required for adapting the software systems to the business process changes.

A business process consists of a set of activities performed by an enterprise to achieve a goal. Its specification includes the description of the activities and relative control and data flow. The software system supporting it is generally an application providing used during the execution of the business process activities. It is clear that a software component can be impacted, more or less significantly, from each business process change. The identification of the components impacted by the business change requirements is not always obvious to the maintenance workers. This is true especially when the change is expressed in terms of business activities with reference to the business context, or when the maintenance workers have not adequate information regarding the software system and its components with reference to such a kind context.

Therefore, it is very important an appropriate identification and comprehension of the relations existing between business process activities and software system components. Such a kind of comprehension provides a great help to the maintenance workers that are called to handle the change requests.

Considering the continuous change in the world of the information technology, there is an increasingly diffusion of Service Oriented Architecture, SOA. Its main strength is the easiness with which a service can be made available and used. This aspect suggests that the architecture of the old systems can be evolved towards a new system based on a service-oriented architecture.

Many approaches have been proposed in literature suggesting guidelines to identify services during the migration of legacy systems toward a service-based architecture (Khadka et al., 2013b) (Cetin et al., 2007). Nevertheless, in the authors knowledge, few papers propose the technical steps to be executed for achieving this goal. In (Balasubramaniam et al., 2008) an architecture-based and requirement-driven service-oriented reengineering method is discussed.

Aversano L., di Brino M., di Notte P., Martino D. and Tortorella M.
Linking Business Process and Software System.
DOI: 10.5220/0005887001920198
In *Proceedings of the Fifth International Symposium on Business Modeling and Software Design (BMSD 2015)*, pages 192-198
ISBN: 978-989-758-111-3

This method assume the availability of architectural and requirement information. The services are identified by performing the domain analysis and business function identification. Other approaches propose to evaluate services by performing either code pattern matching and graph transformation (Matos and Heckel, 2008), or feature location (Chen et al., 2005) or formal concept analysis (Chen et al., 2009). A detailed survey of the service identification methods is discussed in (Khadka et al., 2013a). In (Sneed, 2006), an automatic approach to evaluate candidate services is proposed. Candidate services are considered as groups of object-oriented classes evaluated in terms of development, maintenance and estimated replacement costs. In (Sneed et al., 2012), a tool is presented for supporting the reuse of existing software systems in a SOA environment by linking the description of existing COBOL programs to the overlying business processes.

This paper proposes a method for linking the process description with the components of a software system candidate to be reused in a service oriented architecture. The method exploits a formal description of a business process based on the BPEL language and calculates its textual similarity with the source code of the examined software system. The BPEL language has been chosen for the low effort needed to describe a business process by using it.

Section II describes the proposed approach and supporting tool; Section III describes the obtained experimental results; and concluding remarks and future works are discussed in the last Section.

2 APPROACH TO TRACEABILITY RECOVERY

The approach proposed aims at retrieving the traceability links between the business process activities and supporting software system components. It is based on the extraction of the identifiers of business process model and software components. It is composed of two processing phases:

- *Informationextraction* phase, regarding the extraction of semantic information from both business process and software system source code;
- *Traceabilityrecovery*, aiming at discovering the matching existing between the business information and software system components.

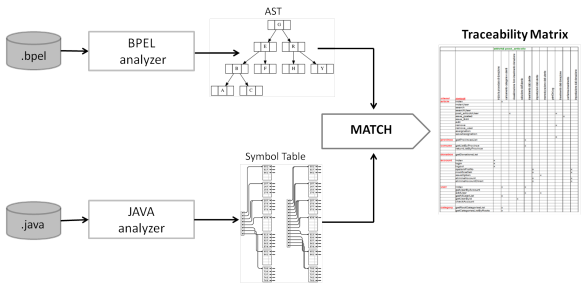

Figure 1: Overview of the approach fro traceability link recovery.

2.1 Information Extraction

The execution of this phase requires the implementation of a two parsers for analysing Java and BPEL files, and extracting all the needed information for performing the next traceability recovering. With this in mind, the JavaCC (Java Compiler Compiler) parser generator ($https : //javacc.java.net/$) was used. This tool reads a grammar specification and converts it into a Java program performing the top-down parser of a file written in the language based on the defined grammar. Then the regular expressions, context-free grammars and semantic rules were defined for describing both BPEL standard 2.0 and Standard Edition 7 for Java.

The implementation of the BPEL parser permits to construct the syntactic tree of the model description, which is the graph that allows expressing easily the process of derivation of a sentence using a grammar. The abstract syntax tree provides a structured view of the modelled business process, and excludes all the detailed information.

Figure 2 shows an example of a parse tree. It describes the business activities as brother nodes, while the son nodes indicate the artifacts needed for executing a business activity.

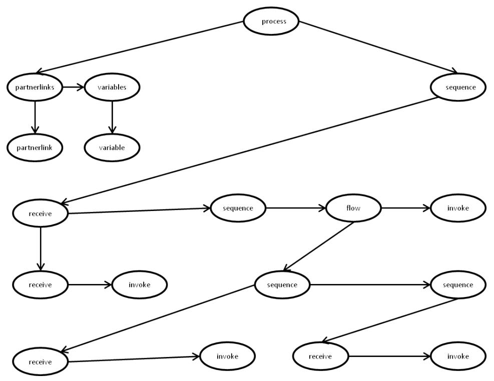

Figure 2: BPEL AST example.

After AST creation, further operations are planned for the correct insertion of comments. To identify the association between comments and code representing activities, it is necessary to make a visit to the tree in order to associate each comment node to the first brother node, which must not be a comment node. Once the association has been found. The analysis of the BPEL AST allows the identification of the identifiers for describing the business process.

The Java parser aims at constructing the symbol table of a supporting Java software systems, used to keep track of the source program constructs and, in particular, the semantics of identifiers, referred to packages, classes, methods, instance variables and local method variable declarations.

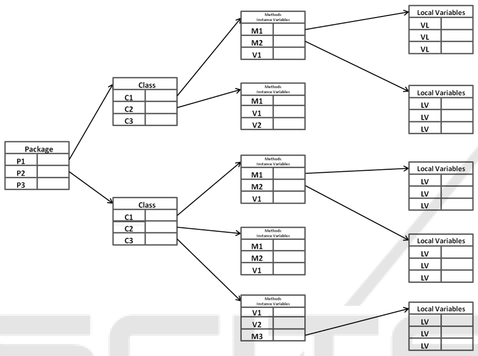

Figure 3: Java Symbol Table example.

The symbol table contains one record for each identifier, with some fields for its various attributes, such as its lexeme, type (e.g. identifier), identifier type that can be simple (integer, real, boolean, etc.), structured (vector or record) or a computational module, such as a function or procedure.

Figure 3 shows that the symbol table structure is hierarchical. On the first layer, there is a list of all packages declared in the project under consideration. Each record of this first layer contains a reference to another list, regarding the classes defined in the package in question. The set of all classes forms the second layer of the symbol table. Therefore, each class contains any references to objects declared in its interior, such as the methods and instance variables, and inner class. Each method can have another layer that represents the set of local variables it declares . Each inner class can be considered as a normal class, which may declare other methods, inner classes and instance variables. The procedure is iterated and accordingly the number of layers grows each time depending on the level of depth that will reach the analysis of the project concerned.

A preprocessing phase analyzes only the comments present within the various classes. Once it has been identified one, it is saved in a map, which will also allow saving the order of appearance of the various comments. After the preprocessing phase, the map is passed as an argument to the parser itself that analyses the comments for identifying additional semantic information coded in the code.

For each identifier, the symbol table records:

- *idName*: the name of the identifier to be saved;
- *kind*: the type of the identifier analyzed (class, method, package, etc ...);
- *scope*: the visibility of identifier (public, protected, private, etc ...);
- *args*: the method arguments;
- *typeRet*: the return type of the method or the type of a variable;
- *comments*: all comments associated with that identifier.

2.2 Traceability Recovery

After constructing the AST for BPEL and symbol table for Java, they are visited in a post-order manner for collecting the information necessary for the continuation of the analysis.

All the steps that follow are summarized in the chart drawn in Figure 4.

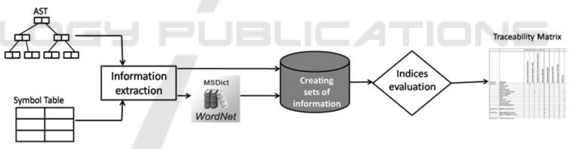

Figure 4: Information extraction phase.

The Information Extraction activity visits the AST BPEL and creates an array of BPEL activities, called *Activity*. Each *Activity* objects includes the BPEL file name, the task name and the set of terms related to the most important information; as an example, *name* and *operations* are kept for the *invoke* activities, while *portType* and *partnerLink* are considered wit reference to the *reply* and *receive* activities.

On the other side, the Java symbols table is entirely in order for identifying all the keys that correspond to the methods. Once one of it has been identified, a string set that contains the method name, any local variables name and inner classes that are contained within it, is created. Every single set of strings is in turn stored within a new map, which has as key a counter of the various set just created.

In accordance with the convention for identifiers nomenclature, when a term composed of two or more words is met, besides the full name, the individual terms are also included in the relevant collection of terms. Before being inserted in the collection, each term is normalized, i.e. all its characters are rendered tiny and all of the other symbols different from character and number are deleted. For example, if a method is called *GetCustomerName*(), the terms *GetCustomerName*, *get*, *customer* and *name* are included in the collection of terms. On the contrary, round brackets are non considered.

According to the above, different subsets of terms are created. For the *invoke* activity of the BPEL model, the following sets are created:

- a set including the terms contained in the argument called *operation*;
- a set including the terms contained in the argument called *name*;
- a complete set including the terms contained in the arguments called *operation*, *name*, *partnerLink*, *inputVariabile* and *outputVariable*.

The sets created for the *reply* and/or *receive* activities re the following:

- a set including the terms contained in the argument called *portType*;
- a set including the terms contained in the argument called *partnerLink*;
- a complete set that including the terms contained in the arguments called *portType* and *partnerLink*.

The terms of terms created for the Java methods are the following:

- a set including the terms of the considered method;
- a complete set including the names of the considered method, the names of its local variables and any inner classes, together with the single split words of the terms.

If no direct link is found between BPEL business activities and Java software methods, it is necessary to make a finer analysis. The first thing that is possible to do is the refinement of the terms, which requires their removal from the stopwords, or words that have no additional information content. In addition, for any term contained in the BPEL and Java full sets the set of synonyms are considered. The WordNet library, which is a lexical-semantic database for the English language, developed from Princeton University, is useful for performing this task.

For each term within the set of created words, a vector of synonyms is generated, which is added to the starting sets of terms.

2.2.1 Creating traceability matrix

After the creation of the sets of terms related to Java methods and BPEL activity, it is possible to proceed with the calculation of similarity between them, in order to properly fill the traceability matrix.

The coefficient used for the calculation of similarity is the Jaccard index. It is also known as the coefficient of Jaccard similarity, and it is a statistical index used to compare the similarity and diversity of sample sets. It is defined as the size of the intersection divided by the size of the union of the sets of samples:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

The value of this coefficient is defined in a range of values going from 0 to 1 (extremes included). In our case, $|A|$ represents the single set of terms obtained from the analysis of a Java method, while $|B|$ is the single set of terms obtained from the analysis of a BPEL activity. Therefore, we have n sets of type $|A|$ and m sets of type $|B|$, where n represents the total number of methods identified taken from the Java parser and m the total number of activities extracted from BPEL.

At this point, after calculated the coefficients, it is possible to generate the traceability matrix, which will be contained in an Excel file.

Generally, the traceablity matrix is composed as it follows :

- *row*: the $i-th$ row represents a method extrapolated from the Java parser, identified by a name but also by the package name and the name of its class;
- *column*: the $j-th$ column represents a basic BPEL activity extrapolated from the corresponding parser. It is identified by the BPEL file name containing the name of the activity and an argument called *name*;
- *intersect*: the i,j cell (i identifies the row and j the column) represents the value of Jaccard index. Simply, the value of this cell is calculated on the sets of terms previously obtained by Java method i and BPEL activity j.

2.2.2 Reporting matches

Whenever there is a correspondence between a Java method and a BPEL activity, the relative Jaccard index in the matrix is marked with a different color.

The most frequent case is that the analyzed Java method name is contained inside the parameter called *operation* (for *invoke* activity) or the parameter called *portType* (for *reply/receive* activities). For this reason, these first sets created based on information obtained from the BPEL activity are compared with each single set created on the basis of the analysis of the Java method name. A statistical study showed a real correspondence exists between a BPEL activity and a Java method if the similarity result has a value either greater of 0.85 or included between 0.33 and 0.55.

If the calculated index is not included in the indicated range, the analysis is done with all the sets created on the basis of the name contained inside the parameter called *name* (for *invoke* activity) or *parterLink* (for *reply/receive* activity). Even in this case, the Jaccard index is calculated between thesets of BPEL terms and the ones created with the Java methods; similarly, it exists a real correspondence between BPEL activity and Java method if the result has a value either greater of 0.85 or included between 0.33 and 0.55.

If the result is not in this range, the complete set is analyzed. The Jaccard index is calculated for all combinations of the complete set; afterwards, the greater index of the column is selected and the correspondence is marked if the index is great o equal to 0.33 (for *invoke* activity) or greater o equal to 0.55 (for *reply/receive* activity).

If in a column (associated with an activity) just one value different from 0 exists, it is marked a correspondence between the related Java method and BPEL activity, even if the value is not within any indicated range. This case is considered because that particular activity can be put in correspondence just with that Java method, even if the possibilities are low.

A prototype supporting tool has been implemented to process Java and BPEL sources code and produces a traceability matrix as output. Table 1 contains a sample output of the prototype. Each line represents a Java method of the analysed software system. The rows list: package name (*com.example*), Java class name (*TestProcess*) and method name (e.g. $getInfo()$). Each column, instead, represents a BPEL activity. It is possible to see: file name with extension .bpel (*process.bpel*), type of activity (e.g. *invoke*) and activity name (e.g. $getInfo$). The row-column intersection of this matrix contains the value of the Jaccard index, that is the numerical value of the correspondence that exists between BPEL activity and Java method.

In this example, the first analysis is done between the activity entitled $getInfo$ and all Java methods identified by the parser. It is possible to notice that between the value of the parameter called *name* of this activity and the Java method called $getInfo()$ there is a strong correspondence; in fact, their similarity in the traceability matrix includes the value 1, which is the greatest possible. This correspondence is indicated with color red within the Excel file.

Table 1: Example of matrix produced by prototype tool.

	process.bpel **INVOKE** **(”getInfo”)**	process.bpel **RECEIVE** **(”receiveIn”)**	process.bpel **INVOKE** **(”callHelp”)**
com.example **TestProcess** **getInfo()**	1	0	0,009
com.example **TestProcess** **setInput()**	0,1818	0,865	0
com.example **TestProcess** **update()**	0,3228	0,1243	0,076

The second analysis concerns activity *receiveIn*. Unlike the previous case, there is not an exact correspondence between this BPEL activity and any Java method. In any case, there is still a very high value (0,865) with the method called $setInput()$. This value will be compared from the set of Java method with one of all sets created for this type of activity, which is the set that contains the name of $portType$ argument, the set that contains the name of $partnerLink$ argument or the complete set with all information relating this activity. This correspondence will turn brown within the Excel file.

Finally, from the analysis of the last activity $callHelp$, it is possible to notice that the Jaccard values are very low, so the prototype will not highlight any possible match between this activity and any Java methods. Consequently, in the Excel file these values remain to their default color that is black.

3 RESULTS

The approach presented in the previous section has been validated in three case studies with the aim of assessing its effectiveness. Specifically three Java projects have been selected.

The first one is downloaded from the web; it deals with the management of a dealership. Originally, it was composed of 1066 Java files (code lines 124459) and 33 BPEL files but to facilitate the correctness verification of the results (operation done by hand) and the interpretation of the same results, only a five real interesting files have been selected.

The second and the third projects regard Java web project. The first one is written for private purpose; while the second one for managing a university exam. For these projects we asked to a third person to write the BPEL file modeling the business process starting from their knowledge without considering the source code. Also these last projects are small enough to permit a right manually verification.

The traceability matrix which contains the Jaccard indexes was generated For each projects. For every project we calculated:

- false positives: correspondences detected but not real;
- true positives: correspondences detected and real;
- false negatives: no correspondences found but actually present;
- true negatives: no correspondences found and not really present.

Table 2 contains a summary of the results obtained for the first case study. The low value of false negatives (just one) indicates that, when the correspondence exists, the proposed approach detect it correctly. The 15 occurrences of false positives are due to correspondences that do not exist: it is possible that the analyzed activity have a nomenclature similar to the one of a Java method, but there is no real correspondence between them.

Table 2: Experimental results for Dealership.

Case Study	False Positives	False Negatives	True Positives	True Negatives
Dealership	15	1	60	13769

Table 3: Precision, Recall, F-Measure for Dealership.

Precision	Recall	F-Measure
0.8	0.98	0.88

Table 4 shows a sinthesys of the results achieved for the second case study. In this case, a high number of false positives was obtained. The analysis of the correspondent Java source code indicates the use of meaningless names given to the various methods in the different classes. In particular, names are not relevant to the reference responsibilities (functionality).

Table 4: Experimental results for Groupon.

Case Study	False Positives	False Negatives	True Positives	True Negatives
Groupon	9	1	6	3413

Finally, Table 6 contains the results of the OnLine shop case study. It shows a discrete number of exact matches. Unlike the previous case, the cause of

Table 5: Precision, Recall, F-Measure for Groupon.

Precision	Recall	F-Measure
0.4	0.86	0.55

false positive was not associated with the inadequate nomenclature, but with the presence of some terms that subsequently brought with them a number of synonyms negatively influencing the results.

Table 6: Experimental results for Online shop.

Case Study	False Positives	False Negatives	True Positives	True Negatives
Online Shop	2	2	5	3981

Table 7: Precision, Recall, F-Measure for Online shop.

Precision	Recall	F-Measure
0.71	0.71	0.71

3.1 Observation

Additional tests have been performed for considering the comments in the BPEL and Java files.

For associating a single comment to the BPEL activity, further nodes have been added to the AST. To capture the association between comments and relative source code, a visit of the tree is performed with the aim of associating every comment node to the first brother node, which obviously must not be in turn a comment node. Once the association has been found, a new record containing the comment is inserted within of the TreeMap of the considered node.

The validation of this variation of the approach was executed by considering the same 3 projects previously used. Comparing the new Jaccard indexes with the previous ones, it was found a deterioration of the results. This is due to considerable increase of terms included in the various sets. Thus, because of the considerable decrease of common terms in proportion to the totals, many of the real correspondence existing between Java method and BPEL activities (i.e. true positives) are not found. Thi experience shows that it is not suggested to consider comments for the creation of the set of terms for identifying a correspondence between BPEL and Java terms.

4 CONCLUSIONS AND FUTURE WORK

The paper presented an approach aiming at facilitating the reuse of the existing software systems that support business processes. In particular, this facilitation is provided by the ability of detecting the correspondences existing between source code components and activities, or processes, modelled by using the BPEL language.

The method implementation entailed the use of two parsers. The information extracted by using the parsers have been expanded and refined for being used in the traceability link recovery. The evaluation and selection of such correspondences has been performed by using the statistical indexes and similarity measure defined in the paper. A first analysis also included the comments in the code but it was observed that their use leads to worse results.

The preliminary results obtained by the proposed approach are encouraging and represent a starting point, for the identification of parts of the code from an existing software system with the aim defining new services to be used in a service oriented architecture. The approach is just based on the nomenclature used for naming methods and activities and does not analyse in details of the analysed software system. The values of *precision*, *recall*, $f-measure$ indicated in Tables 3, 5, 7 show the potential of the proposed approach.

The future work can concern the refinement of the selection of the correspondences in the matrix (refining the values in the range used for the analysis of Jaccard indexes), expanding test cases and extends the analysis also to WSDL files .

REFERENCES

Balasubramaniam, S., Lewis, G. A., Morris, E. J., Simanta, S., and Smith, D. B. (2008). SMART: application of a method for migration of legacy systems to SOA environments. In *Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings*, pages 678–690.

Cetin, S., Altintas, N. I., Oguztüzün, H., Dogru, A. H., Tufekci, O., and Suloglu, S. (2007). A mashup-based strategy for migration to service-oriented computing. In *Proceedings of the IEEE International Conference on Pervasive Services, ICPS 2007, 15-20 July, 2007, Istanbul, Turkey*, pages 169–172.

Chen, F., Li, S., and Chu, W. C. (2005). Feature analysis for service-oriented reengineering. In *12th Asia-Pacific Software Engineering Conference (APSEC 2005), 15-17 December 2005, Taipei, Taiwan*, pages 201–208.

Chen, F., Zhang, Z., Li, J., Kang, J., and Yang, H. (2009). Service identification via ontology mapping. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2009, Seattle, Washington, USA, July 20-24, 2009. Volume 1*, pages 486–491.

Khadka, R., Saeidi, A., Idu, A., Hage, J., and Jansen, S. (2013a). Legacy to soa evolution: A systematic literature review. In *In A. D. Ionita, M. Litoiu, & G. Lewis (Eds.) Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*.

Khadka, R., Saeidi, A., Jansen, S., and Hage, J. (2013b). A structured legacy to SOA migration process and its evaluation in practice. In *IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, MESOCA 2013, Eindhoven, The Netherlands, September 23, 2013*, pages 2–11.

Matos, C. M. P. and Heckel, R. (2008). Migrating legacy systems to service-oriented architectures. *ECEASST*, 16.

Sneed, H. M. (2006). Integrating legacy software into a service oriented architecture. In *10th European Conference on Software Maintenance and Reengineering (CSMR 2006), 22-24 March 2006, Bari, Italy*, pages 3–14. IEEE Computer Society.

Sneed, H. M., Schedl, S., and Sneed, S. H. (2012). Linking legacy services to the business process model. In *6th IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, MESOCA 2012, Trento, Italy, September 24, 2012*, pages 17–26. IEEE.