Journal of Theoretical and Applied Information Technology
10th February 2013. Vol. 48 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AUTOMATIC GENERATION OF BPEL PROCESSES FROM NATURAL LANGUAGE REQUIREMENT

[1]DENG NA, [2]LI DESHENG

[1] Dr., School of Computer, Hubei University of Technology, China

[2] Dr., School of Science, Anhui Science and Technology University, China

E-mail: [1]iamdengna@gmail.com, [2]ldsyy2006@126.com

ABSTRACT

Software's final success depends seriously on its requirement description. Nowadays, natural language is still the main description language of software requirement documents. In order to minimize the comprehension differences between users and developers about requirement, if natural language described requirement could be automatically transformed to BPEL processes, users and developers would reach a consensus rapidly and BPEL's development would be accelerated. In this paper, we propose an automatic generation method from natural language requirement description to BPEL processes. Firstly, requirement description is restricted and formalized; then, for those sentences with the prefix [RECEIVE] and [INVOKE], the relevant web services highly semantically matched are found out from the set of WSDLs; finally, the whole corresponding BPEL process is assembled from bottom to up. We provide a prototype to indicate our method's validity.

Keywords: *Requirement Description, BPEL, Natural Language, Semantic, Web Services, WordNet*

1. INTRODUCTION

Requirement description is one important step in software engineering. It records the functional and non-functional requirement software must satisfy according users' demand. Software's final success depends seriously on its requirement description. Whether the requirement description is clear and complete or not, whether users and developers have reached a consensus or not, whether developers would act on the requirement description or not, all of these will influence the software's final success.

Due to different viewpoints, users and developers often have their own comprehension of the requirement. Users pay their attention on what function software can provide and which level its performance can reach; however, developers prefer to consider from the angle of technology. In most situations, users are not familiar with those professional terms and technical problems. If the comprehension difference is not solved, it will be a hidden trouble in the lifecycle of software development.

Natural language has two main disadvantages: ambiguity and inconsistency. However, nowadays most of the software requirement documents are still written by natural language. This ascribes to two reasons: one is because users and developers nearly don't have the capability of describing requirement formally, and the other is because natural language has abundant glossary and strong expression ability.

Web Services are self-contained and self-described modular applications which can be released, found and used in the Internet [1]. As the de facto standard, Business Process Execution Language (BPEL) [2] is widely used in the composition and orchestration of web services. It makes use of some structural activities and orchestrates web services into a complete and executable business flow. Since BPEL put some useful atomic services together to become a business process satisfying a specific functional requirement, in some extent, BPEL can be seen as a software application. In this paper, we only concerns about BPEL's functional requirement.

This paper aims to automatically transform BPEL's natural language described requirement to BPEL process. The main contributions of this paper are: 1. We propose a restriction and formalization method so that computer can understand the requirement. 2. We show how to find the most matching messages and atomic services with each requirement sentence, and give an algorithm to put atomic services together. 3. A prototype implemented by Java is provided, and the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

experiments on an actual multimedia conference system indicate our method's validity.

2. RELATED WORK

References [3, 4] work over the similarity computation of requirement description. [3] points out that computing requirement similarity is helpful for the reuse of software's design, source code and testing cases, and it gives a framework SimReq. A similarity analysis algorithm is provided in [4], and it contains four steps: word segmentation, stop words deletion, stemming and similarity computation. On the basis of these four steps, we will add synonyms extension and modify the algorithm of similarity computation to fit our situation. With regard to BPEL's automatic generation, [5] converts labeled finite state machine to executable BPEL and WSDL codes, and creates a tool named HUMSA. [6] makes use of model-driven approach to generate BPEL processes. [7-10] study on how to transform UML to BPEL. There are also many researches on sentences similarity [11, 12].

3. INTRODUCTION TO BPEL

BPEL is a XML-formatted business process execution language. It makes use of some kinds of BPEL elements, including basic activities (Receive, Reply, Invoke, Assign, Throw, Rethrow, Exit, Wait, Empty), structural activities (Sequence, If, While, RepeatUntil, ForEach, Pick, Flow, Scope) and some other assistant elements (elseif, else, catch, compensateHandler, faultHandlers, eventHandlers, terminationHandler, catchAll, onEvent, onAlarm, onMessage), to orchestrate web services to an executable business process.

4. RESTRICTION OF BPEL'S NATURAL LANGUAGE REQUIREMENT

Natural language uses un-structural sentences' combination to describe things, so the logic relationships between things are implied in sentences' semantic and it is difficult for computers to comprehend these logic relationships directly from the sentences. For example, a natural language requirement description of stop recording in our multimedia conference system is as follows:

After receiving the request of stopping recording, stop the recording. If the stopping is successful, then return a message of success; or else, return a message of error and inform the chairman for 3 times at the same time. If any mistakes happen in the whole procedure, then return a message of error.

From above we can see that the requirement description is step by step, and this is revealed by the repositions (after, if, then, or else, at the same time). However, these simple logic relationships are hard to understand for computers. To resolve this problem, we shift the work of identifying steps to humans.

Definition 1: Operation sentence

Operation sentence is the sentence representing some operation in BPEL's natural language requirement description, eg: receiving the request of stopping recording, stop the recording.

Definition 2: Judgment sentence

Judgment sentence is the sentence representing deciding to execute some branch according to some condition in BPEL's natural language requirement description, eg: If the recording is successful, or else.

Definition 3: Choice introductory sentence

Choice introductory sentence is the sentence introducing a group of choice branches in the restricted BPEL's natural language requirement description. We use "Choice" in this paper.

Definition 4: Concurrence introductory sentence

Concurrence introductory sentence is the sentence introducing a group of parallel branches in BPEL's natural language requirement description, eg: in parallel, at the same time, simultaneously. We use "in parallel" after restriction in this paper.

Definition 5: Repeat introductory sentence

Repeat introductory sentence represents some operations are executed repeatedly. eg: for three times. We use "repeat for n times" after restriction in this paper.

Definition 6: Assistant sentence

Assistant sentences are the sentences except operation sentences, including judgment sentence, choice introductory sentence, concurrence introductory sentence, repeat introductory sentence and others.

We restrict BPEL's natural language requirement description by the following rules:

Rule 1: Restricted BPEL's natural language requirement description is composed by the main part and some fault handler parts. Each part is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

composed by some steps, and each step is assigned a serial number.

Rule 2: Each operation sentence, judgment sentence, choice introductory sentence, concurrence introductory sentence, repeat introductory sentence is represented by a step.

Rule 3: The serial number of each branch under a choice introductory sentence is the direct sub-serial number of which of the choice introductory sentence.

Rule 4: The serial number of each branch under a concurrence introductory sentence is the direct sub-serial number of which of the concurrence introductory sentence.

Rule 5: The serial number of each branch under a repeat introductory sentence is the direct sub-serial number of which of the repeat introductory sentence.

Rule 6: Each fault handler part corresponds to some steps in the main part. If errors happen in some steps of the main part, they will be handled by fault handler part.

After these rules' restriction, the stop recording example becomes such a form:

The main part:

1 receive the request of stopping recording
2 stop the recording
3 choice
  3.1 if the stopping is successful
    3.1.1 return a message of success
  3.2 or else
    3.2.1 in parallel
      3.2.1.1 return a message of error
      3.2.1.2 repeat for 3 times
        3.2.1.2.1 inform the chairman

Fault handler part:

If any mistakes happen in the procedure of step 1 to step 3

1 return a message of error

We give the mapping between the sentence patterns after restriction and the requirement description after formalization, shown in Table 1:

5. FORMALIZATION OF BPEL'S NATURAL LANGUAGE REQUIREMENT

We give the mapping between the sentence patterns after restriction and the requirement description after formalization, shown as follows:

the sentence patterns after restriction	**the requirement description after formalization**
receive the request of X	[RECEIVE]X request
if X	[CONDITION]X
return X	[REPLY]X
or else	[ORELSE]
in parallel	[FLOW]
Choice	[CHOICE]
repeat for X times	[REPEAT]X
If any mistakes happen in the procedure of step X to step Y	[FAULTHANDLER]X, Y
X	[INVOKE]X

Definition 7: requirement description statement

After natural language described requirement is formalized, each step is defined as a requirement description statement.

Definition 8: requirement description prefix

After natural language described requirement is formalized, the part bracketed by square bracket is defined as requirement description prefix.

Definition 9: requirement description sentence

After natural language described requirement is formalized, the part not bracketed by square bracket is defined as requirement description sentence.

After formalization, the stop recording example becomes such a form:

The main part:

1 [RECEIVE]stopping recording request
2 [INVOKE]stop the recording
3 [CHOICE]
  3.1 [CONDITION]the stopping is successful
    3.1.1 [REPLY]a message of success
  3.2 [ORELSE]
    3.2.1 [FLOW]
      3.2.1.1 [REPLY]a message of error
      3.2.1.2 [REPEAT]3
        3.2.1.2.1 [INVOKE]inform the chairman

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Fault handler part:

[FAULTHANDLER]1, 3

1 [REPLY]a message of error

6. MATCHING AND LOOKUP OF MESSAGES AND ATOMIC SERVICES

The method we adopt is: finding all the most matching message names and operation names from WSDL (Web Services Description Language) documents with formalized requirement description, and using the relevant web services' information to replace the requirement sentences behind [RECEIVE], [REPLY], and [INVOKE].

We give an algorithm calculating the matching degree between requirement sentences described by natural language and message/operation names. Here, requirement sentences are the sentences with the prefix [RECEIVE] and [INVOKE].

Algorithm: SimilarityCalculation

Input: natural language described requirement sentence A; message/operation name B; synonyms set of WordNet C; stop words set D

Output: the matching degree φ

1. After restriction and formalization, A becomes $A^{'}$
2. If the prefix of $A^{'}$ is [RECEIVE] or [INVOKE]

2.1 $A^{''}$ is the requirement description sentence of $A^{'}$

2.2 The set of segmented words of $A^{''}$ is $wordset_A$

2.3 Move stop words from $wordset_A$. For any word w in $wordset_A$, if $w \in D$, then $wordset_A = wordset_A - \{w\}$.

2.4 Extent synonyms for $wordset_A$. For any word w in $wordset_A$, lookup synonyms(w) in C, and put all the synonyms of w into $wordset_A$, that is, $wordset_A = wordset_A + synonyms(w)$.

2.5 Do Stemming for $wordset_A$. For any word w in $wordset_A$, get the etyma of w using Porter[13], denoted as $w^{'}$, and replace w in $wordset_A$ with $w^{'}$, that is, $wordset_A = wordset_A - w + w^{'}$. In this paper, we represent $w^{'}$ as Porter(w).

3. The set of segmented words of message/operation names is $wordset_B$
4. Calculate the matching degree of $wordset_A$ and $wordset_B$ using Algorithm DicePlus.

Algorithm: DicePlus

Input: $wordset_A$ and $wordset_B$

Output: the matching degree φ of $wordset_A$ and $wordset_B$

1. int count=0;
2. or each word w in $wordset_B$

2.1 if (Porter(w) $\in wordset_A \| \exists w' \in$ synonyms(w) satisfying $w' \in wordset_A$)

2.1.1 count=count+1;

3. $\varphi = \dfrac{2 * count}{|wordset_A| + |wordset_B|}$

7. BPEL PROCESS'S ASSEMBLY

BPEL process is a XML-formatted document, and the requirement description after restriction reveals a hiberarchy structure, so we use the idea of from bottom to up in programming to assemble BPEL process. According to steps' serial numbers of formalized requirement sentences, we assemble BPEL's XML snatches level by level. In more detail, firstly, BPEL's XML snatches are generated corresponding to the lowest requirement description sentences; then, based on the prefix of the requirement description sentence in a higher leve, these XML snatches are assembled as a whole to be BPEL's snatch of this higher level's requirement description sentence; finally, the whole BPEL is assembled level by level in the same way.

During BPEL process's assembly, we consider the main part and the fault handler part separately. We firstly assemble the main part's BPEL snatch, and then assemble the corresponding snatch for each fault handler part, finally, we embed fault handler's snatch into main part's.

Algorithm: assembly of the main part

Input: formalized requirement description's main part M

Output: BPEL's XML snatch of the main part

1. Get the number of levels of the main part, denoted as maxLevel. Initialize a empty hashtable HT, the key of which is the serial number of steps, and the value of which is BPEL's XML snatch corresponding to the serial number.
2. Divide M into some heaps according to the level, and each heap has the same level. steps(*level*) denotes the set of all the steps with the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

level *level*. statement(*step*) denotes the requirement description sentence corresponding to step *step*. prefix(*statement*) denotes the prefix of *statement*.

3. for (int level=maxLevel; i>1;i--)

3.1. *Steps* = steps(*level*)

3.1.1. for *step* ∈ *Steps* , its step number is StepID, and its requirement description sentence denoted as *statement* = statement(*step*)

3.1.1.1 *prefix*=prefix(*statement*) , *sentence* = sentence(*statement*)

3.1.1.1.1 if prefix=[RECEIVE], handleReceive(*statement*)

3.1.1.1.2 if prefix=[INVOKE], handleInvoke(*statement*)

3.1.1.1.3 if prefix=[REPLY], handleReply(*statement*)

3.1.1.1.4 if prefix=[CONDITION], handleCondition(*stepID*, *statement*)

3.1.1.1.5 if prefix=[ORELSE], handleOrelse(*stepID*)

3.1.1.1.6 if prefix=[CHOICE], handleChoice(*stepID*)

3.1.1.1.7 if prefix=[REPEAT], handleRepeat(*stepID*, *statement*)

4. get all the step serial numbers stored in HT where the level of the key is 1. Arrange these serial numbers in order as $\{stepID_1, stepID_2 .....stepID_n\}$, then the BPEL process is:

<process>
  <sequence>
    HT.get($stepID_1$)
    HT.get($stepID_2$)
    ……
    HT.get($stepID_n$)
  </sequence>
</process>

Algorithm: handleReceive(*statement*)

1. *sentence* = sentence(*statement*)
2. In WSDL, find the most matching message name *inputMessageName* with *sentence* from all the attributes *name* of <portType>→<operation>→<input>. Suppose the *name* attributes of <operation>, <portType> and <output> are *operationName* , *portTypeName* and *outputMessageName* respectively.
3. BpelStr=<receive createInstance="yes" name=" *operationName* " operation=" *operationName* " partnerLink=" *operationName* " varaible=" *inputMessageName* ">
4. put (StepID, BpelStr) into HT. Save the data during this algorithm's execution.

Algorithm: handleInvoke(*statement*)

1. *sentence* = sentence(*statement*)
2. In WSDL, find the most matching operation name *operationName* with *sentence* from all the attributes *name* of <portType> → <operation>. Suppose the *name* attribute of <portType> is *portTypeName* , the message attributes of <operation>→<input> and <operation>→<output> are *inputMessageName* and *outputMessageName* respectively.
3. BpelStr=<invoke name=" *operationName* " operation=" *operationName* " partnerLink=" *operationName* " inputVariaible=" *inputMessageName* " outputVariaible=" *outputMessageName* "/>
4. put (StepID, BpelStr) into HT.

Algorithm: handleReply(*statement*)

1. get the data stored during the execution of algorithm handleReceive(*statement*)
2. BpelStr=<reply name=" *operationName* " operation=" *operationName* " partnerLink=" *operationName* " varaible=" *outputMessageName* ">
3. put (StepID, BpelStr) into HT.

Algorithm: handleCondition(*stepID*, *statement*)

1. *sentence* = sentence(*statement*)
2. get the direct sub-serial number of *stepID* , and arrange them in order as $\{stepID_1, stepID_2 .....stepID_n\}$. BpelStr is set as:

<condition> *sentence* </condition>
<sequence>
  HT.get($stepID_1$)
  HT.get($stepID_2$)
  ……

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HT.get($stepID_n$)
</sequence>
3. put (StepID, BpelStr) into HT.

Algorithm: handleOrelse(*stepID*)
1. get the direct sub-serial number of *stepID* , and arrange them in order as $\{stepID_1, stepID_2 ..... stepID_n\}$. BpelStr is set as:
<else>
<sequence>
HT.get($stepID_1$)
HT.get($stepID_2$)
……
HT.get($stepID_n$)
</sequence>
</else>
2. put (StepID, BpelStr) into HT.

Algorithm: handleChoice(*stepID*)
1. get the direct sub-serial number of *stepID* , and arrange them in order as $\{stepID_1, stepID_2 ..... stepID_n\}$. BpelStr is set as:
<if>
HT.get($stepID_1$)
HT.get($stepID_2$)
……
HT.get($stepID_n$)
</if>
2. put (StepID, BpelStr) into HT.

Algorithm: handleRepeat(*stepID*, *statement*)
1. *sentence* = sentence(*statement*) , *sentence* represents the times repeated, denoted as TIMES.
2. get the direct sub-serial number of *stepID* , and arrange them in order as $\{stepID_1, stepID_2 ..... stepID_n\}$.
3. BpelStr is set as:
<variable name="repeat_time" type="xsd:int"/>
<sequence>
<assign>
<copy>
<from>
<literal>0</literal>
</from>
<to variable="repeat_time"/>
</copy>
</assign>
<repeatUtil>
<condition>$repeat_time&ltTIMES</condition>
<sequence>
<sequence>
HT.get($stepID_1$)
HT.get($stepID_2$)
……
HT.get($stepID_n$)
</sequence>
<assign>
<copy>
<from>$repeat_time+1</from>
<to variable="repeat_time"/>
</copy>
</assign>
</sequence>
</repeatUtil>
</sequence>

The assembly of fault handler part is similar to which of the main part. The difference only exists in step 4 in algorithm "assembly of the main part". The BPEL's snatch of fault handler part is as follows:

<faultHandler>
<catchAll>
<sequence>
HT.get($stepID_1$)
HT.get($stepID_2$)
……
HT.get($stepID_n$)
</sequence>
</catchAll>
</faultHandler>

Here, $stepID_1$, $stepID_2$ …… $stepID_n$ represent the serial numbers with level 1 of the fault handler requirement description after formalization.

8. IMPLEMENT AND ANALYSIS OF THE PROTOTYPE

We implement a prototype using Java. It can automatically transform restricted requirement description to BPEL process. In our experiments,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

we use the atomic web services of a multimedia conference system.

The human–computer interaction interface of our prototype has four parts:

(1)Inputbox of natural language described requirement

(2)Inputbox of threshold of the matching degree

(3)Outputbox of formalized requirement description

(4)Outputbox of generated BPEL process

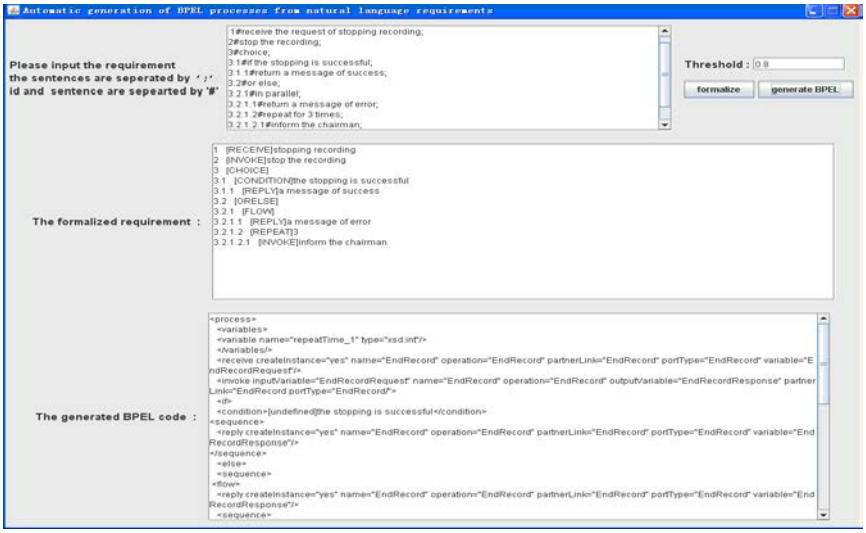

Figure 1: The Prototype Automatically Transforming Requirement Description To BPEL Process

As shown in the Figure 1, through the semantic matching lookup based on WordNet, service "EndRecord" can be found corresponding to "stop the recording" in user's requirement. Thus this indicates the semantic matching method's validity.

9. CONCLUSION

Requirement description has greate influence on software's success. Natural language is still the mainstream description language of requirement document currently. The automatic generation of BPEL process from requirement can help users and developers reach a consensus in a short time, and can also quicken BPEL's development. Our paper proposes such a generation method, which is direct and creative. We provide a prototype to indicate our method's validity. The future research will focus on BPEL's semantic analysis.

ACKNOWLEDGMENTS:

This work was supported by the Natural Science Foundation of Educational Government of Anhui Province of China (No. KJ2013B073), the Science and Technology Plan Project of Chuzhou City (No.201236), and the Talent Introduction Special Fund of Anhui Science and Technology University (No.ZRC2011304).

REFRENCES:

[1] Tidwell D, "Web services-The Web's Next Revolution", 2001, http://www.ibm.com/developerWorks/

[2] Oasis, "Business Process Execution Language(bpel) v2.0", http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[3] Muhammad Ilyas and Josef Küng, "A Comparative Analysis of Similarity Measurement Techniques through SimReq Framework", *Proceedings of the 7th International Conference on Frontiers of Information Technology*, Abbotabad, Pakistan, 2009, pp.1-6.

[4] Johan Natt Och Dag, Björn Regnell, Pär Carlshamre, et al, "Evaluating Automated Support for Requirements Similarity Analysis in Market-driven Development", *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality*, Interlaken, Switzerland, 2001.

[5] Mohanty H, Chenthati D, Vaddi S, et al, "Automatic Generation of BPEL and WSDL from FSM Models of Web Services", *Proceedings of 2006 International Conference*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

on Advanced Computing and Communications, Surathkal, 2006, pp.440-444.

[6] Li Zhang and Wei Jiang, "Transforming Business Requirements into BPEL: A MDA-*International Workshop on Semantic Computing and Systems*, Huangshan, China, 2008, pp. 61-66.

[7] Anisha Vemulapalli and Nary Subramanian. "Evaluating Consistency between BPEL Specifications and Functional Requirements of Complex Computing Systems using the NFR Approach", *Proceeding of 2010 IEEE International Systems Conference*, San Diego, 2010, pp.153-158.

[8] Anisha Vemulapalli and Nary Subramanian., "Transforming Functional Requirements from UML into BPEL to Efficiently Develop SOA-based Systems", *Proceeding of 2009 On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, Vilamoura, Portugal, 2009, pp. 337-346.

[9] Keith Mantell, "From UML to BPEL: Model Driven Architecture in a Web Services World", http://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/#author1

[10]Rainer Anzböck and Schahram Dustdar, "Semi-automatic Generation of Web Services and BPEL Processes - a Model-driven Approach", *Proceedings of the 3rd International Conference on Business Process Management*, Nancy, France, 2006, pp.64-79.

[11]Jin Feng, Yiming Zhou and Trevor Martin., "Sentence Similarity based on Relevance", *Proceedings of IPMU'08*, Malaga, Spain, 2008, pp.832-839.

[12]Chukfong Ho, Masrah Azrifah, Azmi Murad, et al, "Word Sense Disambiguation-based Sentence Similarity", *Proceedings of Coling 2010*: Poster Volume, 2010, pp. 418–426.

[13]Porter M F, "An Algorithm for Suffix Stripping", *Program*, Vol.14, No.3, 1980, pp.130-137.

based Approach to Web Application Development", *Proceedings of 2008 IEEE*