ValySeC

Sun, Chang-ai; Xue, Tieheng; Aiello, Marco

Published in:
IEEE Asia-Pacific Services Computing Conference

ValySeC: a Variability Analysis Tool for Service Compositions using VxBPEL

Chang-ai Sun[1], Tieheng Xue[2], Marco Aiello[3]

[1,2] *School of Information Engineering*
University of Science and Technology Beijing,
100083, Beijing, P.R. China
casun@ustb.edu.cn, xuetieheng@gmail.com

[3]*Johann Bernoulli Institute*
University of Groningen
9747 AG, Groningen, The Netherlands
aiellom@cs.rug.nl

Abstract

Nowadays applications are increasingly developed based on remote Web services and service composition has become a powerful novel development paradigm. Due to the fact that such applications in the context of Internet are deployed and executed in an open and dynamic environment, adaptability is one of the crucial requirements for developing such applications. In our previous work, we presented a variability-based approach to address the adaptability issue of service compositions, and developed the VxBPEL, an extension of BPEL with respect to variability.

This paper presents a variability analysis tool for variability-based adaptive service compositions called ValySeC. ValySeC extracts the variation from service compositions specified using VxBPEL and provides a variation view. With the view, the designer can better understand variation points and the possible instances, and efficiently maintain the variation within service compositions. The paper presents the design and implementation of ValySeC using a case study to handle service compositions with variations to illustrate the main concepts.

1. Introduction

Nowadays, applications are increasingly developed based on Web services and service compositions have evolved as a novel development paradigm [14]. In the traditional software development, the applications are often constructed based on the exiting components which are usually retrieved from a local library. The scenario in the context of service compositions varies greatly. Web services are loosely coupled elements and dynamically orchestrated to fulfill a business goal.

Let us consider a travel agency service. It may compose in a travel package that exposes to the external world by composing, for instance, flight and accommodation services, which are provided by third party service providers. Web services themselves are deployed and executed in an open and dynamic environment, availability of the services is an issue in itself. This in turn requires that the composition should be able to select another one, for example, when a flight service becomes unavailable. Furthermore, the customers may come from different countries and have different purposes, thus their requirements vary significantly. When a traveler is arranging a personal trip to China, she would prefer the cheaper flight and accommodation. However if the trip is for business, the things change. Service compositions executing the business processes must be flexible enough to deal with dynamic requirements and deliver differentiated services. The Business Process Execution Language (BPEL) [5] is an executable service composition language. The standard BPEL is not sufficient for constructing the above-mentioned business process because it only supports the static or fixed Web service compositions.

VxBPEL [9] is an extension to BPEL that we developed to deal with adaptation in Web service compositions from the perspective of variability management, which is originally from the area of the software product lines [11]. VxBPEL provides the constructs for the variability concepts in the language level, and treats the changes as first-class entities, which are currently missing in most related approaches, particularly those focusing on the implementation level.

In this paper, we present a variability analysis tool *ValySeC* that extracts the variation of Web service compositions specified with VxBPEL and provides a variation configuration view. This is particularly useful when service compositions contain complex variation configurations. With the tool, the designer can better understand variation of service compositions, and efficiently maintain the variation. This tool is a followup of previous work reported in [9, 19, 20] and part of a design and analysis tool suite for variability-based adaptive service compositions that we are developing.

The rest of the paper is organized as follows. Section 2 introduces the underlying concepts and techniques. Section 3 presents the variability analysis tool *ValySeC* which is based on the COVAMOF framework and the VxBPEL. Section 4 reports a case study with *ValySeC*. Section 5 discusses related work and Section 6 concludes the paper.

2. Background

We first present the concepts and techniques used by *ValySeC*, and then introduce the variability management platform that we have developed for variability-based adaptive service compositions. Variability is the ability of a software system or artifact to be extended, changed, customized, or configured for use in a specific context [17]. Variability management includes the design, use, and maintenance of variability [3].

2.1 The COVAMOF framework

Variability management is an important reuse issue in product families [11]. Many variability modeling approaches have been reported [16, 2]. However, they are not adequate to handle variability issues relevant for industrial purposes [18]. This observation resulted in the creation of the COVAMOF framework which has been tested and evaluated positively in industrial settings [6, 16, 17]. The COVAMOF framework offers modeling facilities to model variation points and dependencies uniformly over multiple layers of abstraction.

The COVAMOF-VS (http://www.covamof.com/vs) is a tool suite developed for the COVAMOF framework. The tool suite is an add-in for Microsoft Visual Studio .NET, and can be used to create variability models of a software product family, and these models can then be used for the derivation of individual products. All COVAMOF models conform to the COVAMOF meta model presented in Fig. 1.

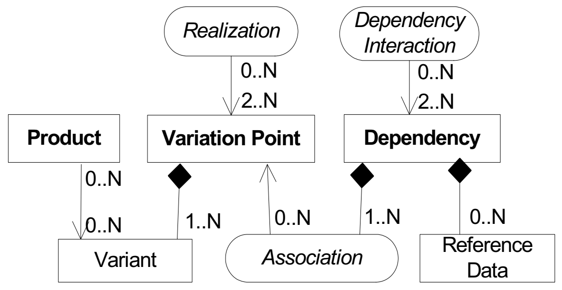


Fig.1. The COVAMOF meta model

The variability concepts within the COVAMOF meta model are

- *Variation point* and *variant*: Variation points represent a location at which a choice is provided. A variation point has a number of properties, such as variation type, abstraction layer, binding time, and rationale. Variants represent the options available at a variation point. Variants have an effectuating actions property, which specifies which effectuating actions should be executed when the variant is selected.
- *Realization*: Variation points can exist at different levels of abstraction. Realization relations specify rules that determine which variants at lower layers of abstraction should be selected, in order to realize the choice at variation points in higher layers.
- *Dependency*: A dependency represents a system property and specifies how the binding of variation points influences the value of that property, i.e., how the selection of certain variants influences the value of that property. Dependencies can have many variation points from different layers of abstraction associated with it and bridge multiple artifacts.
- *Association*: For each variation point associated to a dependency, an association entity is part of the dependency. Associations refer to variation points that affect the value of the systemproperty. Each association defines the relation with one variation point.
- *Reference data*: Besides associations, dependencies also contain so-called reference data elements. These entities contain information on the value of the system property acquired through testing. They consist of a set of variation point bindings, and the corresponding value of the system property.

2.2 VxBPEL

BPEL4WS [5] defines Web service orchestrations in terms of workflows by providing the core constructs, such as *partner, partner links*, *message*, *activities* and some *handlers*. *Activities* can be classified into *simple* activities such as *invoke*, *assign*, *receive* and *reply*, and *structured* activities such as *scope*, *sequence*, *flow* and *switch*.

VxBPEL [9] is an extension to the BPEL4WS that allows for run-time variability in service-based systems. In order to introduce variability into service compositions, VxBPEL extends BPEL with the constructs for defining and managing the variability. During the development of constructs for variants, variation points, and their associations, VxBPEL employs the COVAMOF framework and adapts it to the context of Web services due to its outstanding features, including treating variation points and dependencies as first-class citizens, tool support and its validation in industry. To indicate that a part of BPEL processes may be variable, we enclose it by new VxBPEL elements.

Fig. 2 and Fig. 3 illustrate the syntax of the variant element and the variation point element, respectively. The prefix *vxbpel* indicates the namespace of VxBPEL elements, which are not included in the standard BPEL namespace. To indicate the association between variation point and variants, the *vxbpel:Variants* tag which follows the variation point is used to enclose these variants. For example, the *CA* variant and the *LH* variant are associated with the *selecting an airline service* variation point.

```
<vxbpel:Variant name= “CA”>
  <vxbpel:VPBpelCode>
    <invoke …>
  </vxbpel:VPBpelCode >
</vxbpel:Variant>
```

Fig. 2. The VxBPEL *Variant* construct

```
<vxbpel:VariationPoint
        name= “selecting an airline service”>
     <vxbpel:Variants>
       <vxbpel:Variant name= “CA”>
       <vxbpel:VPBpelCode>
          <invoke …>
          </vxbpel:VPBpelCode >
       </vxbpe:Variant>
       <vxbpel:Variant name= “LH”>
       <vxbpel:VPBpelCode>
          <invoke …>
       </vxbpel:VPBpelCode >
       </vxbpel:Variant>
     </vxbpel:Variants>
</vxbpel:VariationPoint>
```

Fig. 3. The VxBPEL *Variation Point* construct

In order to manage variability constructs at runtime, we extended the BPEL engine ActiveBPEL [1]. Fig.4 illustrates service compositions using VxBPEL and their runtime platform. The specification of service compositions consists of native BPEL elements and VxBPEL elements. At run-time, the original *BPEL engine*, i.e. ActiveBPEL, is wrapped by the *VxBPEL interpreter* which is responsible for the interpretation of the VxBPEL elements. The *variability management* analyzes the relationship among VxBPEL elements and provides a variability configuration view which is a great aid for comprehending variation of service compositions. Thus, VxBPEL is not just an idea, instead a practical approach to constructing service compositions with variability management.

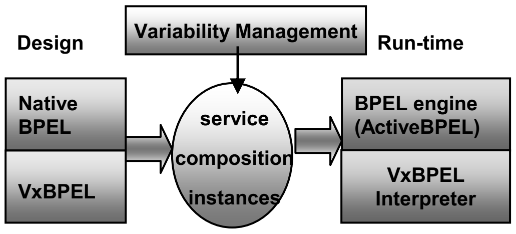


Fig. 4. Service compositions with variability and their run-time support

2.3 The variability management platform for adaptive service compositions

Developing the VxBPEL is the first step towards application of the COVAMOF framework to Web service-based systems. To leverage the potential of the COVAMOF framework for variability management in Web service-based systems, we further developed a framework and related tool suite for modeling and managing the variability of Web service-based systems for design and run-time, respectively [20].

In the current treatment, the variability management platform, as illustrated in Fig. 5, employs the COVAMOF-VS tool suite to maintain variation information of service compositions. In the different abstraction layer, the corresponding model providers are used to extract the variability information from the software artifacts and create the variability model which follows the meta model as illustrated in Fig.1. The COVAMOF-VS then provides the variation point view and the dependency view based on the variation information collected by the model provider. In this sense, the COVAMOF-VS tool suite is only responsible for visualization of variation information.

Although the COVAMOF-VS tool suite provides a generic and powerful infrastructure to visualize and manipulate variation information at the different

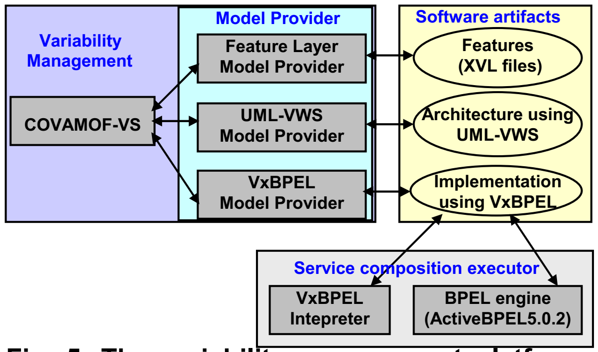


Fig. 5. The variability management platform for service compositions

abstraction layer. However, we feel that variability-based adaptive service compositions call for a new variability management platform because of the following observations.

(1) *Different motivation and settings*. The COVAMOF-VS tool suite is designed to support reuse within software product families, and therefore does not yet focus on adaptability in the context of Service Oriented Architecture.

(2) *Improvement of functionality is impossible*. Currently, source code and documentation of the COVAMOF-VS tool suite are not available.

(3) *Seamless integration with VxBPEL engine and service composition designer is difficulty*. The current version of the COVAMOF-VS tool suite is written in C# and implemented as a DLL library, while the VxBPEL engine and intended service composition designer is implemented in Java.

3. The variability analysis tool *ValySeC*

Based on the concepts of the COVAMOF framework and the VxBPEL, we are developing a variability-based adaptive service composition platform. The platform supports the analysis, design, execution, and maintenance of variation of services compositions using the VxBPEL. *ValySeC* is part of the platform and responsible for the variability analysis and maintenance. We discuss next the design and implementation of *ValySeC*.

3.1 Design principle of *ValySeC*

As mentioned before, when the service composition implements a complex business process and may involve a large number of activities and variations, it then becomes a difficult task to understand and maintain variation configurations. Hence, the primary goal of *ValySeC* is to provide a visual analysis of variations in service compositions and maintain their consistency when the changes happen.

The architecture of *ValySeC* is illustrated in Fig. 6. It employs the Model View Controller (MVC) structure. *ValySeC* reads in VxBPEL specifications and extracts the variability data to form the variability model. The extraction is left for a *parser* which traverses the VxBPEL specifications and collects the variability-related data. These collected data are then stored in the form of the variability meta model as illustrated in Fig. 1. The *UI* is responsible for visualization of variability information once a request from user is received. Actually, *ValySeC* is a kind of reverse analysis tools and thus follows the principle of general reverse engineering [21].

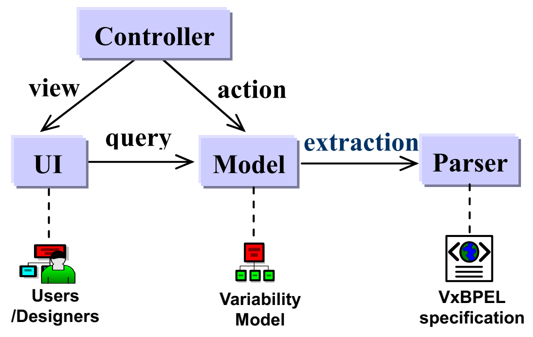


Fig. 6. The architecture of *ValySeC*

In order to aid the designers to better understand and maintain the variation, *ValySeC* provides the following features.

- Extraction of variability-related elements, such as Variation Point, Variant, Rationale, VPChoice, and ConfigurableVariationPoint.
- Reasoning the relationships and dependencies among variability elements.
- Visualization of the variation configuration.
- Query on the variability elements, relationships, and configuration.

Note that the VxBPEL supports the incorporation of variability management into service compositions in different ways. One is to *separate* the variability constructs from the main BPEL process. The other is to *inline* variability constructs in the main BPEL process. In terms of variation design, we recommend that the *inline* way should be employed because the *separate* way splits variability management into multiple files and causes a large number of references to the original BPEL process. However, the inline way scatters variation definitions and configurations among the whole composition specification and thus this poses the difficulty to the extraction and reasoning of variation. To increase adaptability and scalability of *ValySeC*, an adapter is introduced to extract variability data for the different ways of variation representations.

3.2 Implementation of *ValySeC*

We have implemented *ValySeC* in Java and based on JDK 1.6.0. During the implementation, we employ XML parser technologies (i.e. XML DOM interfaces) to extract the various variability-related data which are stored in the Document Class. Based on the collected data, we can further identify the type of variation representations according to the type of the root node corresponding to the service compositions. The type of variation representation belongs to one of the *Separate*, *Inline*, *Configurable*, *and Inline Configurable*.

The relationships among variability elements are decided by their hierarchy associated with the

VxBPEL specification. The hierarchy is easy to obtain from the XML file because these elements are defined following the XML schema and the variability meta model. For example, variants A and B are associated with variation point VP1, then the definitions of A and B must be included as elements of the definition of VP1. Similarly, all detailed information about variants and variation points are defined as their properties or elements.

Once the hierarchy of elements is decided, next is to visualize the variation configuration and provide the interactions to users. We employ a tree view for this task because the tree is a natural choice to represent the hierarchical configuration. Query on the tree view is supported through showing the detailed information about the selected variability elements.

Fig. 7 shows a snapshot of *ValySeC* when the variability representation is the *Inline* type indicated by the top title. The *Inline* type means that the variation elements are defined together the main business process. Note that the circle, triangle and rectangle are used to represent the variation point, variant and collection of variants, respectively. In the service composition illustrated in Fig. 7, the variation point has two variants. To query information of some variation elements, one just needs to select the element in the variation configuration tree view and then click the right mouse button. The separate window within the main window shows the information about the second variant.

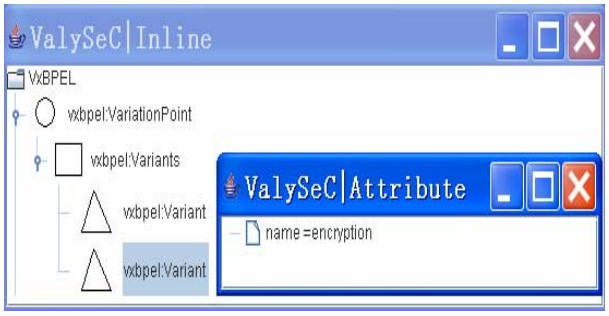


Fig. 7. A snapshot of *ValySeC* with the Inline type

VxBPEL supports complex variable service compositions and the ConfigurableVariationPoint is used to specify the dependencies among variation elements. Fig. 8 shows a snapshot of *ValySeC* with the *Inline Configurable* type. The root node in the variability tree view is vxbepl: VariationConfigurationInformation represented as a rhombus. The vxbepl:ConfigurableVariationPoints that is represented as a four-square set contains one or more ConfigurableVariationPoints represented as a circle. The detailed configuration of each

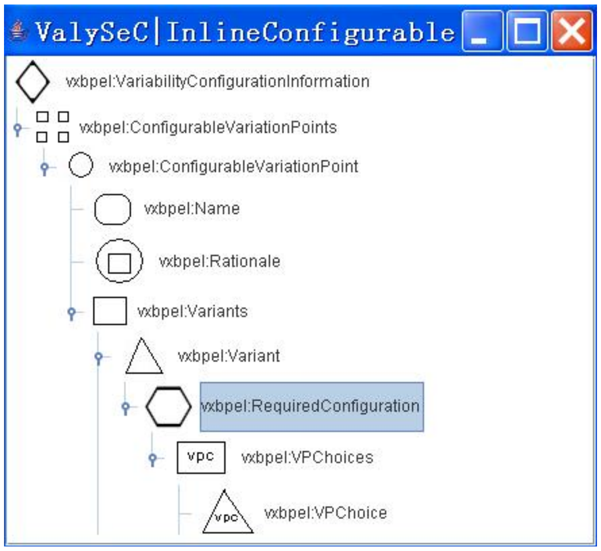

ValySeC|InlineConfigurable
vxbpel:VariabilityConfigurationInformation
vxbpel:ConfigurableVariationPoints
vxbpel:ConfigurableVariationPoint
vxbpel:Name
vxbpel:Rationale
vxbpel:Variants
vxbpel:Variant
vxbpel:RequiredConfiguration
vpc
vxbpel:VPChoices
vpc
vxbpel:VPChoice

Fig. 8. A snapshot of *ValySeC* with the Inline Configurable type

ConfigurableVariationPoint is further elaborated through its name, rationale and variants. The name and rationale are represented as an ellipse and an ellipse filled with a rectangle, respectively. Each variant is elaborated through its name, RequiredConfiguration and a set of VPChoices represented as a rectangle filled with a label of “vpc”. The RequiredConfiguration represented as a regular hexagon can be used to specify the complex dependencies among the configuration, and the VPChoice represented as a triangle filled with a label of “vpc” lists out the possible alternative for this variant.

The tree view can be folded as required. This is particular helpful for the user to focus on some variability elements while omit other elements. The implementation presented above is still a preliminary version. There are still several limitations. Only the variation point view is provided. The dependency view is under development. The variation that *ValySeC* is able to handle lies in the implementation layer. Currently, *ValySeC* focuses on the analysis and maintenance of variability. One of our future tasks is to support the visual variability design based on *ValySeC* and then effectuate the design into service composition specifications.

4. A case study

We use the loan approval application to examine the effectiveness and performance of *ValySeC*.

The application is implemented by several Web services taken directly from the WS-BPEL 2.0 website [13]. This application has also been used to examine the VxBPEL and its interpreter [9]. With this loan approval application, customers send their requests for loans, including personal information and the amount being requested. Based on the information, the loan service runs a simple process that results in either a ‘‘loan approved” message or a ‘‘loan rejected” message. The approval decision can be reached in two different ways, depending on the amount requested and the risk associated with the requester. For low amounts (less than $10,000) and low-risk individuals, approval is automatic. For high amounts or medium and high-risk individuals, each credit request needs to be studied in greater detail.

To process each request, the loan service uses the functionality provided by two other Web services. In the streamlined processing available for low amount loans, a ‘‘risk assessment” service is used to obtain a quick evaluation of the risk associated with the requesting individual. When the streamlined process is not applicable, a ‘‘loan approval” service is used to obtain in-depth assessments of requests.

The graphic representation of the BPEL specification for the loan approval process is illustrated in Fig. 9. The orchestration process involves five activities. Each is a variation point because it may require different processing, namely default or encrypted. Thus, the variability design applies to these activities. Fig. 10 shows a small part of such a VxBPEL specification.

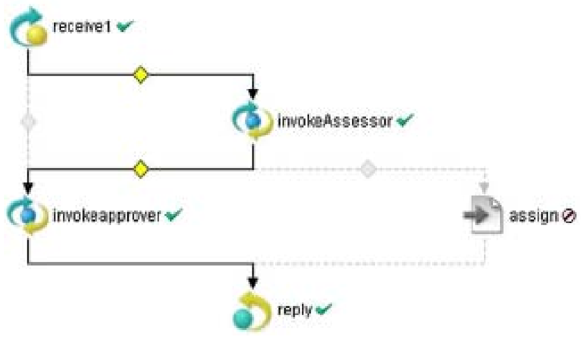
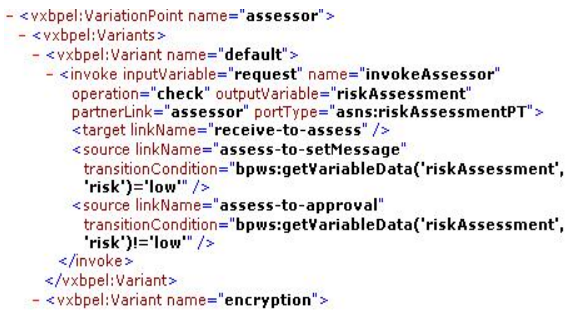
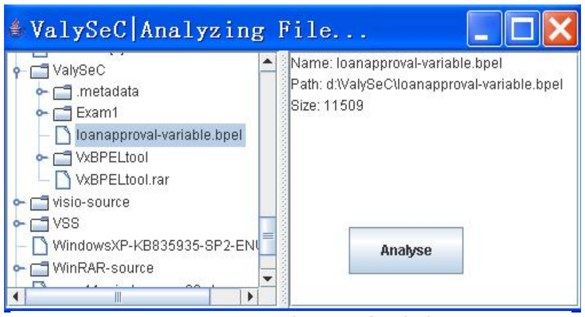


Fig. 9 The flowchart of BPEL specification for the loan approval process



```
- <vxbpel:VariationPoint name="assessor">
  - <vxbpel:Variants>
    - <vxbpel:Variant name="default">
      - <invoke inputVariable="request" name="invokeAssessor"
          operation="check" outputVariable="riskAssessment"
          partnerLink="assessor" portType="asns:riskAssessmentPT">
          <target linkName="receive-to-assess" />
          <source linkName="assess-to-setMessage"
            transitionCondition="bpws:getVariableData('riskAssessment',
            'risk')='low'" />
          <source linkName="assess-to-approval"
            transitionCondition="bpws:getVariableData('riskAssessment',
            'risk')!='low'" />
        </invoke>
      </vxbpel:Variant>
    - <vxbpel:Variant name="encryption">
```

Fig.10. An segment of VxBPEL for the loan approval application



Name: loanapproval-variable.bpel
Path: d:\ValySeC\loanapproval-variable.bpel
Size: 11509

Fig. 11. A snapshot of *ValySeC* for the loan approval application

Now, we employ *ValySeC* to view and maintain the variation information contained by such a variable service composition using VxBPEL. Fig. 11 shows a snapshot when *ValySeC* is used to analyze the VxBPEL specification of the loan approval. Results of the analysis are illustrated in Fig. 12 and reveal that the VxBPEL specification for the loan approval application contains *inline* and *inline configurable* variation configurations. In the section of *inline*, there

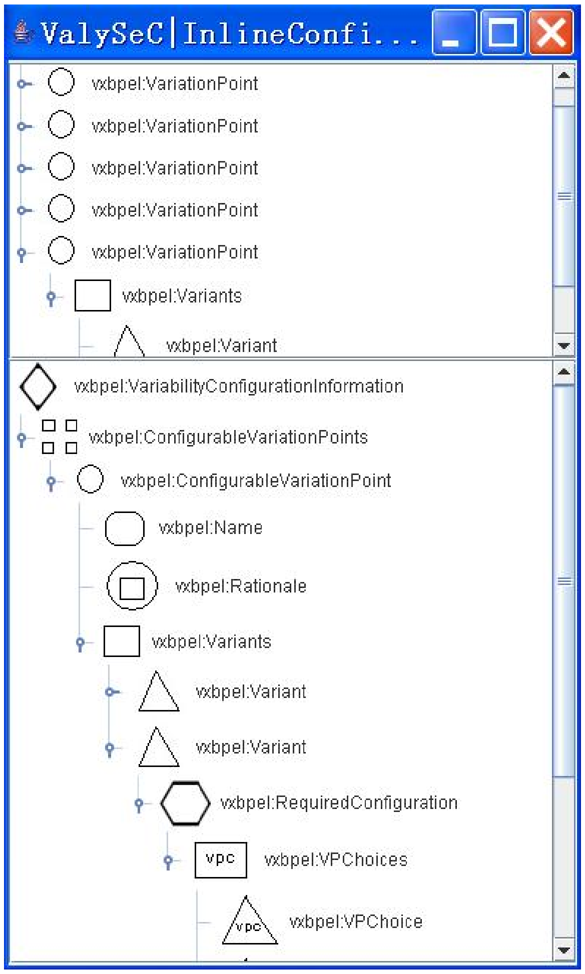


Fig. 12. A snapshot of variation view of the loan approval application

are totally five variation points and each of them is associated with two variants. In the section of *inline configurable*, there is one ConfigurableVariationPoint. It defines two complex variants and they are further defined by their RequiredConfigurations which are specified in terms of a set of VPChoices. By analysing their VPChoices, we find that the first variant requires all "default" processing, while the second one requires all "encrypted" processing.

As to the performance of *ValySeC*, the time overhead mainly results from two aspects. One is related to the computation time that is needed to parse the variation elements in a variable business process and proportional to the amount of variation elements contained by the process. The other is related to the reasoning of the relationships among the variation elements. Compared with the former overhead, the latter one is negligible. Table 1 summarizes the performance of *ValySeC* when it runs the loan approval application in a laptop.

Table 1. The performance of *ValySeC* with the loan approval application

CPU (GHz)	1.8*2
Memory (GB)	1.0
Operating System	Windows XP
Time (ms)	94

With this case study, we have applied *ValySeC* to the loan approval application, a tiny but representative variable service composition using the VxBPEL. Results of the case study have validated the feasibility and performance of *ValySeC*.

5. Related work

We describe related work on variability modeling for Web services, and adaptive service compositions.

Mohan and Ramesh [12] present an approach that makes use of ontology for variability management in product and service families. The ontology is integrated by a Knowledge Management System to assist designers of a system in implementing variability. The advantages include that it offers flexibility in the use of different mechanisms for implementing variability and it is domain independent. However, a drawback is that it requires major involvement from the user, which means the approach can not be used for automatic reconfiguration of a system.

Robak and Franczyk [15] introduce the concept of modeling the variability of Web services using feature diagrams. For a Web service-based system a feature diagram can be created describing the commonalities and differences within the range of possible systems. The advantages includes that it supports automated configuration of a system and provides a clear overview of the variability and commonalities within a system. However, describing variability only in this manner means that realization relations and dependencies are not modeled.

Kim and Doh [10] introduce a framework for modeling adaptable Web services by analyzing the variability. Their modeling framework focuses on the variability of Web services from variation points of structural features and behavioral features. Topaloglu and Capilla [22] discuss the modeling of variability in terms of pattern. Unlike these work, the VxBPEL is developed to address variability design of adaptive service compositions, in particular in connection to BPEL.

AdaptiveBPEL [8] is a service composition framework which leverages aspect-oriented techniques to provide better dynamic adaptability of Web services. Extra concerns are specified as *aspects* and the aspects are weaved into BPEL processes through the policy-driven negotiation process. AOBPEL [4] is an aspect-oriented extension to BPEL which provides a solution to the lack of appropriate means for the modularization of crosscutting concerns and for supporting dynamic changes in BPEL. These aspect-oriented extensions effectively solve the scatter and the tangling problems. However, aspect definitions split up the process logic over different files and thus it difficult to comprehend variation of, in particular, complex service compositions. The implementation of aspects results in a hybrid specification of service compositions. VxBPEL overcomes these advantages by providing a set of variability constructs whose style is consistent with BPEL native constructs and a variation view which is a great aid for comprehending variation of service compositions.

Trap/BPEL and its predecessors [7] are a family of extensions to BPEL for enhancing the robust web services compositions by means of static, dynamic and generic proxies, respectively. The adapted process is augmented with a proxy that replaces failed services with predefined or newly discovered alternatives. Unlike VxBPEL, the approach does not treat changes as first class citizens in the Web service compositions, and hence it is difficult to know about the variation points and their dependencies causing variation configuration and maintenance a difficult task.

6. Conclusions and future work

We have presented a variability analysis tool *ValySeC* which can be used to automatically analyze and maintain the variability information within a service composition using VxBPEL. The VxBPEL improves the standard BPEL with explicit variability management and results in a great deal of flexible design. When the VxBPEL is used for the variability design of complex service compositions, the analysis and maintenance of variability information is a big challenge. *ValySeC* presented in this paper is designed to address this challenge. The effectiveness of *ValySeC* has been validated by a case study.

Our ultimate goal is to develop a variability-based adaptive service composition methodology and platform. Currently, the VxBPEL engine has been integrated with ActiveBPEL engine to support the execution of VxBPEL specifications. For the future work, we plan to extend *ValySeC* to support variability design, and integrate it with the ActiveBPEL Designer to provide an efficient variability analysis, design and maintenance platform.

Acknowledgements

The authors thank all the contributors of the COVAMOF framework and VxBPEL. This research is supported by the National Natural Science Foundation of China (Grant No. 60903003), the Research Fund for the Doctoral Program of Higher Education of China (Grant No.2008000401051), the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, China (Grant No. 2008[890]), and NWO Jacquard project Software as a Service for the varying needs of Local eGovernment (SaS-LeG, http://www.sas-leg.net), contract no. 638.000.000.07N07.

References

[1]ActiveBPEL, http://www.activebpel.org/. 2007

[2]M. Aiello, P. Bulanov, H. Groefsema. Requirements and Tools for Variability Management. *Proceedings of REFS 2010 in conjunction with COMPSAC 2010*, pp245-250.

[3]F. Bachmann, L.J. Bass. Managing variability in software architectures. *Proceedings of ACM SIGSOFT Symposium on Software Reusability*, 2001, pp126–132.

[4]A. Charfi, M. Mezini. AO4BPEL: An Aspect-Oriented Extension to BPEL. *World Wide Web*, 2007,10(3):309-344.

[5]F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Weerawarana, Business process execution language for Web services, Version 1.1 2003.

[6]S. Deelstra, M. Sinnema, J. Bosch. Product derivation in software product families: a case study. *Journal of Systems and Software*, 2005, 74(2):173–194.

[7]O. Ezenwoye, S. M. Sadjadi. TRAP/BPEL: A Framework for Dynamic Adaptation of Composite Services, http://www.cs.fiu.edu/~sadjadi/Publications/, 2006.

[8]A. Erradi, P. Maheshwari. AdaptiveBPEL: a Policy-Driven Middleware for Flexible Web Services Compositions, *Proceedings of Middleware for Web Services (MWS)*, 2005, pp5-12.

[9]M. Koning, C. Sun, M. Sinnema, P. Avgeriou. VxBPEL: Supporting variability for Web services in BPEL. *Information and Software Technology*, Elsevier, 2009, 51(1): 258-269.

[10]Y. Kim, K. Doh. Adaptable Web Services Modeling using Variability Analysis, *Proceedings of Third 2008 International Conference on Convergence and Hybrid Information Technology*, 2008, pp700-705.

[11]F. Linden. Software product families in Europe: The Esaps & Cafe Projects. *IEEE Software*, 2002, 19(4):41-49.

[12]K. Mohan, B. Ramesh. Ontology-based support for variability management in product and service families. *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003, p75.1.

[13]OASIS. Web Services Business Process Execution Language Version 2.0 Committee Draft, <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel>, 2006.

[14]M. Papazoglou. P. Traverso, S. Dustdar, F. Leymann. Service-Oriented Computing: a Research Roadmap. *International Journal on Cooperative Information Systems* (IJCIS), 2008, 17(2):223-255.

[15]S. Robak, B. Franczyk. Modeling Web services variability with feature diagrams. *Proceedings of the NODe 2002*, Springer, 2003, pp120－128.

[16]M. Sinnema, S. Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, 2007, 49(7): 717-739.

[17]M. Sinnema, S. Deelstra, P. Hoekstra. The COVAMOF derivation process, *Proceedings of ICSR 2006*, LNCS 4039, Springer, 2006, pp101–114.

[18]M. Sinnema, S. Deelstra, J. Nijhuis, J Bosch. COVAMOF: a framework for modeling variability in software product families. *Proceedings of the Software Product Line Conference (SPLC2004),* 2004, pp197-213.

[19]C. Sun, M. Aiello. Towards variable service compositions using VxBPEL. *Proceedings of the International Conference on Software Reuse (ICSR)*, LNCS 5030. Springer, 2008, pp257-261.

[20]C. Sun, R. Rossing, M. Sinnema, P. Bulanov, M. Aiello. Modeling and managing variability of Web service-based systems, *Journal of Systems and Software*, Elsevier, 2010, 83 (3): 502-516.

[21]C. Sun, J. Zhou, J. Cao, M. Jin, C. Liu. ReArchJBs: a Tool for Automated Software Architecture Recovery of JavaBeans-based Applications, Proceedings of 16th Australian Conference on Software Engineering (ASWEC2005), 2005, pp270-280.

[22]Y. Topaloglu, R. Capilla. Modeling the variability of Web services from a pattern point of view. *Proceedings of ECOWS 2004*, LNCS 3250, Springer, 2004, pp128-138.