XChor
Choreography Language for Integration of Variable Orchestration Specifications

Selma Suloglu[1], Bedir Tekinerdogan[2] and H. Ali Doğru [1]
[1]*Computer Engineering Department, Middle East Technical University, Dumlupınar Boulevard, Ankara, Turkey*
[2]*Computer Engineering Department, Bilkent University, Ankara, Turkey*
{Selma, dogru}@ceng.metu.edu.tr, bedir@cs.bilkent.edu.tr

Keywords: Service-Oriented Architecture, Choreography, Choreography Languages, Variability Management, Variability Metamodel.

Abstract: In this paper, we propose and develop a new choreography language XChor which can be used to support variability in choreography specifications and integrate these with variability of orchestration specifications. We describe the metamodel of XChor and illustrate the adoption of the language by specifying user verification choreography in the adaptable security system. Orchestration and choreography models are mechanisms to realize service composition and coordination while some of them support variation to deal with reuse challenge. Several approaches have been introduced to support variability in orchestration and choreography languages. Unfortunately, variability is not explicitly addressed in current choreography languages. As such, it is hard to provide a consistent configuration of service composition within and across business organizations.

1 INTRODUCTION

Several organizations develop, share and reuse business processes by establishing collaboration with other organizations in order to fulfill different stakeholder needs. Being agile is an important challenge in business process integration context which requires a dynamic environment. Service-oriented architecture (SOA) is a promising approach to realize such environments by designing and developing distributed systems (Erl, 2005). SOA aims to facilitate reuse of services and incorporates service consumers and service providers. A service is self-contained, and can be independently deployed in a distributed component.

Building enterprise solutions to realize business processes typically requires the composition of multiple existing enterprise services. Composite services can be further recursively composed with other services to derive higher level solutions. Two different types of service compositions are defined: 1: service choreography where the interaction protocol between several partner services is defined from a global perspective. 2: service orchestration, where the interaction logic is specified from the local point of view of one single participant, called the orchestrator.

Reuse in SOA can be achieved by managing variability in different granularity levels, namely choreography, orchestration and atomic services. Assuming that all granularity levels can be treated as services, variability can come from (i) their interfaces (functions and parameters), (ii) connectors (the way they interact) and (iii) composition (the way they are gathered in order to achieve a goal). Interface variability requires a configuration mechanism specifying when and how to change its functions and parameters. Connector variability needs a relation mechanism to indicate when and which connector is used between two services. Composition variability necessiates a tailoring mechanism to define in which order and how services are interacting with each other. Services offer different functionalities regarding their variability bindings. Therefore, it is the composition's responsibility to provide a consistent variability binding between interacting services. This requires a mechanism to establish variability associations which determines when and how interacting services bind to specific variants. In other words, composition is responsible for handling consistent variability binding of interacting services and providing a configuration infrastructure.

Addressing and fulfilling all these variability

Suloglu S., Tekinerdogan B. and Dogru A.
XChorChoreography Language for Integration of Variable Orchestration Specifications.
DOI: 10.5220/0004774601210130
In *Proceedings of the Third International Symposium on Business Modeling and Software Design (BMSD 2013)*, pages 121-130
ISBN: 978-989-8565-56-3

needs to provide seamless integration of services. To cope with such challenges several approaches have been introduced. However, explicit introduction of variability integrated with choreography languages is not addressed. Specification of consistent variability binding and configuration of interacting services are not considered in the choreography language level. Moreover, there is a lack of support to reuse existing choreographies.

In this article, we first analyse and discuss existing orchestration and choreography languages with respect to variability management. We identify the problems and the requirements for variability in choreography languages. To support interface and composition variability in choreography specifications we developed a new domain specific language called XChor.

The remainder of the paper is organized as follows. Section 2 firstly describes variability management in existing choreography and orchestration languages. Then the requirements for managing variability in choreography languages are defined and problems in existing languages are stated. Section 3 introduces the metamodel developed by authors for supporting variability in choreography specifications and integrating these with variability of orchestration specifications. Section 4 describes the XChor language and demonstrates its usage through an example. Finally section 5 provides the conclusions.

2 VARIABILITY MANAGEMENT IN EXISTING ORCHESTRATION AND CHOREOGRAPHY LANGUAGES

Obviously for small systems we could handle orchestration specifications using traditional approaches such as interaction diagrams. Variable parts and their relations can be modeled and implemented by data and through 'if' control structures. However, for integration of large scale systems soon the traditional approaches are less expressive and not tractable. Therefore, to cope with variability in choreography, orchestration, and atomic services various language approaches have been introduced. We have listed the popular approaches in Table 1. We evaluate these approaches with respect to the following criterias:

- *Composition Approach:* Defines whether the language supports choreography and/or orchestration. Orch is the abbreviation of orchestration and Chor is that of choreography.
- *Variability Support:* Defines whether the language supports variability. 'Yes' indicates that the language provides explicit language mechanisms for variability. 'Implicit' indicates that although the language does not provide explicit mechanisms, variability is supported implicitly. 'No' means that there is no variability support.
- *Tool Support:* Availability of tools.
- *Modelling Approach:* Defines the adopted modelling approach which can be either based on interaction or interconnection. Modeling based on interaction represents definition of one building block (document or specification) for the whole system, whereas interconnection suggests modeling control flow logic per participant. Intera is the abbreviation of interaction and Interc is that of interconnection.

BPEL (OASIS 2007), VxBPEL (Koning et al., 2009), Jolie (Montesi et al., 2007) and Jorba (Lanese et al., 2010) purely target orchestration as the composition approach and interconnection as the modelling approach. Among them VxBPEL has an explicit variability model. On the other hand, Jorba, a rule-based approach to dynamic adaptation implemented on top of the Jolie language, provides a mechanism without explicit specification of variability.

WSMO (Fensel et al., 2007), BPMN (OMG 2011) and Reo (Arbab, 2004) target orchestration and choreography specification. While WSMO provides an interconnection model, Reo and BPMN include interaction and interconnection models. Among them, Reo offers variability support by hyper-graph transformation.

BPEL abstract processes, WS-CDL (W3C, 2005), Let's Dance (Zaha et al., 2006), MAP (Barker et al., 2009), BPEL4Chor (Decker et al., 2007), and an extension of it – BPEL^{gold} (Kopp et al., 2010) all target choreography for service composition. An interaction modelling approach is followed by WS-CDL, Let's Dance and MAP, whereas interaction model is applied in BPEL abstract processes, BPEL4Chor and BPELgold. Moreover, MAP supports an interaction model by separating choreography definition to peers related with services.

According to Table 1, the languages supporting variability are VxBPEL, Jorba and Reo. VxBPEL language seems to be the only language which provides explicit support for variability mechanisms based on ConIPF Variability Modeling. Framework (COVAMOF) (Sinnema, Deelstra, Nijhuis, Bosch, 2004.). The approach extends BPEL with variability constructs, such as <<VariationPoint>> and <<Variant>>. However, the language does not support variability of choreography. In parallel, there is no mechanism to inspect the global view of variability when more than one VxBPEL orchestration interacts.

Table 1: Comparison of existing orchestration and choreography languages.

	Composition Approach	Variability Support	Tool Support	Modelling Approach
BPEL 2.0	Orch	No	Yes	Interc
VxBPEL	Orch	Yes	No	Interc
Jolie	Orch	No	Yes	Interc
Jorba	Orch	Implicit	Yes	Interc
Reo	Orch Chor	Implicit	Yes	Interc Intera
WSMO	Orch Chor	No	Yes	Interc
BPMN 2.0	Orch Chor	No	Yes	Interc Intera
WS-CDL	Chor	No	No	Intera
Let's Dance	Chor	No	No	Intera
BPEL4Chor BPEL^{gold}	Chor	No	Yes	Intera
MAP	Chor	No	Yes	Interc Intera

On top of the Jolie orchestration language, Jorba defines adaptation interfaces specifying function replacements whenever a change in service interface and parameter is needed. However, the relationship between rules and the coverage of variability is implicit and the management of rules as a separate variability model is usually difficult to manage. There is no mechanism to explicitly specify variation points and variants as in VxBPEL tags.

Reo, a comprehensive approach to service composition proposes a hyper-graph transformation approach to manage change. Services as nodes are connected via edges. In other words, variability is provided by reconfiguration of services which is seen as an internal part of the system. Therefore, there is no explicit variability model defined to intervene and change the composition, accordingly no explicit specification of relations between variability of services taking part in the composition.

2.1 Problem Statement

The analysis of the existing choreography languages shows that variability in both orchestration and choreography is not supported in any of the languages. Besides, interface and composition variability support is not explicitly addressed with a single variability model covering choreography, orchestration and atomic services. Concretely we can identify the following problems:

- *Lack of explicit expressiveness of variability in choreography specifications*

There is no language that explicitly represents variability in choreography in order to integrate orchestration specifications. Moreover, variability modelling in choreography, orchestration and atomic services as a whole is not explicitly covered in one single model. This impedes the consistent configuration of choreography and orchestration specifications with regard to variability.

The lack of explicit abstractions for variability easily leads to the scattering of variability concerns over service compositions. Likewise, enabling or disabling a variability results in reorganization of the composition. This complicates the understanding of variable parts, relations amongst them and the overall goal for business process engineers and developers. Tracing these scaterred variations can be achieved to a certain degree, but in large scale systems traceability and understandability decrease gradually. As a result, this scattering reduces the maintenance of the system.

- *Lack of explicit specification of variability associations between interacting services*

Choreography interrelates a set of orchestrations, atomic services and establishes connection with other choreographies. Interacting services' variability constraints and shapes possible choreography abilities and composition. Likewise, variability of choreography dictates proper service variability bindings and specified configurations resulting service interfaces with different functionality and parameters. In order to reveal these dependencies and relations between choreography and services, an explicit association and mapping should be defined. In other words, configuring choreography requires configuring other services in order to consistently collaborate with each other. Therefore, configuration and binding of service variability requires an integrated model comprising choreography, orchestration specifications, and atomic services with variability. There is no language supporting such integrated configuration model dealt with variability of all granularity levels.

- *Lack of support for reusing existing choreographies*

The importance of reusing existing choreographies is addressed in some approaches, but reusing as a part of the other choreography is not emphasized sufficiently. There are ways to handle choreography-to-choreography relationships such as collaborating via exposed choreography interfaces. In case of variability, it is more difficult to utilize choreography specifications with proper bindings. Therefore, the way to bind to other choreographies should be specified.

Although several choreography languages address some of the above stated concerns, no single orchestration or choreography language covers all of them. Moreover, there is no specified mechanism to associate and map orchestration and choreography variability for consistent integration. Even if variabilities for choreography, orchestration and atomic services are explicitly specified, seamless and consistent mapping cannot be achieved due to different concepts and capabilities of different variability models. In that, one variability model can constrain the other one. For instance, COVAMOF model used in VxBPEL orchestration specification does not have external variation definition and can not be completely mapped with a model providing external variation. Therefore, the modeling of variability in choreography consistent with orchestration and atomic services cannot be achieved easily. To support the systematic management of variability and the consistent composition of choreography specifications, a choreography model that incorporates variability concepts is needed.

3 A METAMODEL FOR VARIABILITY MANAGEMENT IN CHOREOGRAPHY

To enable integration of orchestrations, atomic services in the scope of choreography, we propose a metamodel in which atomic services and orchestrations are evaluated under service concept. The main difference in specification between orchestrated and atomic service comes from revealing external behavior to service environment. That is, orchestrated service can define external interaction with other services if required. Moreover, there is no constraint that an atomic service can not specify its interaction. Therefore, atomic services and orchestrations are treated as services in our metamodel.

The metamodel basically enables to define choreographies and services, to specify variability of each one and to integrate these variabilities in order to provide a consistent collaboration. Figure 1 depicts the overview of service and choreography relations based on our metamodel so as to support interface and composition variability. Two main blocks are depicted; choreography and service.

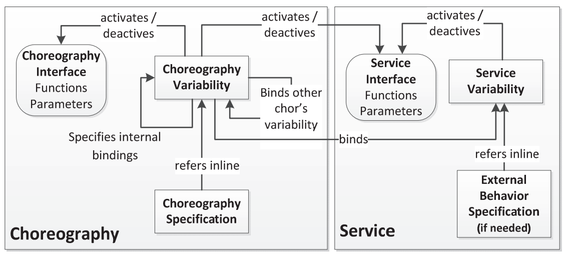

Figure 1: Overview of choreography and service relations based on our metamodel.

Both choreography and service, interfaces without variation are defined fulfilling all possible functional requirements. Choreography interface is only configured with regard to its own variability specification, whereas service interface is configured via both its own variability specification and variability specification of choreography that takes part in. Configuration of service is achieved by activating/deactivating functions and setting/unsetting parameters. With this mechanism, different choreographies utilize different interfaces of the same service which brings service reusability.

Choreography variation leads to proper bindings of variations of other choreography and services via mapping so as to provide interacting interface consistency. Choreography and external behavior specification of services include inline references of their own variability to point out the changeable parts.In this way, choreographies and services include a set of possible required behavior in order to fulfill different composition needs, which enables reuse of choreography and services.

3.1 Variability Specification

The rightmost part of the metamodel in Figure 2 presents the variability specification constructs. This part has been defined based on existing variability metamodels in the literature. A comparative literature study has been conducted in (Lianping et al., 2009). Based on this part of the metamodel, choreography and services can define their internal

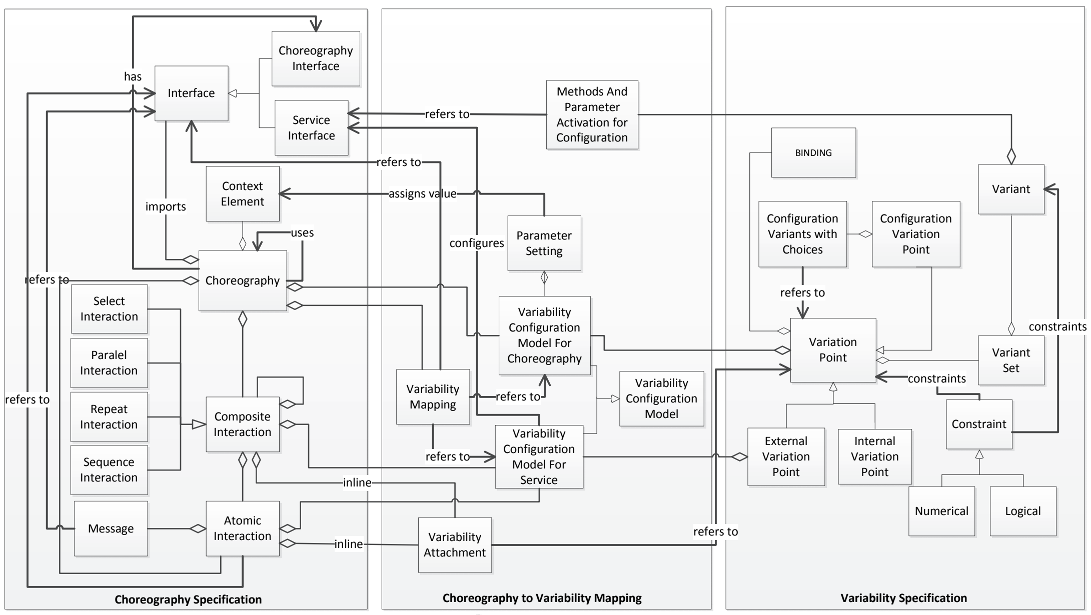

Figure 2: The metamodel for variability management in choreography.

and external variation points, related variants and constraints among them. A special type of variation point, configuration variation point eases management and understanding of variability mechanism by holding details of internal variation point bindings.

For different variation point relationships, contraints provide a mechanism to establish a convenient binding and selection by defining numerical and logical constraints.

3.2 Choreography Specification

The leftmost part of the metamodel represents the elements to define a choreography composition and interfaces of choreography and services. Choreography comprises a set of services and choreographies, identifying composability via service interactions. Service interactions specify the way how the services collaborate which is realized by atomic and composite interactions.

Choreography and service interfaces expose a set of functions without variability specifications. Other than services, a choreography interface states required functions from other services and choreographies.

3.3 Choreography to Variability Mapping

The middle part of the metamodel represents the concepts to define the mapping between choreography and variability constructs. Mainly these constructs are responsible for configuration of interfaces, establishment of variability associations and representing variability references in composition.

Variability configuration model for service and choreography includes a set of variation points, constraints among them and service interactions (for services only). Variability Association facilitates choreography to identify proper bindings of utilized service and choreography variability.

Methods And Parameter Activation for Configuration provides a configuration mechanism to define method activation/deactivation and parameter setting/unsetting of referred service interface.Variability attachment specifies conditions of variation point and variant selections used in choreography composition. Tagging with variability attachment specifications, the parts of the composition gains dynamicity that changes the behavior of choreography. When conditions are satisfied, the part is added to the final composition.

4 XChor LANGUAGE

The authors have developed a new domain specific language, XChor, based on the metamodel that we have described in the previous section. XChor (XChor, 2012) has been implemented using Xtext (Xtext, 2012) in the Eclipse development environment.

XChor Language facilitates to create three different models. Configuration interface models cover variability specifications stated in (Nguyen et al., 2011). Choreography model can specify twelve service interaction patterns described in (Barros et al., 2005).

The basic elements of XChor is shown under three model to cope with variability in choreography. Models are exemplified based on a part of a real life case study, verification of a user in adaptable security system.

Adaptable Security System is an authentication system residing between customers and third party applications or institutions that supports different authentication types of data, including software and hardware (biometric device) parts. The system has the ability to be integrated and applied to a military installation or to a banking system, which requires fulfilling different stakeholder needs. Applicability to different stakeholder systems requires different functionality support and behaviour. User verification can be done offline or online by a third party authority such as web services or certain devices like: PDA, PC, ATM, or mobile phone. The third party authority gets different types of data as required user credentials: (1) username and password, (2) username and password with instant mobile text, (3) e-sign, (4) biometric data; fingerprint, finger vein, and/or iris. Then, according to the online and offline verification result, the system will allow or ban users entering the integrated application.

Device support is important as different devices have different capabilities. ATM, PDA and mobile phone can be used with (1), (2) and (3). PC supports (1), (2), (3) and (4). Therefore, the system should change verification processing functions according to used devices and parameters to be verified.

While modeling this system, user verification is treated as a choreography utilizing other choreographies and services. In the following sections, (i) configuration interfaces for defining and managing variability of choreographies and services, (ii) interfaces of user verification choreography and interrelated services, and (iii) user verification choreography are depicted.

4.1 Configuration Interface

Configuration interface model covers service and choreography variability specifications internally and externally to depict possible abilities, to configure others and to be configured by others. To depict possible abilities; Choreography can specify internal, external and configuration variation points, whereas services can only depict external variation points. The external ones are used to be configured by choreographies and services. Capabilities to configure its own interface or other services' intefaces as activating/deactivating and setting/unsetting parameters are also specified in this model. Numerical or logical constraints among variability specifications are depicted.

Different user authentication types such as biometric authentication, supported authentication modes (online and/or offline), transaction types (real or fake transaction) are the system's behaviours need to be configured differently. Therefore, each is treated as variability in configuration interface of user verification choreography.

To enable authentication variability, both types of authentication and parameters used in encryption function are changed with regard to the usage of biometrics or not. For this purpose a configuration variation point named as "authentication_type" as external and two internal variation points "i_auth_type" and "i_encryption_parameters" are defined. Binding of "authentication_type" configures consistent bindings of "i_auth_type" and "i_encryption_parameters".

"i_auth_type" is specified with "internalVP" keyword (line 5). "username_passw" is a mandatory variant, whereas "onetimepassw" (line 9) and "esign" (line 10) are optional in other words can be selectable. At least one and at most two variants can be selected among the following alternatives: "fingerprint" (line 12), "fingervein" (line 13), "iris" (line 14), and "face" (line 15). The binding time of this variation point is runtime (line 17). "authentication_type" is specified as external (line 26). The variation point has two optional variants specified (lines 29-30); "userinfo" and "biometrics". "userinfo" variant is realized (line 33) by selection of "defaultparams" variant of "i_encryption_parameters" variation point.

For "biometrics", the realization requires two selections at the same time: (i) minimum one variant among "fingerprint fingervein iris face" set should be selected from "i_auth_type" variation point (line 35) and "setparams" variant of "i_encryption_parameters" variation point (line 36). Default variant of the "authentication_type" configuration variation point is "userinfo" (line 37). Configuration type is parameterization and it is bound at development time represented as "devtime" (line 39).

```
1 Configuration interface vconf_verification of choreography userverification
2
3  //determines number of different biometric authentication types
4  @composition
5  internalVP i_auth_type:
6        mandatory
7              variant username_passw
8        optional
9              variant onetimepassw
10             variant esign
11       alternative
12             variant fingerprint
13             variant fingervein
14             variant iris
15             variant face
16             (min:1,max:2)
17       bindingTime runtime
18 //determines authentication mode
19 @composition
20 internalVP i_auth_mode:
21       optional
22             variant mode_online:activateMethods(service:thirdparty,funct:getconnection,savehasheddata,verify)
23             variant mode_offline:activateMethods(service:storage,funct:get_hashed_data)
24       bindingTime devtime
25
26 configuration authentication_type:
27       varType externalVP
28       optional
29             variant userinfo
30             variant biometrics
31       realization "it is realized by i_encryption_parameters and i_auth_type variability points"
32       confvariant userinfo mapping
33             VPName i_encryption_parameters selectedVariants(defaultparams)
34       confvariant biometrics mapping
35             VPName i_auth_type selectedVariants(fingerprint fingervein iris face; min:1, max:1)
36             VPName i_encryption_parameters selectedVariants(setparams)
37       defaultVariant userinfo
38       type parameterization
39       bindingTime devtime
```

Figure 3: Configuration interface of user verification choreography.

```
1 Constraints
2     i_auth_type requires i_auth_mode selectedVariants(mode_online)
3     i_auth_mode mode_online const protocol = "https"
4     i_auth_type esign const i_encryption_parameters defaultparams = "username,password and esign"
5
6  Parameter Settings
7        parameter noofbiometricauthtypeselected = #ofSelectedVariants{fingerprint fingervein iris face} Of i_auth_type
8        parameter defaultparams = [username_passw] + [selected{onetimepassw,esign}]
```

Figure 4: Constraint and parameter setting specification in configuration interface of user verification choreography.

Any variant can activate required functions in service and choreography interfaces. “i_auth_mode”, internal variation point (line 20) is responsible for activation of different functions of storage and thirdparty services when its related variants are selected. For instance, “mode_online” varaint activates “getconnection, savehasheddata, verify” functions of thirdparty service when selected (line 22).

Constraints includes a logical constraint (line 2), stated that “i_auth_type” variation point requires “mode_online” variant of “i_auth_mode” variation point to be selected. In lines 3-4 numerical constraints are depicted in which “mode_online” variant of “i_auth_mode” variation point contraints the “protocol” property to be set to “https”.

Moreover, any variability in choreography configuration interface that affects context elements in choreography can be defined in Parameter Settings part. Their values are set when the choreography is configured. For instance, “noofbiometricauthtypeselected” in Figure 5 (line 31) identifies the number of times for extracting features from biometric data. Its value is assigned (line 7) when variants of “i_auth_type” is selected.

4.2 Choreography

Choreography model includes composition constructs with variability attachments, context elements and variability associations between interacting services and choreographies. User verification choreography composes nine different services and interacts with three other choreographies. Importing collaborating choreographies and services with or without their own configuration interfaces provides an opportunity to utilize them with different configuration interfaces, that is with different service interfaces.

```
1 choreography userverification
2
3    import configuration vconf_verification
4
5    use choreography chor_warning
6    use choreography chor_warning
7    use choreography chor_connection
8
9    import service encryption with configuration vconf_encryption
10   import service imageretrieval
11   import service credentials
12   import service storage
13   import service attemptcalc
14   import service comparison with configuration vconf_comparison
15   import service responsewindow
16   import service interfaceprep with configuration vm_interfaceprep
17   import service thirdparty with configuration vm_thirdparty
18
19   Context Elements
20    wrongattempts 0
21    fakeinterface false
22    noofbiometricauthtypeselected 0
23
24   Choreography Variability Mapping
25    VP i_encryption_parameters maps service encryption VP encryption_params
26          Variant defaultparams maps Variant withdefaultparams
27          Variant setparams maps Variant withparams
28        ...
29   Function verify:
30       sequence (
31       #vp i_auth_type ifOneSelected( fingerprint fingervein iris)# repeat noofbiometricauthtypeselected times
32       (
33          imageretrieval receive message extractfeatures(biometric_data) refers imageretrieval.extract_features
34       )
35
36       ...
37      #vp i_auth_mode ifSelected(mode_offline)# sequence (
38          encryption send{storage} referedDestinations (comparison) message sendstoreddata() refers storage.get_hashed_data
39          #vp i_transaction_type ifSelected(faketransaction)# storage send{comparison} message compare(hasheddata) refers
                                                                                                 comparison.compare
40       )
41       comparison send{attemptcalc} message calculateworngattemps(result) refers attemptcalc.calculate_wrong_attempts
42       %comp wrongattempts = attemptcalc.calculate_wrong_attempts%
43       ...
44    )
```

Figure 5: User verification choreography specification with XChor.

```
Service interface encryption

function encrypt
      precondition(sessioncreated == true)
      postcondition(data_encrypted == true)
      input(credentials)
      output hasheddata

function setparams
            precondition(params_required      ==
true)
            postcondition(set_params == true)
            input(parameters)

      portName encryption binding hostname:8082
```

```
Choreography interface chor_verification of userverification

function verify
      precondition(authentication_mode_selected == true)
      postcondition  (verification_result_set == true)
      input(user_info)
      output response

portName verifyuser binding hostname:8082

      required interfaces
            from chor_warning function { warn }
            from chor_connection function { closeconnection }
            from chor_alert function { alert }
```

Figure 6: Encryption service and user verification choreography interfaces.

Variables defined with their default values based on the Context Elements part are affected by service interactions. For instance, "wrongattempts" is newly specified here to store the number of wrong attempts to limit verification trials.

User verification choreography associates its internal variation points and related variants to those of utilized services' in order to configure service interface variability. The association between lines 25-27 ensures that when "i_encryption_parameters" variation point is bound to one of its variants, "encryption_params" variation point of encryption service is bound accordingly to provide a consistent interaction. With this, when defaultparams is selected, encryption service interface is configured with regard to withdefaultparams variant.

User verification choreography carries out "verify" functionality (line 29) comprising a set of interactions. Atomic and composite interactions are tagged with variability attachments whenever the part of the composition is changeable with regard to variability. In Figure 5, the lines 31-34, 37-40, and 39 include attachments referring to specified variation declarations in the configuration interface of the user verification choreography. "#vp i_auth_mode ifSelected(mode_offline)" to depict the point which composition can change (line 37).

4.3 Service and Choreography Interface

Service and choreography interface model comprises only interface specifications without variability. Each choreography and service has its own interface including all possible functionalities to be configured by configuration interfaces.

The interface of encryption service utilized in user verification choreography is shown in the left hand side of Figure 6. Exposed functionalities "encrypt" (line 3), and "setparams" (line 9) with pre-post conditions, input and outputs are depicted. Other services and choreographies can collaborate with it using "encryption" port (line 14).

Interface of user verification choreography; "chor_verification" depicts its functionality "verify" with pre-post conditions, input and output parameters. Different from encryption service interface, it explicitly states required choreographies with a list of functions.

4.4 Tool Support

Xtext is used to implement XChor Language which provides a development environment for domain specific languages to developers with Eclipse IDE integration. XChor files created from three models are: (i) choreography interface, (ii) service interface, (iii) configuration interface for choreography, (iv) configuration interface for service, and (v) choreography specification. These files can be categorized under configuration, services, and choreographies packages respectively in order to increase understandability.

Choreography, orchestration and atomic services are defined with variability specifications in Xtext. Binding variability and revealing a consistent collaboration require association analysis between variability specifications. This analysis requires considering variability constraints, choreography and service configurations (coming from configuration interfaces) with regard to variation selections. For this purpose, XChorS tool is provided

- to analyse variability associations which reveal configuration effects on orchestration and service interfaces,
- to configure choreographies and services with regard to variant selections, and
- to output configured XChor files in a specified destination folder.

XChorS tool employs parsing, variability association analysis, and configuration phases. It also includes some utilities for developers; binding time analysis and variation point redundancy analysis.

The tool parses related XChor files, discovers dependencies and constraints between them which are specified in configuration interfaces and choreography specification. The variability association analysis shows which variation points are related with which services and service functions.

It helps in the configuration phase to determine which services interact with each other and which functions should reside with which parameters in their interfaces. According to variation selections, the tool (i) configures interfaces by enabling and disabling its functions and parameters, (ii) prepares choreography compositions and external behaviour specifications of orchestration by examining whether the parts with variation attachments are included. Finally, the tool outputs configured choreography and related services and configuration interfaces if there are variation points that will be bound at runtime.

5 CONCLUSIONS

Existing orchestration and choreography languages do not address interface and composition variability explicitly.

Moreover, a single variation model covering all granularity levels, namely choreography, orchestration and atomic services is not proposed. Our approach is based on reusing existing architecture via explicit variability definition and management in SOA proposing a solution to fulfill interface and composition variability requirements. Taking into account challenges of variability scattered throughout the architecture, making feasible to develop variable service-oriented systems, and integrating variable orchestration specifications, a new variability meta-model and language; XChor is constructed and explained in detail. Variability constructs are treated as first class entities and can be defined in all granularity levels.

As a result of our contributions, we improve development of variable service-oriented systems reducing their complexity while providing consistent service interaction with regard to variability specifications. We think that in addition to modelling variable choreographies and relating them to orchestrations and services, verification of the model is important. So, verification of variable choreography is taken into consideration as a future work. Moreover, a runtime environment for XChor and relation with standard modelling languages are our current ongoing research.

REFERENCES

Arbab F., 2004. Reo: a channel-based coordination model for component composition, *In MSCS, Journal of Mathematical Structures in Computer Science.* Cambridge University Press.

Barker A., Walton C. D., Robertson D., 2009. Choreographing Web Services, *In Journal of IEEE Transactions on Services Computing.* IEEE.

Barros A.P., Dumas M., ter Hofstede, A. H. M., 2005. Service Interaction Patterns, *In 3rd International Conference on Business Process Management*. Springer.

Decker G., Kopp O., Leymann F. and Weske M., 2007. BPEL4Chor: Extending BPEL for Modelling Choreographies*, In ICWS'07, 5th International Conference on Web Services,* IEEE Computer Society.

Erl T., 2005. *Service-Oriented Architecture: Concepts, Technology, and Design,* Printice Hall, Indiana, 1st edition.

Fensel, D., Lausen, H., Polleres, A., Bruijn, J. de, Stollberg, M., Roman, D., Domingue, J., 2007. *Enabling Semantic Web Services, The Web Service Modeling Ontology*. Springer.

Lanese I., Bucchiarone A., and Montesi F., 2010. A Framework for Rule-Based Dynamic Adaptation, *In TGC 2010, 5th Symposium on Trustworthy Global Computing*. Springer.

Montesi F., Guidi C., Lucchi R. and Zavattaro G., 2007. JOLIE: a Java Orchestration Language Interpreter Engine, *In Journal of Electronic Notes in Theoretical Computer Science*. Elsevier.

OASIS, 2007. *Web Services Business Process Execution Language Specification, WS-BPEL*. OASIS.

OMG, 2011. *BPMN 2.0 Specification.* OMG.

Koning M., Sun C., Sinnema M., Avgeriou P., 2009. VxBPEL: Supporting variability for Web services in BPEL. *In Journal of Information & Software Technology*. Elsevier.

Kopp O., Engler L., Lessen T., Leymann F., Nitzsche J., 2010. "Interaction Choreography Models in BPEL: Choreographies on the Enterprise Service Bus", *In S-BPM ONE 2010 - the Subjectoriented BPM Conference*.

Lianping C., Babar A., Nour M. A., 2009. Variability management in software product lines: a systematic review, *In SPLC'09, 13th International Software Product Line Conference*. Carnegie Mellon University.

Nguyen T., Colman A., Talib M. A., Han J., 2011. Managing service variability: state of the art and open issues, *In VaMoS '11, 5th Workshop on Variability Modeling of Software-Intensive Systems.* ACM.

Sinnema M., Deelstra S., Nijhuis J., Bosch J., 2004. COVAMOF: A Framework for Modeling Variability in Software Product Families, *In SPLC3, 3rd Software Product Lines Conference.* Springer.

W3C, 2005. Web Services Choreography Description Language Specification, W3C.

XChor 2012. XChor Language Representation in Xtext, available at: http://www.xchor.com/XChorLanguage-xtext.pdf.

Xtext, 2012. Xtext 2.3.1, available at http://www.eclipse.org/Xtext/.

Zaha J. M., Barros A. P., Dumas M., Arthur H. M., 2006. Let's Dance: A Language for Service Behavior Modeling, *In OTM 2006, 4th On the Move to Meaningful Internet Systems Conference*. Springer.