IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.

ITP4843

Generating Synchronization Contracts for Web Services

Otto Preiss
Dept. of Information Technologies
ABB Switzerland LtdCorporate Research
5405 Dättwil, Switzerland
otto.preiss@ch.abb.com

Anuj P Shah
Dept. of Industrial and Systems Engineering
GA Inst. of Technology324919 GA Tech Station
Atlanta, GA 30332-1030
ashah@isye.gatech.edu

Alain Wegmann
School of Computer and Communications
Swiss Federal Inst. of Technology
1015 Lausanne, Switzerland
alain.wegmann@epfl.ch

ABSTRACT

Component composition and Web-service composition have in common that the usage pattern of a certain black-box component or Web-service needs to be described precisely. Because usually many interactions between the client and the service provider are required to obtain a certain desired functionality, one would hope to find the specification of permissible interaction sequences with the service provider in service description formalisms, such as the WSDL. This paper shows an approach to automatically generate such interaction specifications for service descriptions from the XMI representation of UML sequence diagrams. It uses an algebraic representation based on path expressions, which are incorporated into the WSDL. The paper further argues for their value at design and runtime.

1. INTRODUCTION

Complex software systems are increasingly built by composing components and web-services, which are usually available as black box entities. To achieve the intended behavior from the composed software systems, it is essential that the usage of the constituent components be syntactically and semantically correct. In the sequel we use the term server component to refer to both a Web-service and a software component [1].

The permissible usage of a server component is precisely specified in what is called a contract. In addition to the standard syntactic conventions, contracts should contain information on the behavior and therefore functionality of the server component. Because valuable, (re)usable server components are typically coarse-grained and provide extensive functionality, obtaining more than trivial functionality requires an involving interaction scenario between client and server. Behavior contracts are currently covered through formal assertions (e.g., pre and post conditions) that are restricted to individual methods. Since the goal of clients is to obtain coherent pieces of functionality that usually span over several methods, assertions related to methods are insufficient. There are no widely accepted means for precisely specifying and representing permissible invocation sequences in a standardized form; a form that would allow tools to assist in coding and contract enforcement at runtime. Consequently, we propose the usage of so-called synchronization contracts for describing how to coherently utilize a meaningful piece of functionality provided by the server component. They were originally intended to specify how to deal with a server component in parallel and distributed setups.

Almost all the information to generate synchronization contracts is produced but hardly reused. Typically, server component designers comprehensively specify the semantics of the interaction of the server component with its environment by using UML design diagrams [2]. Due to their simplicity, sequence diagrams are most often employed to model the permissible set of sequences of message invocation for plausible usage scenarios of the server component. A set of sequence diagrams thus diagrammatically specifies the synchronization contract. Based on that insight we propose an approach to automatically create synchronization contracts from UML sequence diagrams. This approach utilizes existing technologies and also draws upon existing research work. The technologies are UML, the Metadata Interchange format XMI [3], path expressions [4], and the Web-Service Description Language WSDL [5].

The UML sequence diagrams (discussed in section 4.1) are first transformed to XMI (section 4.2) using some standard tool. The XMI is parsed, using any standard XML parser, to extract relevant information for synchronization contracts. This information is then represented in path expressions, which are some form of process algebra (section 4.3). The path expressions are finally embedded in the WSDL specification (section 4.4) of a Web-service, using some proposed new constructs. Together with the specification of a contract it is also essential that the contract be somehow enforced. Hence section 5 discusses the possible usages of the synchronization contracts so generated. We conclude with some remarks on open issues (section 6). For details on the employed technologies, the user is referred to referenced publications in the relevant sections.

2. RELATED WORK

In the context of this work, the notions of contract as used in the software component community are most relevant [6] [7]. Four types or levels of contracts for software components are identified: basic or syntactic, behavioral, synchronization, and quality-of-service.

Typically, programmers use the likes of Interface Definition Language (IDL) and C++ header files for the specification and usage of syntactic contracts. In some languages (like Eiffel [8] or contract extensions to Java [9]) pre and post conditions, and invariant formalisms are available for specifying and using behavior contracts. In essence, behavioral contracts specify what the client can expect on proper invocation of a specific functionality, i.e., a specific method. Synchronization contracts refer to the definitions that shall specify how a software component can cope with parallelism in distributed setups. Synchronization has been dealt with in various domains in computer science. In [4] Campbell and Habermann discuss path expressions as a means of specifying synchronization of processes in operating systems. This formalism is also suggested in [6] and further elaborated on in [10] to specify synchronization issues in software components. However, we propose to extend synchronization contracts to explicitly specify semantically coherent server usage scenarios, i.e., to define the sets of permissible sequences of message invocations.

The notion of a contract is also found in Web-service descriptions. WSDL specifies in which way Web-services have to publish their features. However, it does not go beyond syntactical conventions and from a functional viewpoint can be compared to current IDLs, which are used in distributed object technologies. The business need for formalisms to describe interaction patterns for Web-services is supported by the fact that standardization attempts like the WSFL [11] try to formalize how business functionality can be realized by collaborating Web-services. It is different to our approach in its intent to describe the overall interaction scenario of several Web-services and not contractually specify a

Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

single server component. Further, it is one more language in addition to WSDL and not a natural extension to WSDL, as is our proposal.

A third area of related work in specifying the interaction behavior of black-box entities, e.g., programming language classes, is based on state diagrams. For instance, the development methods Fusion [12] or Fondue [13] employ state diagrams to depict what they call system life-cycle model and system interface protocol, respectively. The approach differs to ours in that, firstly, the state diagrams specify all possible transitions (method invocations) without giving a hint to semantically meaningful invocation sequences; secondly, the state diagram representation is not represented textually in a contract and consequently not proposed as feasible extension to existing contract technologies.

3. THE RUNNING EXAMPLE

We use a simple running example to illustrate our approach: A bank offers its services to its customers by providing a large set of Web-services to access common functionality like accessing balance information or allowing the clients to transfer money from one account to another.

A part of the Web-services is the account transfer service provided by a bank account transfer server component. For reasons of simplicity, the server component implements just one interface with three methods:

M1: getAuthorization([in]Account, [in]ClientID, [out]TransactionID)
M2: withdraw([in]Account, [in]amount, [in]TransactionID)
M3: deposit([in]Account, [in]amount, [in]TransactionID)

To transfer money from one account to another, the permissible sequence for a single transaction, identified by the transaction identification (TransactionID), is:

M1 -> M2 -> M3.

There is no other permissible sequence to use the account transfer component. Hence, client software shall not try to do so and the component shall not have to check all other eventualities for a possible misuse. But such a synchronization contract, though intended, is not explicitly stated.

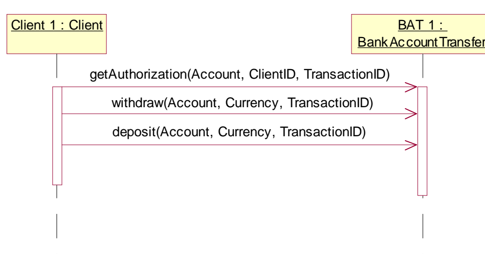
4. AUTOMATIC CREATION OF SYNCHRONIZATION CONTRACTS

4.1. The UML Sequence Diagram

The sequence diagram, shown in Figure 1, models the synchronous message calls on the bank account transfer component. It shows the only permissible sequence of operations or method calls for transferring money from one account to another. All other sequences of method calls, on this component, are not modeled here and would thus be considered a “misuse of the component”. This sequence diagram graphically represents important pieces of the synchronization contract for the bank account interface.

Note that it is assumed that all possible sequences of message calls on a server component are modeled and made available through sequence diagrams. The synchronization contract would otherwise be incomplete, and the contract enforcement entities (discussed in Section 5) would over constrain the usage of the server component.

Figure 1: Sequence diagram for the bank account transfer example



4.2. The XMI Representation

XMI [3] is a XML based OMG standard to textually represent and interchange meta-model information. A XML DTD has been defined therein to represent UML diagrams in XMI.

A UML sequence diagram models the synchronization of message invocations between the participants in a Collaboration. Each sequence diagram represents one possible synchronization pattern of a set of participants. The same participants could appear in multiple diagrams to exhibit other synchronization patterns or interactions amongst them. Hence, all sequence diagrams could be represented in a single or multiple XMI files.

Each participant exhibits a specific Role as an instance of some specific Class. Therefore, before generating the XMI representation of the collaboration or the dynamic model of the software system, the XMI representation of the static model, i.e. the definition of data-types, interfaces and classifiers, is created. The interactions or message sequences are documented by tagging the message entities with constructs defining preceding and following message entities. The actions to be taken on message invocation are also listed to complete the XMI representation of the diagram.

In the following we use our bank account example to exemplify the above. We used [14] to generate the XMI representation from our sequence diagram.

The first step, which consists of the syntactical type conventions of the Client and the BankAccountTransfer server component represented in XMI, is not shown here.

Second, to be able to represent the behavioral model, the roles of the BankAccountTransfer (Figure 2) and the Client are described. The XMI representation for the role lists the ID (UML:ClassifierRole.base) of the base classifier with the available set of features or operations (UML:ClassifierRole.availableFeature) of this role and the utilized set of messages (UML:ClassifierRole.message1).

To represent the interaction, the messages exchanged between the participants are documented to necessary detail. An extract of the document for the ‘withdraw’ message for our BankAccountTransfer sequence diagram is given in Figure 3. The XMI representation of the message includes details of the message sender (the Client), the receiver (the BankAccountTransfer server component), the message that follows the ‘withdraw’ (UML:Message.message3, which is the ‘deposit’), the predecessor message (UML:Message.predecessor, which is the ‘getAuthorization’), and finally the action to be taken on the invocation of the ‘withdraw’ message.

In summary, the XMI representation of the sequence diagram completely specifies the information needed for the synchronization contract. Further, the information can easily be accessed utilizing a standard XML parser.

Figure 2: XMI representation of the BankAccountTransfer role

```
<UML:ClassifierRole xmi.id = ‘clsRlBankAccountTransfer1’ name = ‘BAT 1’>
  <UML:ClassifierRole.base>
<Classifier xmi.idref = ‘clsBankAccountTransfer’/>
  </UML:ClassifierRole.base>

  <UML:ClassifierRole.availableFeature>
<Feature xmi.idref = ‘oprGetAuthorization’/>
<Feature xmi.idref = ‘oprWithdraw’/>
<Feature xmi.idref = ‘oprDeposit’/>
  </UML:ClassifierRole.availableFeature>

  <UML:ClassifierRole.message1>
<Message xmi.idref = ‘msgGetAuthorization’/>
<Message xmi.idref = ‘msgWithdraw’/>
<Message xmi.idref = ‘msgDeposit’/>
  </UML:ClassifierRole.message1>
</UML:ClassifierRole>
```

Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Figure 3: XMI representation of the 'withdraw' message

```
<UML:Message xmi.id = 'msgWithdraw' name =
'withdraw(Account, TransactionID, Currency)'>
   <UML:Message.sender>
<ClassifierRole xmi.idref = 'clsRlClient1'/>
   </UML:Message.sender>

   <UML:Message.receiver>
<ClassifierRole xmi.idref = 'clsRlBankAccountTransfer1'/>
   </UML:Message.receiver>

   <UML:Message.message3>
<Message xmi.idref = 'msgDeposit'/>
   </UML:Message.message3>
   <UML:Message.predecessor>
<Message xmi.idref = 'msgGetAuthorization'/>
   </UML:Message.predecessor>

  <UML:Message.communicationConnection>
<AssociationRole xmi.idref = 'assocRlBAT_Client'/>
  </UML:Message.communicationConnection>

  <UML:Message.action>
<Action xmi.idref = 'callActionWithdraw'/>
  </UML:Message.action>
</UML:Message>
```

4.3. The Algebraic Representation

Based on the original ideas of Campbell and Habermann [4] we suggest including a so-called path expression into the declarative type information of a server component. This construct describes how an instance of that type may be used by one or several client components. While the original authors proposed the method for synchronizing between procedure executions by different processes (with its main application to operating systems), we see the value not only in the integrity protection of a server component with respect to multiple client components but also in the unambiguous description of the permissible sequence of method invocations to obtain a desired functionality that is relevant for one client only.

We briefly present those parts of the formalism that are needed to comprehend the running example. For details, we refer to [4] [15]. For our purpose, a path expression expresses the temporal relationships between the methods identified for a component server. It specifies the sequence (with notation ";"), single selections among alternatives (","), and parallelism of methods ("{}").

Hence, **A;B;C** specifies that these methods must be in sequence starting with A. **A,B** specifies that only A or B may be called. And, **{A}** defines that a number of client components may invoke A in parallel. Combinations of these basic constructs are possible.

We expect to obtain the path expression for a server component by examining the interaction scenario between a client and server represented in XMI. With respect to our running example, we need to consider the type information for the BankAccountTransfer server component by parsing the XMI for the static classifier type information (<Classifier xmi.idref = 'clsBankAccountTransfer'/>) and its defined messages (<UML:ClassifierRole.message1>) in the context of one specific client component (or sender as it is called in the XMI <UML:Message.sender> tag). This lets us construct the expression below as the one described permissible scenario:

```
msgGetAuthorization; msgWithdraw; msgDeposit;
```

4.4. The WSDL Representation

The Web Service Description Language (WSDL [5]) is an XML format for the abstract description of services ("abstract endpoints" as they are called in the standard) in the form of operations and messages.

Figure 4: Extended WSDL representation

```
<?xml version="1.0"?>
<definitions name="BankTransfer"
...
   <types>
     ...
   </types>
...
    <portType name="BankAccountTransfer">
      <operation name="GetAuthorization">
         ...
      <operation name="Withdraw">
         ...
      <operation name="Deposit">
         ...
      <path name="anyPath">
<expression="GetAuthorization;
   Withdraw;
  Deposit">
        </expression>
      </path>
   </portType>
...
    <service name="BankAccountTransfer">
      <documentation>Service to transfer
        ...
      </documentation>
       <port name=" BankAccountTransfer ">
      </port>
   </service>

</definitions>
```

The abstract descriptions are then bound to a concrete network protocol and message format (e.g. SOAP on http).

In general, WSDL consists of five major parts: type descriptions of the exchanged data (enclosed in the <types> tag block), the port type definitions (enclosed in the <portType> tag block), the service definition (enclosed in the <service> tag block), and the messages and bindings (not shown here). A service can have a number of service ports and a service port a number of operations.

For our purpose of expressing permissible sequences of operation invocations, we propose a new port type related construct enclosed in a <path> </path> tag pair, which shall hold the path expression related information. The construct extends the "port" definitions of a service. Figure 4 depicts the relevant part of a WSDL description for our running example. The portType with name "BankAccountTransfer" has been extended with a path definition called "anyPath". This path expression is valid for accessing the service "BankAccountTransfer", which conforms to the port type definitions of the same name.

It should be noted that although the example is trivial, any form of path expression could of course be represented in that way. The path expression formalism not only allows capturing sequences, alternatives, etc. but also supports nesting.

5. USAGE OF SYNCHRONIZATION CONTRACTS

If we assume the existence and accessibility of a synchronization contract as discussed in the previous sections, development environments can make use of it and provide tool automation support at client and server design time. Further, contracts may also serve as the basis for guard functionality to protect the server at runtime from unwanted interaction sequences.

5.1. Client Design Time Support

Since the synchronization contracts are machine process-able, development environments can raise their level of automation. For devel-

Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

oping client-side software we envision three levels of contract-aware coding support:

(a) *Provision of code completion facilities*; similar to today's code completion facilities, where the source code editors provide context sensitive drop down lists with attribute or method options that could follow the chosen variable name, synchronization contracts may further limit the options by presenting only those methods that would still be in proper sequencing.

(b) *Provision of basic code skeletons*; source code editors could be enhanced to provide entire method sequence skeletons that can be selected by the programmer from the choice of possible paths. If the paths in the synchronization contract were even augmented with textual identification of use cases or collaborations, programmers could choose among these "high level" code skeleton options.

(c) *Provision of compile time verification*; compilers can be enhanced to verify client source code for proper sequences of method calls to server objects.

Note, similar kinds of tool-based automation are of course also conceivable at modeling time. E.g., the path information can also be used to support the construction of UML object diagrams, collaboration diagrams, and sequence diagrams in which existing Web-services or components are part of.

5.1. Server Design- and Runtime Support

The gain at server design time lies in relieving the programmer from taking counter measures to deal with unwanted sequences of client calls. Further, guard construction (see below) can be automated.

The runtime support at server side is grounded in the possibility to provide specific guards that verify contract adherence at runtime, i.e., to make sure that only permissible sequences of client calls are dispatched to the server and an appropriate exception handling is activated otherwise. In general, some form of interception mechanism must assure the execution of guard functionality.

Since path expressions can be transformed into state diagrams quite easily, it is also possible to automate the generation of guard code. A possible algorithm is presented in [4]. There are a number of feasible design options for guard implementations in current distributed technologies. For instance, an implementation can be in the form of policy objects as defined in COM+ [16], or it can be an extension to the current proxy and stub constructs in Microsoft COM or stub and skeleton in OMG's CORBA. To have more fine grained control, it is of course also conceivable that guards are implemented by the programmer as part of the Web-service or software component, e.g. in the form of façade objects [17].

6. CONCLUSIONS

Most of today's software is built by composing components and services, and it appears that future will see even more development in this fashion. The components are available as black box entities, the precise specification of the permissible usage of which is documented in software contracts. For software assemblies to work as intended and to aid the software development process, it is required that contracts be comprehensively specified and made available in easy to interpret, machine-readable form. This is important to save effort of the designers and the developers by automating the process of specifying and implementing contracts, from the design stage itself.

We proposed a way of specifying synchronization contracts, in addition to the existing syntactic and behavioral contracts. The contract is specified in a simple and precise form without having the developer go through hardly-ever-adopted formalisms. Existing technologies have been utilized not only for the specification but also for showing the feasibility of a process for automatically generating synchronization contracts. Based on our general observation, the process assumes that the designers employ UML sequence diagrams for specifying the synchronization characteristics of the server components. However, the developers may choose to utilize other modeling aids like state diagrams. Though our approach will not be directly applicable in that case, the basic process of extracting information from a UML diagram in XMI and then mapping it over to path expressions can still be utilized. The utilization of sequence diagrams, as they are currently defined in UML 1.3, only supports our limited usage of synchronization contracts, i.e. parallelism or repetition is not explicitly taken care of. UML extensions, partly proposed in real-time communities, or agreed upon annotations of sequence diagrams could overcome these deficiencies. Another potential shortcoming in our approach is to over-constrain a server component. This happens if the server designer makes an incomplete design, i.e. if she chooses to model only one (or a few) out of many possible interactions to obtain the same functionality. This could then prevent the client from using an invocation sequence that would not have harmed the server, but it would also not prevent the client from obtaining the sought after functionality.

Having processes and tools in place, which automatically generate the information on how to properly collaborate with a server component to obtain a coherent piece of functionality, would more explicitly support the goals of (re-)users and significantly ease development of complex software.

REFERENCES

[1] C. Szyperski, *Component Software - Beyond Object-Oriented Programming*. Reading, Massachusetts: Addison-Wesley, 1998.

[2] OMG, "Unified Modeling Language V1.3," , Specification, June 1999.

[3] OMG-XMI-RTF, "XML Metadata Interchange (XMI) Version 1.1," Object Management Group OMG Document ad/99-10-02, October 25 1999.

[4] R. H. Campbell and A. N. Habermann, "The Specification of Process Synchronization by Path Expression," in *Proc. International Symposium on Operating Systems*, 1973, pp. 89-102.

[5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," Ariba, IBM, Microsoft, Submission to W3C Note, March 15 2001.

[6] A. Beugnard, J.-M. Jezeguel, N. Plouzeau, and D. Watkins, "Making Components Contract Aware," *IEEE Computer,* vol. 32, pp. 38-45, July 1999.

[7] QCCS. (2002, Jan). Quality Controlled Component-Based Software Development. European Community IST Project-1999-20122 [Online]. Available: http://www.qccs.org/

[8] B. Meyer, *Eiffel: The Language*, 2 ed. London: Prentice Hall, 1992.

[9] M. Wiedmann, H. Buchwald, and D. Seese, "Design by Contract in Java - a Roadmap to Excellence in Trusted Components," *INFORMATIK*, pp. 9-14, 2000.

[10] D. Watkins, "Using Interface Definition Languages to Support Path Expressions and Programming by Contract," in *Proc. Technology of Object-Oriented Languages and Systems (TOOLS 26)*, 1998, pp. 308-319.

[11] F. Leyman, "Web Services Flow Language (WSFL 1.0)," IBM Software Group, May 2001.

[12] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes, *Object-Oriented Method: The Fusion Method*. London: Prentice Hall, 1994.

[13] S. Sendall and A. Strohmeier, "Specifying Concurrent System Behavior and Timing Constraints Using OCL and UML," in *Proc. UML 2001 - The Unified Modeling Language: Modeling Languages, Concepts and Tools*, 2001, pp. 391-405.

[14] Unysis. (2002, March). Unysis XML Tools for Rose. Unysis Corporation [Online]. Available: http://www.rational.com/support/downloadcenter/addins/rose/index.jsp

[15] O. Rees, "Using path expressions as concurrency guards," ANSA, Cambridge, Technical report TR.022.00, February 24 1993.

[16] D. S. Platt, *Understanding COM+*. Redmond, WA: Microsoft Press, 1999.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison-Wesley, 1995.

Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

0 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/proceeding-paper/generating-synchronization-contracts-web-services/32084

Related Content

Application and Research of Interactive Design in the Creative Expression Process of Public Space
Yuelan Xu (2022). *International Journal of Information Technologies and Systems Approach (pp. 1-13).*
www.irma-international.org/article/application-and-research-of-interactive-design-in-the-creative-expression-process-of-public-space/307028

Classification of Polarity of Opinions Using Unsupervised Approach in Tourism Domain
Mahima Goyaland Vishal Bhatnagar (2016). *International Journal of Rough Sets and Data Analysis (pp. 68-78).*
www.irma-international.org/article/classification-of-polarity-of-opinions-using-unsupervised-approach-in-tourism-domain/163104

Teaching Media and Information Literacy in the 21st Century
Sarah Gretterand Aman Yadav (2018). *Encyclopedia of Information Science and Technology, Fourth Edition (pp. 2292-2302).*
www.irma-international.org/chapter/teaching-media-and-information-literacy-in-the-21st-century/183941

Toward a Theory of IT-Enabled Customer Service Systems
Tsz-Wai Luiand Gabriele Piccoli (2009). *Handbook of Research on Contemporary Theoretical Models in Information Systems (pp. 364-383).*
www.irma-international.org/chapter/toward-theory-enabled-customer-service/35841

Scholarly Identity in an Increasingly Open and Digitally Connected World
Olga Belikovand Royce M. Kimmons (2018). *Encyclopedia of Information Science and Technology, Fourth Edition (pp. 6779-6787).*
www.irma-international.org/chapter/scholarly-identity-in-an-increasingly-open-and-digitally-connected-world/184373