A Model-Prover for Constrained Dynamic Conversations

Diletta Cacciagrano  
University of Camerino  
Camerino, Italy  
diletta.cacciagrano@unicam.it

Flavio Corradini  
University of Camerino  
Camerino, Italy  
flavio.corradini@unicam.it

Rosario Culmone  
University of Camerino  
Camerino, Italy  
rosario.culmone@unicam.it

Luca Tesei  
University of Camerino  
Camerino, Italy  
luca.tesei@unicam.it

Leonardo Vito  
University of Camerino  
Camerino, Italy  
leonardo.vito@unicam.it

ABSTRACT

In a service-oriented architecture, systems communicate by exchanging messages. In this work, we propose a formal model based on OCL-constrained UML Class diagrams and a methodology based on Alloy Analyzer respectively for describing and verifying any first-order constrained client-server conversations. This framework allows us to verify conversation protocol designs at a fairly detailed level and to check first-order logic constraints on both message flows and message contents.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Design tools and techniques; D.2.1 [Software Engineering]: Methodologies; D.2.2 [Software Engineering]: Software/Program verification, Formal methods, Model checking, Validation

Keywords

Web Service, Conversations, WSDL, UML, OCL, Alloy

1. INTRODUCTION

The recent trend in Web Services is fostering a scenario where clients perform run time queries in search of services, services provide some given capabilities, and both systems communicate by exchanging messages. Message passing is a mechanism for robust and loosely coupled interactions which, differently from traditional RPC models, is not based on a fairly rigid request-response interaction style. The set of related messages exchanged by multiple interacting parties is called conversation; in particular, a client-server conversation is a special case where only two interacting parties are involved. The Web Services Description Language (WSDL) [11] is the standard used for publishing abstract and concrete descriptions of Web Services - including the schemas of exchanged messages, the name and type of operations that the service exposes and some simple interaction patterns. On the other hand, there are a multitude of specifications for describing conversation rules - [2], [9], [8] and [10] are few examples - each of them defining a structured language expressing (temporal, priority, etc.) relationships between the exchanged messages.

Different models have been defined in order to specify and verify the behavior of a service in terms of flow of exchanged messages\(^1\). For example, in [18] mediated composite services specified in BPEL are verified against the design specified using Message Sequence Chart and Finite State Process notations, while in [16, 19] finite automata are augmented with XML messages, XPath [12] expressions and boolean conditions, in order to verify temporal properties of the conversations of single and composite Web Services\(^2\).

2. FRAMEWORK AND METHODOLOGY

In this scenario, we propose a formal model for describing and verifying any first-order constrained client-server conversation. The model is independent from the conversation language: we only assume a generic XML-based document describing conversations and WSDL [11] describing message schemas\(^3\). As in [17], constraints are CLiX [3] rules (i) abiding/disabling message transitions in the conversation flow and (ii) well-typing the involved messages. The verification procedure relies on Alloy [1], an object-oriented, first-order logic-based language.

\(^1\)In terms of flow of exchanged messages, the behavior of a service describes the changes of its states; message-, activity- and event-based specifications are well-known formal models, relying on different kinds of actions to change state.

\(^2\)The verification framework proposed in [16, 19] is based on SPIN [22] and inputs BPEL specifications of Web Services translated into PROMELA, a boolean-logic based language: for this reasons, it can only achieve partial verifications by fixing the sizes of the input queues in the translation, and complete verifications only under stronger conditions.

\(^3\)The proposed framework also fits on a scenario in which message templates are described by XML schemas [7].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

iiWAS2008, November 24 - 26, 2008, Linz, Austria

Copyright 2008 ACM 978-1-60558-349-5/08/0011 ...$5.00.
order logic-based modeling language, equipped with an analyzer providing a unique hybrid of features associated with theorem provers as well as model checkers. At this aim, the fully XML-based model (i.e., conversation, WSDL and CLiX documents) is encoded into an OCL-constrained UML Class Diagram, making possible by Alloy (i) to verify the conversation design at a fairly detailed level, both on message flow and on message contents, and (ii) to check constraint configurations, both generic (consistency constraints) and specific (customized to the conversation). Modeling a conversation as an OCL-constrained UML Class Diagram has an interesting consequence: it is possible to build an incremental verification procedure in Alloy, testing the diagram initially equipped with only one constraint - if there exists - then enriching the previous diagram with one more constraint only after a successful verification result, and so on. It follows that the global verification procedure is partitioned in local steps, since the successful/unsuccessful result of a phase is associated to a well-known constraint. In the following, we explain in detail the main key assumptions.

**Linking conversation document and WSDL:** We denote by \( W_c \) the generic (XML-based) conversation document, by \( W_m \) the WSDL document containing the templates of any exchanged message, and by \( G \) the set of CLiX rules constraining \( W_c \) and \( W_m \) XML-elements. Then, we state a relationship, called stability, between \( W_c \) and \( W_m \), as follows: for each operation \( o \) in \( W_m \), the schema associated to each input, resp. output/fault, operation element \( p \) in \( o \) is the schema of an inbound, resp. outbound, message type \( m \) in \( W_c \). Side effects of this assumption are the following: (i) the scope of any rule in \( G \) only involves \( W_m \) operation element schemas, and (ii) there is a syntactic match between message XML-identifiers in \( W_c \) and operation element XML-identifiers in \( W_m \), making possible to rightly encode CLiX rules into OCL constraints (and vice versa)\(^4\).

**First-order Guarded automata and UML Class Diagrams:** In \([16, 19]\) it has been proved that any conversation can be modeled as a boolean guarded automaton. Our framework is based on an extension of this model, obtained imposing CLiX\(^5\) rules as first-order logic guards. A conversation is modeled by a so-called Constraint Diagram, i.e. a UML Class Diagram equipped with OCL \([4]\) constraints. Intuitively, a Constraint Diagram is an UML specification of the guarded automaton associated to a conversation: (i) each class represents a message type, (ii) two classes \( m_1 \) and \( m_2 \) are related if there is a state \( q \) and two transitions, respectively labeled with \( m_1 \) and incoming to \( m_1 \) and labeled with \( m_2 \) and outgoing to \( q \), and (iii) OCL constraints correspond to CLiX guards.

Formally, a Constraint Diagram is an UML specification of a conversation on its own, i.e. without reasoning in terms of guarded automata: to model a - both existing and novel - conversation by a Constraint Diagram, it suffices to define classes, associations and OCL constraints in such a way that (i) each class models a message type, (ii) associations among classes correspond to interactions involving message types associated to those classes, and (iii) constraints are OCL formulas on class attributes, those classes being associated to messages to constrain. The main reasons of replacing a guarded automata-based model by an OCL-constrained Class Diagram consist of the following points: (i) differently from other UML models, a Class Diagram is suitable for describing and for designing respectively existent and new conversation protocols; (ii) it is a well-known UML diagram which can be annotated by OCL expressions; (iii) it is suitable to be verified by Alloy; (iv) it looks as a suitable specification where automatically importing - in the form of templates - OCL constraints expressing consistency properties, i.e invariant for any conversation; (v) it can express properties which first-order logic, UML without OCL and OCL itself cannot. To better explain the last point, it suffices to consider the transitive closure property: it is well-known that it cannot be expressed in first-order logic, and also that both UML and OCL have no transitive closure operator. However, UML equipped with OCL constraints attempts to axiomatize the transitive closure operator. As a consequence, it is possible to express a simple property stating that “any defined message type has to be useful” - i.e. it is used in at least one conversation - just introducing in \( C_{\text{dy}} \) an empty class \( \text{Start} \) representing an empty message, for every “initial class” - i.e. associated to an initial message - I an association from \( \text{Start} \) to \( I \), and, for every class \( X \), an OCL constraint of the following form: context \( X \)

\[
\text{def: } \text{tr\_closure} : \text{Set}(\text{Message}) = \\
\quad \text{self\_next\_union}(\text{self\_next}\_\text{collect}(e | e\_tr\_closure)) \quad \text{inv: } \text{self} \in \text{tr\_closure}(\text{Start})
\]

**Formalizing UML and OCL in Alloy:** Formalizing UML and OCL for the purpose of analysis and verification is a well-known topic: consider the use of B \([24]\), a formalization of OCL in Isabelle/HOL \([15]\), syntactic analyzers \([5]\), simulators \([6]\), compilers enabling run-time checking of specifications \([23]\), model checkers \([20]\) and integrations with theorem provers \([14]\), the USE tool \([25]\) implementing an interpreter of OCL for run-time checking. In the framework we propose here, the translation of UML into Alloy is fully automatic\(^6\) thanks to UML2Alloy \([13]\), a filter tool formatting UML Class Diagrams enriched with OCL constraints as Alloy specifications. The current version of UML2Alloy performs the translation creating a text file with the Alloy model; the designer, which knows UML and OCL but maybe does not have any notion about Alloy language syntax, only needs to use the Alloy Analyzer to open the text file and perform the analysis.

### 3. A MODEL FOR VALID FIRST-ORDER CONSTRAINED CONVERSATIONS

In this section, we formally define a model for valid first-order constrained client-server conversations, where valid is intended w.r.t. a set of CLiX rules.

**Notation 1.** We denote by \( W_c \) the generic (XML-based) document describing a client-server conversation; by \( W_m \) the generic XML-based document containing the templates of any \( W_c \) conversation message, and by \( G \) the set of CLiX rules constraining \( W_c \) and \( W_m \) (message) XML elements; by \( M = \{ m_k | k \in [1..n], n \geq 1 \} \) the finite set of message types involved in \( W_c \) and described in \( W_m \); by \( M_i \) and \( M_o \) the

\(^4\)It is well-known that both OCL and CLiX support first-order logic, and that OCL can be encoded into CLiX.

\(^5\)CLiX is a logical language, used both to constrain XML documents internally and to execute inter-document checks. It allows constraints to be described using a mixture of first-order logic and XPath expressions.

\(^6\)Differently from \([21]\), where the translation is manual.
finite sets of respectively inbound and outbound message types in $M = M_i \cup M_o$; by $x(d)$ the XML schema describing an element $d$.

First, we abstract from the tuple $(W_i, W_o, G)$, replacing it with its guarded automaton-based representation.

**Definition 1.** The First-Order guarded (FOG) automaton associated to $(W_i, W_o, G)$ is the tuple $A = (S, M, V, s, f, \delta, G)$, where:

i. $S$ is a finite set of states;
ii. $M = M_i \cup M_o$ is as above described;
iii. $V = \{v_1, ..., v_M\}$ is a vector of XML local variables, where $\forall v_j \in \{v_1, ..., v_M\}$, $v_j$ is associated to $m_j \in M$;
iv. $s \in S$ is the initial state and $f \in S$ is the final state;

$v. \; G = \{(q, g(q), \delta(q), Q) | g(q), \delta(q) \in CLIX\}$ where $q \in S$, $m_k \in M$ and $v_j \in \{v_1, ..., v_M\}$, $d_j \in \{d(v_j(q)), \lambda\}$.

vi. $\delta = \{(q, (l, g(q)), Q) \in S \times \{X, Y\} \times \{\{m_i\} \cup \{m_o\} | m_i \in M_i\}$ and $g(q) \in G$.

Message types and local variables are XML documents. Each local variable $v_j$ in $V$ corresponds to a message type $m_j$ in $M$. $\forall v_j \in S$ and $\forall v_j \in \{v_1, ..., v_M\}$, $d(v_j(q))$ denotes the XML document obtained enumerating all the sent/received (until the state $q$) message instances that match to the type $m_j$. Each transition $\delta \in \delta$ is in one of the following two forms:

- **(receive-transition)** $\tau = (q_1, (m_k, g(q_1)), Q)$, where $m_k \in M_i$: the transition nondeterministically changes the state of the automaton from $q_1$ to $q_2 \in Q$, it appends the received message instance (of type $m_k$) from the input queue and it updates $v_k$ in $V$, corresponding to $m_k$, by the concatenation of the received instance, in the case $\delta(q_1)$ holds;

- **(send-transition)** $\tau = (q_1, (\{m_k\}, g(q_1)), Q)$, where $m_k \in M_o$: the transition nondeterministically changes the state of the automaton from $q_1$ to $q_2 \in Q$, it appends the sent message instance (of type $m_k$) to the input queue of the client and it updates $v_k$ in $V$, corresponding to $m_k$, by the concatenation of the sent instance, in the case $\delta(q_1)$ holds.

**Definition 2.** Let $A = (S, M, V, s, f, \delta, G)$ be the FOG automaton associated to $(W_i, W_o, G)$. Given a guard $g(q) = g(m_k, (d_1, ..., d_M)) \in \delta$, then:

i. $(d_1, ..., d_M)$ denotes the actual context of $g(q)$, obtained filtering out all the local variables such that no XML attribute of theirs is involved in $g(q)$.

ii. $X(g(q))$ denotes the formal context of $g(q)$, obtained by an XML-schema concatenation of those local variable included in the actual context of $g(q)$, i.e. $X(g(q)) = \bigcup_{(d_1, ..., d_M)} x(m_k)$, where $x(m_k) = \lambda$ if $d_i = \lambda$.

**Notation 2.** Let $W_m$ be a WSDL document. We denote by $O_m$ the set of operation in $W_m$, for every $o \in O_m$, by $p_{in}(o)$ and $p_{out}(o)$ respectively the input and the output/fault operation elements of $o$.

We also assume that $W_i$ and $W_o$ are related as follows: for each operation $o \in O_m$, for every $p_k \in p_{in}(o)$ (resp. $p_{out}(o)$), for every $m_k \in M_i$ (resp. $M_o$), $x(m_k) = x(p_k)$.

We formally define this kind of relationship between $W_i$ and $W_o$ as follows.

**Definition 3.** Let $A = (S, M, V, s, f, \delta, G)$ be the FOG automaton associated to $(W_i, W_o, G)$, and let $W_m$ be a WSDL document. $W = (A, W_m)$ is stable if and only if $\forall q_1 \in S$ such that $(q_1, (m_k_1, g(q_1)), Q_1) \in \delta$:

i. $\exists o \in O_m$ such that $p_{in}(o) = \{p_{k_1}\}$ and $x(p_{k_1}) = x(m_k_1)$;
ii. $q_2 \in Q_i$, $\exists h(2 \leq h \leq 3)$ s.t. $(q_2, (m_k_2, g(q_2)), Q_2) \in \delta$ iff $p_{out}(o) = \{m_k_2\}$ $2 \leq h \leq 3$ and $x(p_{k_2}) = x(m_k_2)$.

The stability assumption (Definition 3) implies that it is possible to use everywhere the identifier $m_k$ of message type in place of the identifier $p_k$ of operation element, and that both formal and actual contexts of any guard in $G$ only involves $W_o$ operation element schemas.

Given $W = (A, W_m)$ stable, we can build a Class Diagram equipped by OCL constraints, called Constraint Diagram and denoted by $Cd_W$, semantically equivalent to $W$, where (i) each class corresponds to a message type, (ii) class associations correspond to state transitions, and (iii) OCL constraints in $Cd_W$ correspond to CLIX guards in $G$. The correspondence between $W = (A, W_m)$ stable and $Cd_W$ is formally defined as follows.

**Definition 4.** Given $W = (A, W_m)$ stable, the Constraint Diagram $Cd_W$ associated to $W$ is a Class Diagram obtained translating $W$ by the encoding $\{\} \; \text{so defined:}$

i. $\forall o \in O_m$ such that $o \in \{\}$

$<\text{operation name="m_k1"} />$

$<\text{input name="m_k1" message="tns:m_k1-Document"/>}$

$<\text{output name="m_k2" message="tns:m_k2-Document"/>}$

$<\text{fault name="m_k3" message="tns:m_k3-Document"/>}$

$\forall m_k \in \{p_{in}(o) \cup p_{out}(o)\} (1 \leq k \leq 3)$ such that $m_k \in \{\}$

$<\text{element name="m_ky"/>}$

$<\text{xsd-m_ky:element name="m_ky-Element"/>}$

and such that $\exists t_k$ such that $t_k \in \{\}$

$<\text{xsd-m_ky:element name="m_ky-Element"/>}$

then $[m_k]_W = c(m_k)$, where $c(m_k)$ is the class of $t_k$.

ii. $\forall o \in O_m: p_{in}(o) = \{m_k\}$ and $p_{out}(o) = \{m_k\}$ $\exists h(2 \leq h \leq 3)$, there exists an association between $c(m_k)$ and $c(m_k)$:

$\forall m_k, m_k_\bot \in O_m: p_{in}(o) = \{m_k\}$ and $p_{out}(o) = \{m_k\}$, there exists an association from $c(m_k)$ to $c(m_k)$ if and only if $\exists q_1, q_2 \in Q_s$ such that $(q_1, (m_k, g(q_1)), Q_1), (q_2, (m_k, g(q_2)), Q_2) \in \delta$ and $q_2 \in Q_i$;

$\forall m_k, m_k_\bot \in O_m: p_{out}(o) = \{m_k\}$ and $p_{in}(o) = \{m_k\}$, there exists an association from $c(m_k)$ to $c(m_k)$ if and only if $\exists q_1, q_2 \in Q_s$ such that $(q_1, (m_k, g(q_1)), Q_1), (q_2, (m_k, g(q_2)), Q_2) \in \delta$ and $q_2 \in Q_i$;

$\forall m_k, m_k_\bot \in O_m: p_{in}(o) = \{m_k\}$ and $p_{out}(o) = \{m_k\}$, there exists an association from $c(m_k)$ to $c(m_k)$ if and only if $\exists q_1, q_2 \in Q_s$ such that $(q_1, (m_k, g(q_1)), Q_1), (q_2, (m_k, g(q_2)), Q_2) \in \delta$ and $q_2 \in Q_i$;

$\forall m_k, m_k_\bot \in O_m: p_{out}(o) = \{m_k\}$ and $p_{in}(o) = \{m_k\}$, there exists an association from $c(m_k)$ to $c(m_k)$ if and only if $\exists q_1, q_2 \in Q_s$ such that $(q_1, (m_k, g(q_1)), Q_1), (q_2, (m_k, g(q_2)), Q_2) \in \delta$ and $q_2 \in Q_i$.

For every guard $g(q) = g(m_k, (d_1, ..., d_M)) \in \delta$, then $g(q) = \text{context} \bigcup_{m_k \in \{m_j\}} \text{inv} : g$, where $x(m_k)$ in $X(g(q))$ and $g$ is a formula semantically equivalent to $g(q)$.

Notice that it would be also possible to start with the design of a novel conversation in the form of Constraint Diagram, building from it a stable pair of XML documents.

**Example 1.** Suppose to project a toy authentication service defining the following scenario: (i) the client is required either to register by a Registration form, or to login by a Login form; (ii) after filling a Registration form, the client can only access to a Login one; (iii) after filling a Login form, the client is allowed to enter the system only if either it has already registered in a past session and login username is
Rent session and login username is valid; (iv) the allowed max number of failed logins is 3. In terms of WSDL documenting

\text{RegistrationRQ} \text{valid}, or he has just filled a \text{Registration}

\text{InvalidLoginRS} \text{as inbound element,} \text{ValidLoginRS} \text{and} \text{RegistrationRS} \text{as outbound elements, and a} \text{LoginRQ} \text{operation, including} \text{RegistrationRQ} \text{and} \text{RegistrationRS} \text{respectively as inbound and outbound elements. Fig. 1 shows the Constraint Diagram \text{Cdw}, associated to the protocol above described, which has to be input into UML2Alloy. The attributes of a class correspond to the WSDL attributes of the message described by the class itself. LoginRQ.allInstances denotes the set of LoginRQ instances, and LoginRQ.allInstances->count(InvalidLoginRS) denotes the number of LoginRQ instances associated with InvalidLoginRS ones.

4. REFERENCES
