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]: Client/server; D.2.1 [Software Engineering]: Methodologies; D.2.2 [Software Engineering]: Design tools and techniques; D.2.4 [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 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 Service Description Language (WSDL) [10] 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 conversations - [2], [8], [7] and [9] 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

The proposed framework also fits on a scenario in which mediated composite services are described by WSDL document describing message schemas and CLiX

1The verification framework proposed in [15, 18] is based on SPIN [21] 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.

2The proposed framework also fits on a scenario in which message templates are described by XML schemas [6].

3CLiX 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 constraints on both message flows and message contents.

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 specification language: we only assume to handle a generic XML-based document describing conversations, a WSDL document describing message schemas and CLiX
specifying in XML first-order logical constraints on message templates and transitions, as already proposed in [16]. The verification procedure relies on Alloy [1], an object-oriented, first-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 [3]-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 assumptions.

Linking conversation document and WSDL: We denote by $W_c$ a generic (XML-based) conversation document, by $W_m$ a WSDL document containing the templates of any exchanged message, and by $G$ a 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 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 ones: (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 formulas (and vice versa).

Guarded automata and UML Class Diagrams: In [15, 18] 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 first-order logic guards and expressing them as CLiX rules. A conversation is modeled by a so-called Constraint Diagram, i.e. a UML Class Diagram equipped with OCL formulas. Intuitively, a Constraint Diagram is an UML specification of the guarded automaton associated to a conversation, where: (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 either $m_1 \rightarrow m_2$ and incoming to $q$ and labeled with either $m_2 \rightarrow m_1$ and outgoing to $q$, and (iii) OCL formulas correspond to CLiX guards.

Substantially, 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 formulas in such a way that (i) each class models a message type, (ii) associations among classes correspond to interactions involving message order logic and XPath expressions.

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 $W_c$ a generic (XML-based) document describing a client-server conversation; by $W_m$ a generic XML-based document containing the templates of any $W_c$ conversation message, and by $G$ a 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 $M_o$ types associated to these classes, and (iii) constraints are OCL formulas on class attributes. The advantages of an OCL-constrained Class Diagram w.r.t. a guarded automaton are the following ones: (i) being a UML diagram - and differently from the other UML ones - a Class Diagram is suitable for describing and for designing respectively existent and novel conversation protocols with constraints, since it is a well-known UML diagram which can be annotated by OCL formulas; (ii) it is suitable to be verified by Alloy; (iii) it looks as a suitable specification where automatically importing - in the form of templates - OCL formulas expressing consistency properties, i.e invariant for any conversation; (iv) 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 formulas 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 trace - just introducing in a Constraint Diagram (i) an empty class Start representing an empty message, (ii) for every class $I$, associated to an initial message, an association from Start to I and (iii) for every class $C$ the following OCL formula:

$$context C$$

$$def: tr\_closure = Set(Message) = self.next->union(self.next->collect(c | c.\_tr\_closure))$$

$$inv: self in tr\_closure(Start)$$

**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 [23], a formalization of OCL in Isabelle/HOL [14], syntactic analyzers [4], simulators [5], compilers enabling run-time checking of specifications [22], model checkers [19] and integrations with theorem provers [13], the USE tool [24] implementing an interpreter of OCL for run-time checking. In the framework we propose here, the translation of UML into Alloy is fully automatic thanks to UML2Alloy [12], a filter tool formatting UML Class Diagrams enriched with OCL formulas as Alloy specifications. The current version of UML2Alloy performs the translation creating a text file; 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.

*It is well-known that both OCL and CLiX support first-order logic, and that OCL can be encoded into CLiX.

*Different from [20], where the translation is manual.
the finite sets of respectively inbound and outbound message types in $M = M_I \cup M_o$; by $x(d)$ the $d$'s XML schema.

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

**Definition 1. A First-Order guarded (FOG) automaton associated to $(W_e, W_m, G)$ is $A = (S, M, V, q_0, q_n, \delta, G)$, where:**
 i. $S = \{q_t \mid t \in [0..n], n \in \mathbb{N}\}$ 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 $\forall j \in [1..|M|]$, $v_j$ is associated to $m_j \in M$;
 iv. $q_0 \in S$ is the initial state and $q_n \in S$ is the final state;
 v. $G = \{g(i,k) = g(q_i, m_k, d(i,k), \delta(i,k))\}$ CLIx rule, such that $q_i \in S, m_k \in M$ and $\forall j \in [1..|M|]$, $d_j \in d(i,j)$.

\[ \delta = \{(q_i, (l,g(i,k), Q_i))\} \] is a state transition relation, where $q_i \in S, Q_i \subseteq S, l \in \{m_k | m_k \in M\} \cup \{m_i | m_i \in M\}$ and $g(i,k) \in G$.

Message types and local variables are XML documents. Each local variable $v_j$ in $V$ corresponds to a message type $m_j$ in $M$. $\forall q_i \in S$ and $\forall j \in [1..|M|]$, $d_j(i)$ denotes the XML document obtained enqueuing all the sent/received (until the state $q_i$) message instances that match to the type $m_j$.

Each transition $\tau \in \delta$ is in one of the following two forms:

(1) Receive-transition $\tau = (q_i, (m_k, g(i,k), Q_i))$, where $m_k \in M_I$; the transition nondeterministically changes the state of the automaton from $q_i$ to $q_k$, returning 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 $g(i,k)$ holds;

(2) Send-transition $\tau = (q_i, (m_i, g(i,k), Q_i))$, where $m_i \in M_o$: the transition nondeterministically changes the state of the automaton from $q_i$ to $q_k$, it appends the sent message instance (of type $m_i$) 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 $g(i,k)$ holds.

**Definition 2. Let $A = (S, M, V, q_0, q_n, \delta, G)$ be a FOG automaton associated to $(W_e, W_m, G)$. Given a guard $g(i,k) = g(q_i, m_k, d(i,k)) \in G$, then:**
 i. $(d_1, ..., d_{|M|})$ denotes the actual context in $q_i \in S$ of $g(i,k)$, obtained filtering out all the local variables such that no XML attribute of theirs is involved in $g(i,k)$;
 ii. $X(g(i,k))$ denotes the formal context in $q_i \in S$ of $g(i,k)$, obtained as the XML-schema concatenation of those local variables included in the $g(i,k)$ actual context, i.e.

\[ X(g(i,k)) = \bigcup_{x \in [1..|M|]} x(m_x) \] where $x(m_i) = \lambda$ if $d_j = \lambda$.

**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 $p_\text{in}(o)$ and $p_\text{out}(o)$ respectively the input and the output/fault operation elements of $o$. We also assume that $W_e$ and $W_m$ are related as follows: for each operation $o \in O_m$, for every $p_k \in p_\text{in}(o)$ (resp. $p_\text{out}(o)$), for every $m_k \in M_o$ (resp. $M_I$), $x(m_k) = x(p_k)$ holds. We formally define this kind of relationship between $W_e$ and $W_m$ as follows.

**Definition 3. Let $A = (S, M, V, q_0, q_n, \delta, G)$ be a FOG automaton associated to $(W_e, W_m, G)$, and let $W_m$ be a WSDL document. $W = (A, W_m)$ is stable if and only if $\forall q_i \in S$ such that $(q_i, (m_k, g(i,k)), Q_i) \in \delta$.

i. $\exists o \in O_m$ such that $p_\text{in}(o) = \{p_k\}$ and $x(p_k) = x(m_k)$;

ii. $\exists q_i \in Q_{11}, \exists h (2 \leq h \leq 3)$ s.t. $(q_i, (m_k, g(i,k)), Q_i) \in \delta$ if $p_\text{out}(o) = \{m_k\} | \ 2 \leq h \leq 3 \}$ and $x(p_k) = x(m_k)$.

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

Given $W = (A, W_m)$ stable, we can build a Class Diagram equipped by OCL formulas, called Constraint Diagram and denoted by $\mathcal{C}_W$, semantically equivalent to $W$, where (i) each class corresponds to a message type (i.e. a WSDL operation parameter), (ii) class associations correspond to state transitions, and (iii) OCL formulas in $\mathcal{C}_W$ correspond to CLIx guards in $G$. The correspondence between $W = (A, W_m)$ stable and $\mathcal{C}_W$ is formally defined as follows.

**Definition 4.** Given $W = (A, W_m)$ stable, a Constraint Diagram $\mathcal{C}_W$ associated to $W$ is a Class Diagram obtained translating $W$ by the encoding $\langle \rangle$ so defined:
 i. Let $\forall o \in O_m$ such that $o \subseteq$$<operation name='''O1'>$

\[ \langle input name='''m_k1' message='''tns:m_k1-Document'\rangle$\

\[ \langle output name='''m_k2' message='''tns:m_k2-Document'\rangle$\]

\[ \langle fault name='''m_k3' message='''tns:m_k3-Document'\rangle$\]

\[ \langle X(m) \rangle = \langle m \rangle \]


...$

\langle x\rangle$

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

ii. Let $\forall o \in O_m$, $p_\text{in}(o) = \{m_k\}$ and $p_\text{out}(o) = \{m_k\} | \ | 2 \leq h \leq 3 \}$ if there exists an association from $c(m_1)$ to $c(m_k)$;

iii. $\forall i, o2 \in O_m, p_\text{in}(o2) = \{m_k\}$, $p_\text{out}(o2) = \emptyset$, $p_\text{in}(o2) = \{m_k\}$, $\exists q_i, q_o \in S$ such that $(q_i, (m_k, g(i,k)), Q_i) \in \delta$, $(q_i, (m_k, g(i,k)), Q_o) \in \delta$ and $q_o \in Q_{11}$, if there exists an association from $c(m_k)$ to $c(m_k)$;

iv. $\forall o, o2 \in O_m, p_\text{out}(o1) = \{m_k\} | \ 2 \leq h \leq 3 \}$ and $p_\text{out}(o2) = \{m_k\}$, $\exists q_i, q_o \in S$ s.t. $(q_i, (m_k, g(i,k)), Q_i) \in \delta$, $(q_o, (m_k, g(i,k)), Q_o) \in \delta$ and $q_o \in Q_{11}$, if there exists an association from $c(m_k)$ to $c(m_k)$;

v. For every guard $g(i,k) = g(q_i, m_k, d(i,k)) \in G$, then $[g(i,k)] = \mathcal{C}_W$.

The encoding is reversible - up to the introduction in the Constraint Diagram of (i) an empty class Start representing an empty message, (ii) for every class $i$, associated to an initial message, an association from Start to $i$, and (iii) for every class $i$, an attribute denoting the nature of the message - i.e. either inbound or outbound - modeled by $C_i$ - it is also possible to start with the design of a novel conversation in the form of Constraint Diagram, and then deducing 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 valid, or he has just filled a Registration form in the current session and login username is valid; (iv) the allowed max number of failed logins is 3. In terms of WSDL document, we could define a Login operation, including LoginRQ as inbound element, ValidLoginRS and InvalidLoginRS as outbound elements, and a Registration operation, including RegistrationRQ and RegistrationRS respectively as inbound and outbound elements. Fig.1 shows the Constraint Diagram C_D_W, 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