Constraint-based Dynamic Conversations

Diletta Cacciagrano, Flavio Corradini, Rosario Culmone, Leonardo Vito
Mathematics and Computer Science Department
University of Camerino
Camerino, Italy

Abstract

In a service-oriented architecture, systems communicate by exchanging messages. In this work, we propose two conversation specification language-independent models - the first based on first-order guarded automata and the latter based on attribute grammars - for describing valid client-server conversations. On the basis of the latter, we also propose a purely XML-based framework for a dynamic monitoring/validation of valid sequences of valid invocation messages.

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) [8] is the standard used for publishing abstract and concrete descriptions of Web Services - including the schema 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 - some examples are [1], [6], [5] and [7] - 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 Web Services in terms of flow of exchanged messages\(^1\). For example, in [12] mediated composite services specified in BPEL are verified against the design specified using Message Sequence Chart and Finite State Process notations, while in [13] finite automata are augmented with XML messages, XPath [10] expressions and boolean conditions, in order to verify temporal properties of single and composite Web Service conversations.

2 Models and framework

In the following, we describe the main contributions of this work.

2.1 Conversation specification language-independent models

First, we propose two conversation specification language-independent models for describing valid client-server conversations based on valid exchanged messages - where valid is intended w.r.t. message templates and transitions. Without loss of generality, we only assume to handle a generic XML-based document describing conversations, a WSDL document describing message schema\(^2\) and CLiX\(^3\) [2] specifying first-order logical constraints in XML on message templates and transitions, as proposed in [11].

The first model is based on an extension of the boolean guarded automata model proposed in [13]. As defined in [13], a guarded automaton is suitable to be easily translated into PROMELA - the SPIN [14] input language based on boolean logic - and verified by SPIN, being the verification of Web Service specifications the aim of [13]. We extend this model (i) by imposing first-order logic guards, (ii) by expressing them as CLiX rules, (iii) by describing message types as WSDL operation parameters, as well as (iv) by optimizing the automaton memory representation as a finite

\(^1\)In terms of flow of exchanged messages, the behavior of a service

\(^2\)The proposed framework also fits on a scenario where message templates are described by XML schema [4].

\(^3\)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.
vector of XML variables and (v) by modeling the automaton history - made of sent/received message instances - as a simple concatenation of XML documents.

The latter model is based on attribute grammars [15], i.e. context-free grammars equipped with a finite set of attributes, a set of evaluation rules and a set of logical conditions for attribute values.

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 \( W_m \) elements. Then, we state a relationship, called stability, between \( W_c \) and \( W_m \) as follows: for each operation \( o \) in \( W_m \), the XML scheme associated to each input, resp. output/fault, operation parameter \( p \) in \( o \) is the XML scheme 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 parameter XML schema, and (ii) there is a syntactic match between XML schema of messages in \( W_c \) and XML schema of operation parameters in \( W_m \).

From guarded automata to attribute grammars: On the one hand, it is not surprising that most of the conversation specifications presented in the introduction are closely related to the finite state automata formalism; in fact, a conversation can be viewed as a sequence of message symbols accepted from a finite state automaton [13].

On the other hand, it easy to prove that a guarded automaton can be mapped into a regular attribute grammar\(^4\), and also the vice versa: Proposition 5.1 is an example of encoding from guarded automata into (regular) attribute grammars. Regarding the expressive power on boolean logic of guarded automata - i.e. regular attribute grammars - w.r.t. non-regular attribute grammars, the main results are summarized in [18, 16]: (i) guarded automata can accept the same class of languages as Turing machines; (ii) the terminating guarded automata with finite variable domains accept regular languages; (iii) simple guarded automata - with unrestricted variable domains - can accept at least some context sensitive languages; (iv) guarded automata, where attributes are set of variables without structure, cannot accept context-free languages.

If we take into account first-order logic, it is not so clear how regular attribute grammars and non-regular ones are related in term of expressiveness. Anyway, we could think about first-order constrained conversations which can be easier specified by non-regular attribute grammars than by regular ones: it suffices to take into account any conversation where the sequence of invocations includes either operation trade-off or memory-based properties. For instance, consider the non-regular attribute grammar \( G \) with start symbol \( s \) and set of productions \( P \) so defined:

\[
\{s := m_1 B m_2 \ g_1(m_1, m_2); B := m_1 m_2 \ g_2(m_1, m_2),\}
\]

where \( g_1 \) and \( g_2 \) a first-order logical conditions on \( m_1, m_2 \).

\( G \) models a conversation where (i) two WSDL operations, we say \( O_1 \) and \( O_2 \), are defined with input parameters respectively \( m_1 \) and \( m_2 \), (ii) the sequence of \( O_1 \) (invocations) has to precede the one of \( O_2 \), (iii) the number of \( O_1 \) (invocations) equals the one of \( O_2 \), and (iv) for any \( O_1 \) and \( O_2 \) (invocation), constraints \( g_1 \) and \( g_2 \) have to be satisfied.

Although guarded automata are suitable to specify the most of existing conversations, attribute grammars look as a good compromise in terms of expressivity, constraint complexity and amount of variables - above all for modeling more complex (i.e. non-regular) conversations. For this reason, the input for our monitoring/validation framework is given in the form of attribute grammar. Another reason behind the choice of specifying constrained conversations in the form of attribute grammars is related to constraint aspects: the scope of a conversation constraint includes both message templates and transitions; it follows that a constraint can be put neither only in a conversation document - since here we could only define conditions to able/disable message transitions - nor in a WSDL document - since here we could only define conditions to well-type messages. Logical conditions in attribute grammars are associated to productions (i.e. transitions) and they involve symbol attributes (i.e message attributes): for this reason, they look suitable to encode conversation constraints.

\[2.2\] From attribute grammars to invocation stubs

In the following, we describe in detail the monitoring/validation framework, based on attribute grammars.

1. Let \( C = \langle W_c, W_m, G \rangle \) be a constrained client-server conversation, where \( W_m \) is a WSDL document - being \( W_m \) and \( W_c \) syntactically related by a stability relationship - and \( G \) is a CLiX rule document. \( C \) can be in the form of either an XML-formatted guarded automaton or an XML-formatted attribute grammar. In the first case, a slave parser worries to translate it in the form of an associated XML-formatted attribute grammar \( G_W \).

2. Given \( G_W \), the slave parser links by namespaces respectively nonterminals with WSDL operations, terminals with WSDL operation parameters, CLiX rule arguments with WSDL element attributes. It also marks grammar productions in order to distinguish those ones corresponding to client’s requests and those ones corresponding to service’s responses.

3. WSDL2Java [9] is a concrete platform providing the client-side generation - through the elaboration of WSDL
operation descriptions - of Java classes and interfaces for setting SOAP endpoints and invoking the operations associated to those descriptions. For our purpose, WSDL2Java has been modified in order to handle complex types on the fly, by Java reflection. We denote by $S$ the stub produced by WSDL2Java.

4. A programmer freely uses $S$ to write any client’s business logic involving any operation invocation sequence. We denote by $J$ a client business logic $j_0; q_1.I n v o k e(m_{i1}); j_1; m_{o1}); q_2. I n v o k e(m_{i2}); j_2; m_{o2}); \ldots; q_n. I n v o k e(m_{in}); j_n; m_{on})$ on $S$. $q_x. I n v o k e(m_{ix})$ denotes the invocation of a method $I n v o k e$ on $q_x$ with input instance $m_{ix}$, and its effects consist of (i) marshalling\(^5\) and packing a SOAP message to invoke the corresponding WSDL operation $o_x$ with the corresponding input instance $m_{ix}$\(^6\), and (ii) activating a master parser. $j_0$ denotes any Java code which does not involve an invocation, and $j_x(m_{ox})$ denotes any Java code which does not involve an invocation and possibly involves an output/fault instance $m_{ox}$ - that one associated to $m_{ix}$\(^7\), and (iii) a master parser. $j_0$ denotes any Java code which does not involve an invocation, and $j_x(m_{ox})$ denotes any Java code which does not involve an invocation and possibly involves an output/fault instance $m_{ox}$ - that one associated to $m_{ix}$\(^7\) - relative to a previous invoked operation $o_x$.

5. The master parser inputs $G_W$ and $W_m$. Since (i) non-terminals and terminals in $G_W$ are linked by namespaces to respectively $W_m$ operations and operation parameters, then any grammar production of the form $< Q, x > := I(m_{ix}) | < Q, j > g(x, ix)$ corresponds to a possible instruction $q_x. I n v o k e(m_{ix})$.

The master parser looks for deriving in $G_W$, $m_{i1}[m_{o1}]m_{i2}[m_{o2}]; \ldots; m_{in}[m_{on}]$\(^9\).

(Request case) For any $m_{ix}$, the master parser triggers an error whenever a $q_x. I n v o k e(m_{ix})$ in $J$ is not correct w.r.t. the conversation protocol, i.e. when it cannot apply a production $< Q, x > := I(m_{ix}) | < Q, j > g(x, ix)$.

(Response case) Analogously, it triggers an error whenever a $j_x(m_{ox})$ in $J$ is not correct w.r.t. the conversation protocol, i.e. when it cannot apply a production $< Q, x > := O(m_{ox}) | < Q, j > g(x, ox)$.

Otherwise, it invokes the validator OpenCLiXML [3]. OpenCLiXML is an open source Java implementation of the freely available CLiX specification from Systemwire [17]: it provides optimized rules processing against data represented in XML and, more important, it includes inter-document checks.

6. OpenCLiXML checks $m_{ix}$ (respectively $m_{ox}$) w.r.t. the CLiX rule linked to $g(x,ix)$ (respectively to $g(x,ox)$) and an XML history document $H$, containing the concatenation of all the input and output/fault instances.

- If the validation is positive:
  (Request case) the SOAP message generated from $q_x. I n v o k e(m_{ix})$ is forwarded to the network (i.e. the invocation is fired) and the input instance contained in it is appended in $H$.
  (Response case) the output instance is extracted from the received SOAP message, it is appended in $H$, it is unmarshalled\(^10\) and the corresponding object is assigned to $m_{ox}$.

- In the opposite case: the master parser catches the OpenCLiXML exception and it returns informations about the error. In the (Request case) the invocation is not fired.

The above procedure allows the programmer:
(i) to automatically generate a message sequence $m_{i1}[m_{o1}]m_{i2}[m_{o2}]; \ldots; m_{in}[m_{on}]$ which belongs to the conversation protocol associated to the service; (ii) to forward an invocation message only after a validation step (items 5-6) - hence without waiting for service acks, error messages and other additional informations; (iii) to precisely detect which requests/responses have triggered an error.

Fig.1 shows the monitoring/validation framework in terms of relationships among components. Labels in boxes denote documents, those ones in ovals denote processes and those ones in diamonds denote exceptions.

---

\(^5\)The marshalling procedure consists of transforming a programming object into a XML serialized object.

\(^6\)A SOAP message contains all the WSDL operation, input - and eventually output/fault - message names involved in the current invocation.

\(^7\)The symbol $j_x(m_{ox})$ denotes zero or one occurrences of $m_{ox}$ - i.e. respectively $j_x$ and $j_x(m_{ox})$. The latter denotes the case when the code involves an output/fault instance associated to $m_{ox}$ and relative to a previous invoked operation $o_x$, where $o_x$ prototype includes an output/fault parameter $m_{ox}$. We recall that the correspondence is contained into the received SOAP message.

\(^8\)The symbol $[Q, j]$ denotes zero or one occurrences of $Q, j$.

\(^9\)The symbol $[n_{ox}]$ denotes zero or one occurrences of $n_{ox}$.

\(^10\)The unmarshalling is the inverse procedure of marshalling, and it consists of transforming a XML serialized object into a programming object.
3 Background on attribute grammars

In the following, we will introduce some background knowledge on the attribute grammar model. Expert readers can skip the topic and jump to Section 4.

Definition 3.1 A context-free grammar is a tuple \( G = \langle N, T, s, \mathcal{P} \rangle \), where \( N \) is a finite set of nonterminal symbols, \( T \) is a finite set of terminal symbols, \( s \in N \) is the start symbol and \( \mathcal{P} \) is a set of productions of the form \( X_0 ::= X_1X_2 \ldots X_{m-1}X_m \), where \( X_0 \in N \), \( X_1X_2 \ldots X_{m-1}X_m \in (N \cup T)^+ \) and \( m \geq 1 \).

The language associated to a context-free grammar \( G \) is denoted by \( \mathcal{L}(G) \) and it is the set of strings \( w \in T^* \) which can be derived by the start symbol \( s \) applying a finite number of productions in \( \mathcal{P} \):

\[ \mathcal{L}(G) = \{ w \in T^* | s \xrightarrow{*} w \} \]

where \( S \xrightarrow{*} w \) means that \( s = \beta_0 \Rightarrow \beta_1 \Rightarrow \ldots \Rightarrow \beta_k \) is a derivation tree over \( G \), where \( \forall i \in [1..k], \beta_i \Rightarrow \beta_{i+1} \) means that \( \beta_{i+1} = uX_0v, \beta_i = uX_1X_2 \ldots X_{m-1}X_mv \) and \( X_0 ::= X_1X_2 \ldots X_{m-1}X_m \in \mathcal{P} \).

Definition 3.2 A derivation tree \( t \) over a context-free grammar \( G \) is a tree labeled with symbols from \((N \cup T)\) s.t.: i. the root of \( t \) is labeled with \( s \); ii. for every interior node \( n_0 \) with children \( n_1, \ldots, n_m \), there exists a production \( X_0 ::= X_1X_2 \ldots X_{m-1}X_m \) such that \( \forall i \in [0..m], \text{lab}(n_i) = X_i \) (in left-to-right order), where \( \text{lab}(n_i) \) denotes the label of the node \( n_i \); iii. every leaf node is labeled with a terminal.

We denote by \( t_C(w) \) a derivation tree over \( G \), where \( w = \text{lab}(f_1) \ldots \text{lab}(f_h) \) and \( \forall j \in [1..h], f_j \) is a leaf node in \( t_C(w) \). Obviously \( L(G) = \{ w \in T^* | \exists t = t_C(w) \} \).

Now, we can give the definition of a First-order attribute (FOA) grammar.

Definition 3.3 A FOA grammar \( G \) is a context-free grammar \( G = \langle N, T, s, \mathcal{P} \rangle \), enriched with the following elements: i. Attribute grammar vocabulary: it is a tuple \( \langle A_t, S, I, A \rangle \): - \( A_t \) is a finite set of attributes; - \( S, I, A \) are functions from \((N \cup T)\) to the powerset of \( A_t \); - \( \forall X \in (N \cup T), A(X) = I(X) \cup S(X) \) is the set of attributes, \( I(X) \) is the set of inherited attributes, \( S(X) \) is the set of synthesized attributes of \( X \); - \( \forall X \in N, I(X) \cap S(X) = \emptyset \); - \( \forall X \in T, S(X) = I(X) = \emptyset \); - \( I(S) = \emptyset \). ii. Evaluation rules: Denoting \( D(A(X)) \) the semantic domain of \( A(X) \), then \( \forall X_0 ::= X_1 \ldots X_m \in \mathcal{P} \), it holds: - \( \forall A \in S(X_0) \) there is a function \( r_1 : D(A(X_1)) \times \ldots \times D(A(X_m)) \times D(I(X_0)) \rightarrow D(S(X_0)) \); - \( \forall A \in I(X_k) \) there is a function \( r_2 : D(A(X_0)) \times D(S(X_1)) \times \ldots \times D(S(X_m)) \rightarrow D(I(X_k)) \); iii. First-order logical conditions: \( \forall X_0 ::= X_1 \ldots X_m \in \mathcal{P} \), there is a set of first-order logical formulas \( \{ \phi | \phi \text{ is defined on } D(A(X_0)) \times \ldots \times D(A(X_m)) \} \).

Definition 3.4 Given a FOA grammar \( G \), a derivation tree \( t_G(w) \) over \( G \) is valid if and only if (i) all the attribute values conform to the evaluation rules and (ii) all the logical conditions are true.

Definition 3.5 The language associated to a FOA grammar \( G \) is \( L(G) = \{ w \in T^* | \exists t = t_G(w) \text{ and } t_G(w) \text{ is valid} \} \).

FOA grammar specifications define the dependencies among attributes. Such specifications must exhibit certain properties - for instance, dependencies must not be circular. However, there are methods transforming an attribute grammar in such a way such properties hold.

4 Models for valid first-order constrained client-server conversations

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

Notation 4.1 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 \( M_f \) the finite sets of respectively inbound and outbound message types in \( W_c \) and \( W_m \); by \( x(d) \) the \( d \)'s XML scheme.

First, we abstract from the tuple \( \langle W_c, W_m, G \rangle \), replacing it with its guarded automaton-based representation.

Definition 4.1 A First-Order guarded (FOG) automaton associated to \( \langle W_c, W_m, G \rangle \) is \( A = \langle S, M, H, q_0, q_n, \delta, G \rangle \), where:

i. \( S = \{ q_t | t \in [0..n], n \in \mathbb{N} \} \) is a finite set of states;
ii. \( M = M_I \cup M_O \) is as above described;
iii. \( H = \{ h_{1, \ldots, h_{|M|}} \} \) 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\(^{11}\);
\( G = \{ g_{(i,k)} = g(q_i, m_k, \langle d_1, \ldots, d_{|M|} \rangle) \} \text{ CLiX rule} \), such that \( q_t \in S, m_k \in M \) and \( \forall j \in [1..|M|], d_j \in \{ h_j(i), \lambda \} \).
vi. \( \delta = \{ \{ q_t, \langle i, g_{(i,k)}(Q_i) \rangle \} \} \) is a state transition relation, where \( q_t \in S, Q_i \subseteq S, i \in [1..|M|], \{ m_k | m_k \in M_i \} \cup \{ m_k | m_k \in M_o \} \) and \( g_{(i,k)} \in G \).

\(^{11}\)\( q_0 \) and \( q_n \) can coincide.
Message types and local variables are XML documents. Each local variable $h_j$ in $H$ corresponds to a message type $m_j$ in $M$. ∀$q_i \in S$ and ∀$j \in [1...|M|]$, $h_j(i)$ denotes an XML document - the history of $m_j$ until the state $q_i$, obtained by enqueuing all the sent/received message instances, until the state $q_i$, that correspond to the type $m_j$. Each transition $\tau \in \delta$ is in one of the following two forms:

(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 $q_p \in Q_i$, it removes the received message instance (of type $m_k$) from the input queue and it updates $h_k$ in $H$, corresponding to $m_k$, by the concatenation of the received instance, in the case $g_{i,k}$ holds.

(send-transition) $\tau = (q_i, (\overline{m}_k, g_{i,k}), Q_i)$, where $m_k \in M_o$: the transition nondeterministically changes the state of the automaton from $q_i$ to $q_p \in Q_i$, it appends the sent message instance (of type $m_k$) to the input queue of the client and it updates $h_k$ in $H$, corresponding to $m_k$, by the concatenation of the sent instance, in the case $g_{i,k}$ holds.

**Definition 4.2** Let $\mathcal{A} = (S, M, H, q_0, q_n, \delta, \mathcal{G})$ be a FOA automaton associated to $\langle W_c, W_m, \mathcal{G} \rangle$. Given a guard $g_{i,k}$, we denote by $O_m$ the set of operation in $W_m$; for every $o \in O_m$, by $p_{in}(o)$ and $p_{out}(o)$ the sets of respectively input and output/fault parameters of $o$. We also assume that $W_c$ and $W_m$ 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. $M_o$), $x(m_k) = x(p_k)$ holds. We formally define this kind of relationship between $W_c$ and $W_m$ as follows.

**Definition 4.3** Let $\mathcal{A} = (S, M, H, q_0, q_n, \delta, \mathcal{G})$ be a FOA automaton associated to $\langle W_c, W_m, \mathcal{G} \rangle$, and let $W_m$ be a WSDL document. $W = (A, W_m)$ is stable if and only if ∀$q_i \in S$ such that $(q_i, (m_k, g_{i,k}), Q_i) \in \delta$:

i. ∃$o \in O_m$ s.t. $p_{in}(o) = \{p_k\}$ and $x(p_k) = x(m_k)$;

ii. $\exists q_{i_h} \in Q_i, (2 \leq h \leq 3)$ s.t. $(q_{i_h}, (\overline{m}_k, g_{i,k}), Q_{i_h}) \in \delta$ iff $p_{out}(o) = \{p_k\}$, $2 \leq h \leq 3$ and $x(p_k) = x(m_k)$.

The stability assumption (Definition 4.3) implies that it is possible to use wherever the WSDL operation parameter $p_k$ in place of the message $m_k$, and that the actual context of any guard in $\mathcal{G}$ only involves $W_m$ operation parameter XML schema.

5 From FOA automata to FOA grammars

In this section, we define the second model for valid first-order constrained client-server conversations, as usual meaning valid w.r.t. a set of CLiX rules. The model has been widely justified in the introduction. Here, we define an attribute grammar associated to a conversation $C = \langle W_c, W_m, \mathcal{G} \rangle$ by encoding a guarded automaton associated to $C$ into the attribute grammar formalism.

**Definition 5.1** Let $W = (A, W_m)$ be stable. A FOA grammar $G_W = (N, T, s, \mathcal{P})$ associated to $\mathcal{A} = (S, M, H, q_0, q_n, \delta, \mathcal{G})$ is defined as follows:

i. $T = \{\Gamma_m \mid m_i \in M\} \cup \{0\}$

ii. $s = \langle Q, 0 \rangle$

iv. $\mathcal{P} = \mathcal{P}_{eq} \cup \mathcal{P}_{rs}$, where:

$v$. (Attribute grammar vocabulary): it is a tuple $\langle A, T, I, A \rangle$:

vi. Evaluation rules: For every $\langle Q_i, o \rangle \in N$, $A(< Q_i, o >) = I(< Q_i, o >)$

vii. (First-order logical conditions): Given $\langle Q_i, o \rangle$ in $\mathcal{P}$, $g_{i,k} = \text{CLiX rule associated to the transition } (q_i, (m_k, g_{i,k}), Q_i) \in \delta$ (respectively $(q_i, (m_k, g_{i,k}), Q_i) \in \delta$ and $q_n \in Q_i$).

Notice that client’s and service’s productions belong to the same grammar $G_W$; however, there is no ambiguity among productions corresponding to client’s requests ($\mathcal{P}_{eq}$) and the ones corresponding to service’s responses ($\mathcal{P}_{rs}$).

6 A toy example

Suppose to design a toy authentication service as follows: (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, with LoginRQ as input parameter, ValidLoginRS and InvalidLoginRS respectively as output and fault parameters, and a Registration operation, with RegistrationRQ and RegistrationRS respectively as input and output parameters.

A FOA grammar $G_W$ is defined in the following.

$$
< Q.0 > ::= < Q.1Registration > | < Q.2Login >
$$

$$Q.1Registration ::= I(a.1)< Q.3OutRegistration >
Q.2Login ::= I(a.2)< Q.4OutLogin >
Q.3OutRegistration ::= 0(m.3)< Q.2Login >
Q.4OutLogin ::= 0(m.4)< Q.1Registration >
Q.5
$$

$$m_1 = RegistrationRQ \quad m_4 = InvalidLoginRS
m_2 = LoginRQ \quad m_3 = ValidLoginRS
m_3 = RegistrationRS
$$

rule1: context LoginRQ inv:
(self.RegistrationRS.RegistrationRQ->notEmpty() &&
self.RegistrationRS.LoginRQ->notEmpty()) implies
self.RegistrationRS.LoginRQ.login = self.username

rule2: context LoginRQ inv:
LoginRQ.allInstances -> count(InvalidLoginRS) <= 3

For readability, the grammar is not in XML format and constraints are written as OCL formulas. At this aim, Fig.2 shows the authentication protocol above described in term of Class Diagram. Each class models an operation parameter with its WSDL attributes; analogously, each OCL formula translates a constraint. LoginRQ.allInstances denotes the set of LoginRQ instances and LoginRQ.allInstances->count(InvalidLoginRS) denotes the number of LoginRQ instances associated to InvalidLoginRS ones.

References


Figure 2. A toy authentication service Class Diagram.

\[\text{It is well-known that the first-order logical fragment of OCL can be encoded into CLiX.}\]