STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 2, 2012

ALGEBRAIC APPROACH TO IMPLEMENTING AN ATL MODEL CHECKER

LAURA FLORENTINA STOICA AND FLORIAN MIRCEA BOIAN

Abstract. The Alternating-Time Temporal Logic (ATL) is interpreted over the game structures. An open system interacts with its environment and its behavior depends on the state of the system as well as the behavior of the environment. In this paper we will use an algebraic approach to implement an ATL model checker. Our tool is implemented in client-server paradigm: ATL Designer, the client tool, allows an interactive construction of concurrent game structures as a directed multi-graph and the ATL Checker, the core of our tool, represents the server part and is published as Web service. The ATL Checker includes an algebraic compiler implemented with ANTLR (Another Tool for Language Recognition) support. The main function of the Web service is to parse a given formula, find the set of nodes in which the formula is satisfied and return the result to the user.

1. Introduction

An *open system* is a system that interacts with its environment and whose behaviour depends on the state of the system as well as the behaviour of the environment. In order to construct models suitable for open systems, the Alternating-time Temporal Logic (ATL) was defined [1].

A Computation Tree Logic (CTL) specification is interpreted over Kripke structures, which provide a model for the computations of a closed system. In a *closed system* the behaviour is completely determined by the state of the system. In order to capture compositions of open systems, we present an extension of CTL, the alternating-time temporal logic (ATL), which is interpreted over game structures.

ATL extends CTL by replacing the path quantifiers $\exists$ (existential quantification) and $\forall$ (universal quantification) by cooperation modalities $\langle\langle A\rangle\rangle$,

Received by the editors: May 5, 2012.

2010 *Mathematics Subject Classification.* 68Q60, 68Q85, 68N20.

1998 *CR Categories and Descriptors.* **C.2.4** [Computer-Communication Networks]: Distributed Systems - *Client/server*; **D.2.4** [Software Engineering]: Software/Program Verification - *Model checking*; **F.3.1** [Logics and meanings of programs]: Specifying and Verifying and Reasoning about Programs - *Model checking*.

Key words and phrases. algebraic compiler, ATL model checking, ANTLR, Web services.

where A is a team of agents. A formula $\langle\langle A\rangle\rangle\ \varphi$ expresses that the team A has a collective strategy to enforce φ.

ATL is a branching-time temporal logic that naturally describes computations of multi-agent system and multiplayer games. It offers selective quantification over program-paths that are possible outcomes of games [1]. ATL uses alternating-time formulas to construct model-checkers in order to address problems such as receptiveness, realizability, and controllability.

Model checking is a technology used for verification and validation of automated system.

Over structures without fairness constraints, the model-checking complexity of ATL is linear in the size of the game structure and length of the formula, and the symbolic model-checking algorithm for CTL [2] extends with few modifications to ATL.

In the following the ATL language is defined. The operator scheme Σ_{atl} is defined as a triple $\langle S_{atl}, O_{atl}, \sigma_{atl}\rangle$ where set S_{atl} contains the representations of the ATL formulas, $O_{atl} = \{\neg, \vee, \wedge, \rightarrow, \Diamond, \circ, \Box, U\}$ is the set of operators, and the $\sigma_{atl} : O_{atl} \rightarrow S^*_{atl} \times S_{atl}$ is a function which defines the signature of the operators. The $\Diamond$ ('future'), $\circ$ ('next'), $\Box$ ('always'), and U ('until') are temporal operators. The ATL model checker can be defined as the Σ_{atl} - language [4] given in the form $L_{atl} = \langle Sem_{atl}, Syn_{atl}, \mathfrak{L}_{atl} : Sem_{atl} \rightarrow Syn_{atl}\rangle$ where Syn_{atl} is the word algebra of the operator scheme Σ_{atl} generated by the operations from O_{atl} and a finite set of variables, representing atomic propositions, denoted by AP. Sem_{atl} represents ATL semantic algebra defined over the set of ATL formulas which are satisfied by the ATL model. $\mathfrak{L}_{atl}$ is a mapping which associates the set of satisfied formulas from Sem_{atl} to ATL expressions from Syn_{atl} which satisfy these formulas.

Having well-defined ATL language, implementation of an ATL model checker will be equated with an algebraic compiler which translates an ATL formula of the ATL model to set of nodes over which that formula is satisfied.

We have chosen an algebraic approach to implement the ATL symbolic model checking algorithm and we used a Web services technology to make our model checker tool available to various clients. We provide as part of our tool an example of GUI Client for the Web service.

The paper is organized as follows. In section 2 we present the definition of the concurrent game structure with its syntax and semantics. In section 3 is presented the implementation of an algebraic compiler used by our tool to verify satisfiability of ATL formulas for given models. Invocation of the compiler will be accomplished through a Web service described in section 4. In section 5 we test our new model checking tool on a simple ATL model. Conclusions are presented in section 6.

2. ATL logic with its syntax and semantics

Alternating-time Temporal Logic (ATL) is a branching-time temporal logic that naturally describes computations of multi-agent system and multiplayer

games. It offers selective quantification over program-paths that are possible outcomes of games [1].

Concurrent game structures can be used to model compositions of open systems. Unlike in Kripke structures, in a concurrent game structure, the environment is involved in a state transition. The environment is modelled by a set of agents. Each agent may perform some actions and at least one action is available to the agent at each state.

The concurrent game structure in [1] is defined as a tuple $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ with the following components:

- A natural number $k \geq 1$ of *players*. We identify the players with numbers $1, ..., k$.
- A finite set Q of states.
- A finite set Π of *propositions* (also called *observables*).
- For each state $q \in Q$, a set $\pi(q) \subseteq \Pi$ of propositions true at q. The function π is called *labelling* (or *observation*) *function*.
- For each player $a \in \{1, ..., k\}$ and each state $q \in Q$, we identify the moves of player a at state q with the numbers $1, ..., d_a(q)$, where $d_a(q) \geq 1$ represents the number of available moves. For each state $q \in Q$, a *move vector* at q is a tuple $\langle j_1, ..., j_k \rangle$ such that $1 \leq j_a \leq d_a(q)$ for each player a. Given a state $q \in Q$, we write $D(q)$ for the set $\{1, ..., d_1(q)\} \times ... \times \{1, ..., d_k(q)\}$ of moves vector. The function D is called *move function*.
- For each state $q \in Q$ and each move vector $\langle j_1, ..., j_k \rangle \in D(q)$, $\delta(q, j_1, ..., j_k) \in Q$ represents the state that results from state q if every player $a \in 1, ..., k$ chooses a move j_a. The function δ is called the *transition function*.

For a computation starting at state q we refer to it as a *q-computation*. For a computation λ and a position $i \geq 0$, we use $\lambda[i]$ to denote the i-th state of λ, $\lambda[0, i]$ to denote the finite prefix $q_0, q_1, ..., q_i$ of λ and $\lambda[i, \infty]$ to denote the infinite suffix $q_i, q_{i+1}, ...$, of λ.

The s*yntax of a temporal logic ATL* is defined with respect to a finite set Π of propositions and a finite set $\Lambda = \{1, ..., k\}$ of players [1].

An ATL formula is one of the following:

(s1): p, for propositions $p \in \Pi$;
(s2): $\neg\varphi$, $\varphi_1 \vee \varphi_2$, where φ and $(\varphi_i)_{i=\overline{1,2}}$ are ATL formulas;
(s3): $\langle\langle A \rangle\rangle \circ \varphi$, $\langle\langle A \rangle\rangle \Box \varphi$, or $\langle\langle A \rangle\rangle \varphi_1 U \varphi_2$, where $A \subseteq \Lambda$ is a set of players, and φ, $(\varphi_i)_{i=\overline{1,2}}$ are ATL formulas.

The operator $\langle\langle\rangle\rangle$ is a path quantifier, and $\circ$ ('next'), $\Box$ ('always'), and U ('until') are temporal operators.

A *strategy* for player $a \in \Lambda$ is a function $f_a : \mathcal{Q}^+ \to \mathbb{N}$ that maps every nonempty finite state sequence $\lambda = q_0 q_1 ... q_n$, $n \geq 0$, to a natural number such that $f_a(\lambda) \in \{1, ..., d_a(q_n)\}$. Thus, the strategy f_a determines for every finite prefix λ of a computation a move $f_a(\lambda)$ for player a in the last state of λ.

Given a set $A \subseteq \{1, ..., k\}$ of players, a state $q \in Q$ and a set $F_A = \{f_a | a \in A\}$ of strategies, one for each player in A, the outcome of F_A is defined to be the set $out(q, F_A)$ of q-computations that the players from A are enforcing when they follow the strategies in F_A. A *computation* $\lambda = q_0, q_1, q_2, ...$ is in $out(q, F_A)$ if $q_0 = q$ and for all positions $i \geq 0$, there is a move vector $\langle j_1, ..., j_k \rangle$ such that $j_a = f_a(\lambda[0, i])$ for all players $a \in A$ and $\delta(q_i, j_1, ..., j_k) = q_{i+1}$ [1].

Formal definition of ATL semantics given in [1] is defined over a game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$. We write $S, q \models \varphi$ to indicate that the state q satisfies the formula φ in the structure S. When S is clear from the context, we omit it and write $q \models \varphi$.

The satisfaction relation $\models$ is defined for all states q of S inductively as follows:

- $q \models p$, for propositions $p \in \Pi$, iff $p \in \pi(q)$;
- $q \models \neg\varphi$ iff $q \not\models \varphi$;
- $q \models \varphi_1 \vee \varphi_2$ iff $q \models \varphi_1$ or $q \models \varphi_2$;
- $q \models \varphi_1 \wedge \varphi_2$ iff $q \models \varphi_1$ and $q \models \varphi_2$;
- $q \models \varphi_1 \rightarrow \varphi_2$ iff $q \models \neg\varphi_1$ or $q \models \varphi_2$;
- $q \models \langle\langle A \rangle\rangle \circ \varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models \varphi$;
- $q \models \langle\langle A \rangle\rangle \Box\varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, and all positions $i \geq 0$ such that $\lambda[i] \models \varphi$;
- $q \models \langle\langle A \rangle\rangle \Diamond\varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, there exists a position $i \geq 0$ such that $\lambda[i] \models \varphi$;
- $q \models \langle\langle A \rangle\rangle \varphi_1 U \varphi_2$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, there exists a position $i \geq 0$ such that $\lambda[i] \models \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models \varphi_1$

3. Java implementation of ATL model checker

Our algebraic compiler $\mathcal{C}$ translates a formula φ of the ATL model to the set of nodes Q' over which formula φ is satisfied. That is, $\mathcal{C}(\varphi) = Q'$ where $Q' = \{q \in Q | q \models \varphi\}$.

For the implementation of our algebraic compiler we choose the ANTLR [3]. ANTLR is a compiler generator which takes as input a grammar - an exact description of the source language, and generates a recognizer for the language defined by the grammar.

The algebraic compiler $\mathcal{C}$ implements the following ATL symbolic model checking algorithm given by [1]. We add three more symbolic operators $(\wedge, \rightarrow, \Diamond)$ in the ATL symbolic model checking algorithm to show all operators form O_{atl}.

```
Function Eval_A(φ) as set of states ⊆ Q
  case φ = p:
      return [p];
  case φ = ¬θ:
      return Q \ Eval_A(θ);
  case φ = θ1 ∨ θ2:
      return Eval_A(θ1) ∪ Eval_A(θ2);
  case φ = θ1 ∧ θ2:
      return Eval_A(θ1) ∩ Eval_A(θ2);
  case φ = θ1 → θ2:
      return (Q \ Eval_A(θ1)) ∩ Eval_A(θ2);
  case φ = ⟨⟨A⟩⟩ ∘ θ:
      return Pre(A, Eval_A(θ));
  case φ = ⟨⟨A⟩⟩□θ:
      ρ := Q; τ := Eval_A(θ); τ0 := τ;
      while ρ ⊈ τ do
          ρ := τ;
          τ := Pre(A, ρ) ∩ τ0;
      wend
      return ρ;
  case φ = ⟨⟨A⟩⟩◇θ:
      ρ := ∅; τ := Eval_A(θ);
      while τ ⊈ ρ do
          ρ := ρ ∪ τ;
          τ := Pre(A, ρ) ∩ Q;
      wend
      return ρ;
  case φ = ⟨⟨A⟩⟩θ1Uθ2:
      ρ := ∅; τ := Eval_A(θ2); τ0 := Eval_A(θ1);
      while τ ⊈ ρ do
          ρ := ρ ∪ τ;
          τ := Pre(A, ρ) ∩ τ0;
      wend
      return ρ;
End Function
```

The $Pre(A, \rho)$ function, where $A \subseteq \Lambda$ and $\rho \subseteq Q$, returns the set of states q such that from q, the players in A can cooperate and enforce the next state to be in ρ. $Pre(A, \rho)$ contains state $q \in Q$ if for every player $a \in A$ there exists a move $j_a \in \{1, ..., d_a(q)\}$ such that for all players $b \in \Lambda \backslash A$ whatever are their moves we have $\delta(q, j_1, ..., j_k) \in \rho$.

In order to translate a formula φ of an ATL model to the set of nodes Q' over which formula φ is satisfied, the attachment of specific actions to grammatical constructions within specification grammar of ATL is necessary.

The actions are written in target language of the generated parser (in our case, Java). These actions are incorporated in source code of the parser and are activated whenever the parser recognizes a valid syntactic construction in the translated ATL formula. In case of the algebraic compiler $\mathcal{C}$, the actions define the semantics of the ATL model checker, i.e., the implementation of the ATL operators.

The model checker generated by ANTLR from our specification grammar of ATL, takes as input the concurrent game structure S and formula φ, and

provides as output the set $Q' = \{q \in Q | q \models \varphi\}$ the set of states where the formula φ is satisfied.

The corresponding *action* included in the ANTLR grammar of ATL language for implementing the $\circ$ operator is:

```
'<<A>>@' f=formula
 {
   HashSet rez = Pre($f.set);
   $set = rez;
   trace("atlFormula",3);
   printSet("<<A>>@ " + $f.text,rez);
 }
```

For $\circ$ ATL operator in ANTLR we use the @ symbol.

The *formula* represents a term from a production of the ATL grammar, and *f*, *rez* variables are sets used in internal implementation of the algebraic compiler.

The $Pre(\$f.set)$ is a function that returns the set of states *rez* such that from each state of *rez*, the players in A can cooperate and enforce the next state to be in the set of states in which the formula f is satisfied.

The algebraic compiler $\mathcal{C}$ translates formula f of the ATL model S to set of nodes Q' over which formula f is satisfied. The implementation of the algebraic compiler $\mathcal{C}$ is made in two steps. First, we need a syntactic parser to verify the syntactic correctness of a formula *f*. Then, we should deal with the semantics of the ATL language (Sem_{atl}), respectively with the implementation of the operators from set $O_{atl} = \{\neg, \vee, \wedge, \rightarrow, \Diamond, \circ, \Box, U\}$.

Writing a translator for certain language is difficult to be achieved, requiring time and a considerable effort. Currently there are specialized tools which generate most of necessary code beginning from a specification grammar of the source language.

In figure 1 is represented the algebraic compiler implementation process, based on our specification grammar of ATL language.

4. Publishing the ATL model checker as a Web service

Web Services, as a distributed application technology, simplifies interoperability between heterogeneous distributed systems. Clients can access Web services regardless of the platform or operating system upon which the service or the client is implemented [5]. In order to make available our implementation of algebraic compiler as a reusable component of an ATL model checking tool, we published it as a Web service. The Web service will receive from a client the XML representation of an ATL model S and an ATL formula φ. The original form of the ATL model S is passed then to the algebraic compiler $\mathcal{C}$ generated by ANTLR using our ATL extended grammar. For a syntactically correct formula φ, the compiler will return as result $\mathcal{C}(\varphi) = \{q \in Q | q \models \varphi\}$, the set of states in which the formula is satisfied.

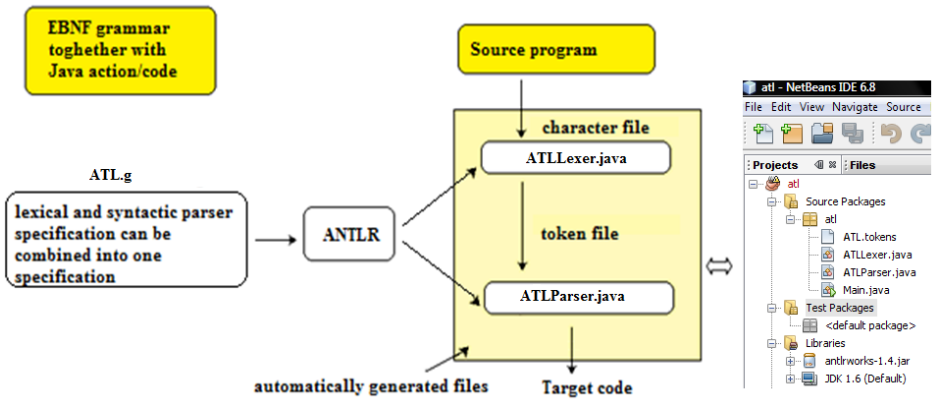

FIGURE 1. Algebraic compiler implementation

Obviously, the formula φ may contain syntactical errors. In order to notify the client about these possible errors, we must override the default behavior of the ANTLR error-handling.

The architecture of the ATL model checker Web Service is showed in figure 2.

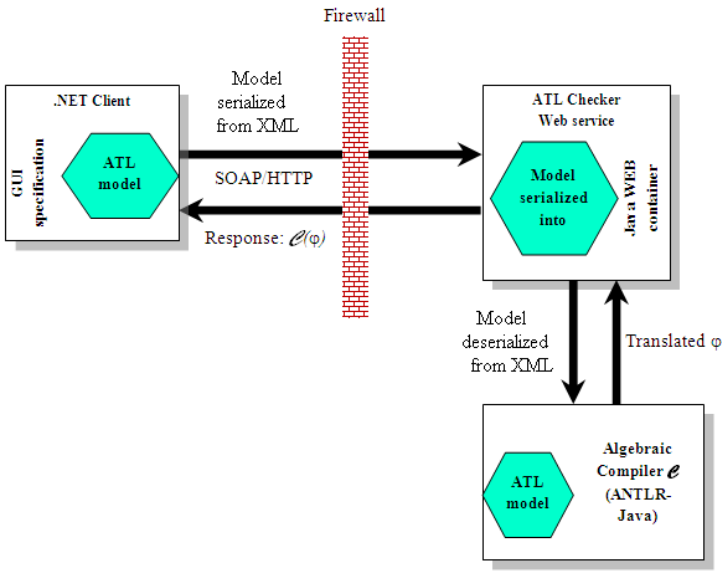

FIGURE 2. Architecture of the Web Service

In conclusion, for a given ATL model and an ATL formula φ, the Web service will parse the formula and will return to client the set of states in which the formula is satisfied if formula is syntactically correct, or a message describing the error from an erroneous formula.

5. Testing the new ATL model checker tool

Consider a system with two processes, Px and Py. The process Px assigns values to the Boolean variable x. When x=*false*, then Px can leave the value of x unchanged or change it in true. When x=*true*, then Px leaves the value of x unchanged. The process Py assigns values to the Boolean variable y, in the same way as the process Px.

We model the synchronous composition of two processes by the S_{xy} concurrent game structure, where $S_{xy} = \langle k, Q, \Pi, \pi, d, \delta \rangle$:

- k=2 (first player denoted process Px, second player denoted process Py);
- $Q = \{q_0, q_1, q_2, q_3\} - q_0$ means $x = y = false$, q_1 means $x = true$ and $y = false$, etc;
- $\Pi = \{x, y\}$;
- $\pi(q_0) = \emptyset, \pi(q_1) = \{x\}, \pi(q_2) = \{y\}, \pi(q_3) = \{x, y\}$;
- $d_1(q_0) = d_1(q_2) = 2$ (means in state q_0 and q_2 move 1 of the first player leaves the value of x unchanged, and move 2 changes the value of x); $d_1(q_1) = d_1(q_3) = 1$ (means in states q_1 and q_3 first player has only one move, namely, to leave the value of x unchanged); $d_2(q_0) = d_2(q_1) = 2, d_2(q_2) = d_2(q_3) = 1$;
- state q_0 has four successors: $\delta(q_0, 1, 1) = q_0, \delta(q_0, 1, 2) = q_2, \delta(q_0, 2, 1) = q_1, \delta(q_0, 2, 2) = q_3$.

A concurrent game is played on a state space. Every player chooses a move. The combination of choices determines a transition from the current state to a successor state.

The model checking tool is based on a *C Sharp GUI* client who allows interactive graphical development of the ATL models.

The model is sent as a XML document to the Web service, together with the formula to be verified. The response from server is displayed in a separate window, as we will see in the following section.

Given the ATL formula $\varphi = \langle\langle A \rangle\rangle \circ (x \wedge y)$ for game structure from figure 3 with $A = \{2\}$, the output of the model checker is $Q' = \{1, 3\}$. From state q_1 if second player chooses the move 2 the next state is q_3 whatever is the move selected by the first player. From the state q_2 for the move 1 of the second player, the first player can choose the move 1. Thus the game remains in state q_2. For that reason the state $2 \notin Q'$.

The answer from the server is showed into separate windows.

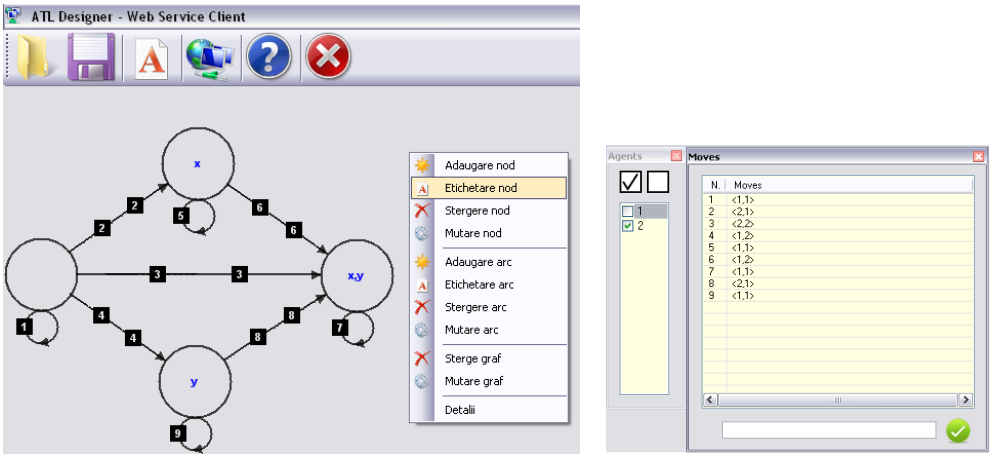

FIGURE 3. CTL Designer - the *C Sharp* client

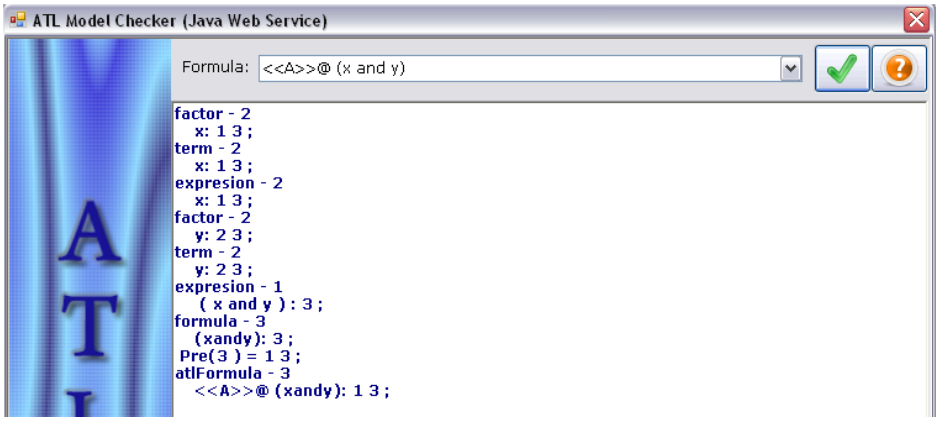

FIGURE 4. Invoking the algebraic compiler (Web service) from the ATL Designer (client)

6. CONCLUSIONS

For a given ATL model (concurrent game structure) and an ATL formula, the Web service will parse the formula and will return to client the set of states in which the formula is satisfied if formula is syntactically correct, or a message describing the error from an erroneous formula.

We built an ATL model checking tool, based on robust technologies (Java, .NET) and well-known standards (XML, SOAP, HTTP).

As a great facility we mention the capability of interactive graphical specification of the ATL model, using the client tool (ATL Designer).

References

[1] R. Alur, T. A. Henzinger , O. Kupferman, *Alternating-Time Temporal Logic*, Journal of the ACM, Vol. 49, No. 5, September 2002, pp. 672713.
[2] L. Cacovean, F. Stoica, *Algebraic Specification Implementation for CTL Model Checker Using ANTLR Tools*, 2008 WSEAS International Conferences, Computers and Simulation in Modern Science - Volume II, Bucharest, Romania, 2008, pp. 45-50.
[3] Terence Parr , *The Definitive ANTLR Reference, Building Domain-Specific Languages*, version: 2007.
[4] E. Van Wyk, Specification Languages in Algebraic Compilers, Theoretical Computer Science, 231(3), 2003, pp. 351385.
[5] Florian M. Boian, *Servicii Web; Modele, Platforme, Aplicatii*. Seria 245 PC, Editura Albastra, Cluj-Napoca 2011, ISBN 978-973-650-266-8.

"Lucian Blaga" University of Sibiu, Faculty of Sciences, Department of Mathematics and Informatics, 5-7 Dr. Ion Ratiu St, 550012, Sibiu, Romania
E-mail address: laura.cacovean@ulbsibiu.ro

Babes-Bolyai University of Cluj-Napoca, Faculty of Mathematics and Informatics, Department of Computer Science, 1 M Kogalniceanu St, 400084, Cluj-Napoca, Romania
E-mail address: florin@cs.ubbcluj.ro