INTERACTIVE AUTOMATA IN XML AND CLIENT-SERVER ENVIRONMENTS

[1]MIRNA EL-HAJJ BARBAR, [2]KABLAN BARBAR, [3]GEORGES RAHBANI

[1]Assoc Prof., Department of Applied Mathematics, Faculty of Economy, Lebanese University, Achrafieh, Lebanon

[2] Assoc. Prof., Department of Applied Mathematics, Faculty of Sciences 2, Lebanese University, Fanar, Lebanon

[3] Assoc. Prof., Department of Applied Mathematics, Faculty of Sciences 2, Lebanese University, Fanar, Lebanon

E-mail: mirna_hajj@hotmail.com, kbarbar@ul.edu.lb, grahbani@ul.edu.lb

ABSTRACT

In this paper, we introduce the new concept of interactive automata by proposing that states are not virtual elements but real objects containing useful information. They are used as models for structured data and system architecture, and then implemented in two developing environments, the XML environment and the client-server environment. These implementations produce, respectively, a generic markup language called Automata Markup Language (AML) and a client-server system, which together realize a homogeneous and uniform platform. An example is given in the E-learning domain where AML is a questionnaire description language and the automata client-server system plays the role of an assessment platform.

Keywords: *Automata, XML, Client-Server, E-learning, Assessment.*

1 INTRODUCTION

In this paper, a new concept of interactive automata is introduced, proposing that states contain useful information and are not virtual elements. Interactive automata are implemented in terms of markup languages and client-server systems in order to build a homogeneous and client-server platform applicable in many domains and, in particular, the E-learning domain.

Typically, automata formalism has been introduced to formalize rational languages in the compilation of programming languages (see [1]). Based on the principle of transitions over states, automata resolve the problem of belongings of words to rational languages. They are viewed as abstract machines where states are virtual elements and not related to the recognized language; and they are implemented by simple programs running on one computer. Many applications of automata have been developed in different areas as formalization model. Pavlos Antoniou et al. in [8] present an algorithm that uses finite automata to find the common motifs with gaps occurring in all strings belonging to a finite set. Constant and Maurel in [7] describe a unified method to compile sets of tables of linguistic constraints into Finite State Automata. In [2], the author proposes a helicopter flight simulator with respect to automata that helps to train helicopter pilots. In [3], automata serve to construct efficient algorithms for the adaptive assessment of student knowledge.

We attempt to use automata as data models and system architectures for client-server platforms. Automata transition functions are, as such, control machines on the server side to present to clients information stored in the states' structure. Therefore, states are not virtual elements in automata, but basic objects containing the input symbols needed for making transitions between states. When an automaton is in a current state, its input symbol is chosen dynamically, and then automata are interactive control machines to ensure only the passage through states. This process is different from the automata application in the compiling domain while input words are given statically and automata are recognition machines for verifying if input words belonging to the automata recognized languages.

From this point of view, we introduce interactive automata by integrating an alphabet in the states' structure and considering them both a data model and an interactive machine. First, interactive automata are translated into a DTD in XML for producing a generic markup language called Automata Markup Language or AML. Each AML document describes an interactive automaton. Secondly, interactive automata are considered as control machines and implemented in the client-server environment. The corresponding system runs over AML documents. This system is split into two parts: the client part and the server part. The client part concerns the states' presentation on the client side, which can be made by XSL style sheets in the XML environment. The server part contains two modules: the initialization module and the transitions module. The initialization module extracts the initial state from an AML document and sends it to the client; while the transitions module implements the automata transition function that takes an input from the client for the current state, determines the following state and sends it again to the client.

The advantage of interactive automata is that states are real objects and not virtual elements. This principle allows the integration within the states' structure of other characteristics of the states' basic objects. In particular, if the basic objects have temporal characteristics, they must be integrated in the states' structure.

Throughout the paper, an application of interactive automata in the assessment domain is illustrated. The AML language describes questionnaires' structure and the client-server assessment system implements the automata transition function (see [4]). In the same manner, interactive automata have been used for modeling the learning process in [5], and an E-learning architecture platform has been produced in which the learning and assessment layers run as automata client-server systems.

We have realized an assessment platform based on automata client-server system that acts over a directory of AML documents corresponding to questionnaires. This platform was developed under XML and ASP technologies.

This paper is organized in the following way. Section II introduces interactive automata. Section III presents the Automata Markup Language and the client-server system associated with interactive automata. Section IV presents the automata based prototype for an assessment platform. Finally, the conclusion discusses the results obtained and the advantages of this approach.

2 INTERACTIVE AUTOMATA

Our aim is to adapt automata formalism to the client-server environment where automata are considered as server systems that present states to clients. Then, states are not virtual elements but contain real data or useful information presented to clients. From this point of view, an alphabet is not only a set to define automata languages but it is also the basic of the states.

Let Σ represent an alphabet, and D represent a set of basic objects. The states domain is then $D \times P(\Sigma)$ where $\times$ is the Cartesian product and P is the power set. An interactive automaton IA over D and Σ is a 5-uplet IA=$<\Sigma, S, s_0, S_t, \delta>$ where:

Σ is the alphabet,

- $S \subseteq D \times P(\Sigma)$, is the set of states,
- s_0 is the initial state and an element of S,
- S_t is the set of terminal states and a subset of S,
- δ is the transition function:

$$\delta : S \times \Sigma \rightarrow S.$$

Example: We may consider the example of a questionnaire on the comprehension of the concept of a "pointer" in the programming language. This is explained in [6]. Each test is formed by a question and a list of answers. The list of tests is presented in table 1.

We consider a walking strategy within tests that works in the following way:

- if the learner's answer to the test t_0 is wrong (response r_3), he or she is sent into a loop of tests (tests t_2, t_3 and t_0),
- if the learner's answer to the test t_0 is quite correct (response r_2), he or she is given a chance (test t_1) to advance.

This strategy is represented by the graph in figure 1 where the symbol | is the or logical operator.

Table 1. A questionnaire

Tests	Questions	Answers
t_0	q_0 : what is the best definition of a pointer ?	r_1 : a pointer is a variable r_2 : a pointer is an address r_3 : a pointer is a function
t_1	q_1 : what is the type of a pointer variable?	r_4 : a memory address r_5 : it depends on the pointed object type
t_2	q_2 : can we use a variable by specifying its location in memory?	r_6 : yes r_7 : no
t_3	q_3 : can we store one variable in another?	r_8 : yes r_9 : no
t_f	q_f :	

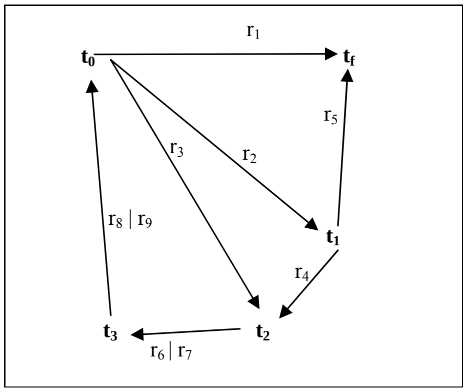


Figure 1. The graph of the questionnaire

The preceding questionnaire can be represented by an IA. The alphabet is $\Sigma=\{r_0, \dots , r_9\}$, the set of answers. The set of basic objects is $D=\{q_0,q_1,q_2,q_3,q_f\}$, the set of questions. Then, the states' domain is $\{q_0,q_1,q_2,q_3,q_f\}\times P(\{r_0,\dots,r_9\})$. For example, t_0 is the pair $(q_0,\{r_1,r_2,r_3\})$ and so on. The interactive automaton corresponding to the questionnaire above has the set of basic objects D as a domain, which is as follows: $IA=<\Sigma,T,t_0,T_f,\delta>$

where :

- $\Sigma = \{r_0, \dots , r_9\}$, is the input alphabet,
- $T = \{t_0,t_1,t_2,t_3,t_f\}$ is the set of states,
- t_0 is the initial state,
- $T_f= \{t_f\}$ is the set of terminal states,
- δ is the transition function: $\delta : T \times \Sigma \rightarrow T$, given in table 2.

Table 2: A transition function

Function δ	**t_0**	**t_1**	**t_2**	**t_3**	**t_f**
r_1	t_f				
r_2	t_1				
r_3	t_2				
r_4		t_2			
r_5		t_f			
r_6			t_3		
r_7			t_3		
r_8				t_1	
r_9				t_1	

Proposition: There is an equivalence between classical automata and interactive automata.

Proof: It is obvious by the definition that an IA is a classical automaton. On the other hand, let $A=<\Sigma,S,t_0,S_t,\delta>$ be a classical automata. Let s be a state, and we will call Trans(s))=$\{\sigma\in\Sigma$ / $\delta(s,\sigma)$ is defined$\}$ the set of all elements in Σ defining transitions over the state s. The interactive automaton corresponding to A is then $IA=<\Sigma,S',t_0,S'_t,\delta'>$ where :

- $S'=\{(s,Trans(s))/s\in S\}$ and $\delta'((s,Trans(s)),\sigma)= (\delta(s,\sigma),Trans(s,\sigma))$
- $s'_0=(s_0,Trans(s_0)$
- $S'_t=\{(s,Trans(s))/s\in S_t\}$
- $\delta : S' \times \Sigma \rightarrow S'$ is defined by :
  for $s'=(s,Trans(s)) \in S'$ and $\sigma\in\Sigma$,

$$\delta'(s',\sigma)=\begin{cases}(\delta(s,\sigma),Trans(\delta(s,\sigma)) & \text{if } \sigma\in Trans(s)\\ \text{undefined} & \text{otherwise.}\end{cases}$$

The function b: S $\rightarrow$ S' defined by b(s)=(s,Trans(s)) is a bijection. Moreover, for each transition $\delta(s,\sigma)=s1$, there exists a transition $\delta'(b(s),\sigma)=b(s1)$ and vice versa. We may then conclude that there exists an interactive automaton equivalent to every classical automaton and vice versa.

3 IMPLEMENTATION

Since interactive automata are considered at the same time as data structure model and system architecture, their implementation is made, respectively, in the XML data structuring and the client-server environments.

3.1 The Automata Markup Language

Interactive automata are built on alphabets and basic objects. Therefore, interactive automata represent an abstract data model that can be implemented in terms of languages. To do this, interactive automata are translated into a DTD in the XML environment. Each interactive automata component is associated with a declaration in the DTD. This translation produces an Automata Markup Language (AML) where a valid AML document represents an interactive automaton. AML is a generic language and contains a part common to all interactive automata, and has a variable part which depends on the alphabet and the set of basic objects. The AML DTD is described below:

```
<!-- The automaton -->
<!ELEMENT automaton(sigmas, states,
          initialState, terminalStates, delta>
<!ELEMENT sigmas(sigma*)>
<!ATTLIST sigma id ID #REQUIRED>
<!ELEMENT states (state*)>
<!ATTLIST state id ID #REQUIRED>

<!-- The initial state -->
<!ELEMENT initialState (EMPTY)>
<!ATTLIST  initialState idState IDREF
                #REQUIRED>

<!-- The set of terminal states -->
<!ELEMENT terminalStates (terminalState*)>
<!ELEMENT terminalState EMPTY>
<!ATTLIST terminalState idState IDREF
          #REQUIRED>

<!-- The transition function delta-->
<!ELEMENT delta (transition*)>
<!ELEMENT transition (EMPTY)>
<!ATTLIST  transition originState IDREF
          #REQUIRED>
<!ATTLIST  transition targetState IDREF
          #REQUIRED>
<!ATTLIST  transition label IDREF
          #REQUIRED>.
```

The common part is the kernel of the AML language. Each real application must give the definition of the elements <sigma> and <state> for the variable part.

Let's take an example from the assessment domain. A questionnaire is a list of tests that are composed of questions and answers. To describe a questionnaire, we add to the DTD of AML the declaration of the variable part, which is the following sequence:

```
   <!ELEMENT sigma(#PCDATA)>
   <!ELEMENT state(question, answers)>
   <!ELEMENT question(#PCDATA)>
   <!ELEMENT answers(answer*)>
   <!ELEMENT answer(EMPTY)>
   <!ATTLIST  answer idSigma IDREF
                    #REQUIRED>.
```

The questionnaire of example1 is then translated into the following AML document:

```
<!-- The interactive automaton -->
<automaton>

<--The alphabet -->
<sigmas>
     <sigma id="r1"> a pointer is a
        variable</sigma>
     <sigma id="r2"> a pointer is an
        address</sigma>
     <sigma id="r3"> a pointer is a
        function</sigma>
     …
     <sigma id="r9">false</sigma>
<sigmas>

<--The set of states -->
<states>
```

```
 <state id="t0">
   <question id="q0"> what is the best definition
          for a pointer?</question>
   <answers>
     <answer idSigma="r1"/>
     <answer idSigma="r2"/>
     <answer idSigma="r3"/>
   </answers>
 </state>
   …
 <state id="tf">
   <question id="qf"></question>
   <answers></answers>
 </state>
</states>

<!-- The initial state-->
 <initialState   idState="t1"/>

<!-- The set of terminal states -->
 <TerminalStates>
    <terminalState   idState="tf"/>
</terminalStates>

<!-- The transition function -->
<delta>
   <transition origineState="t0"  label="r1"
           targetState="tf">
   <transition origineState="t0"  label="r2"
           targetState="t1">
   <transition originState="t0"  label="r3"
           targetState="t2">
           …
</delta>
</automaton>
```

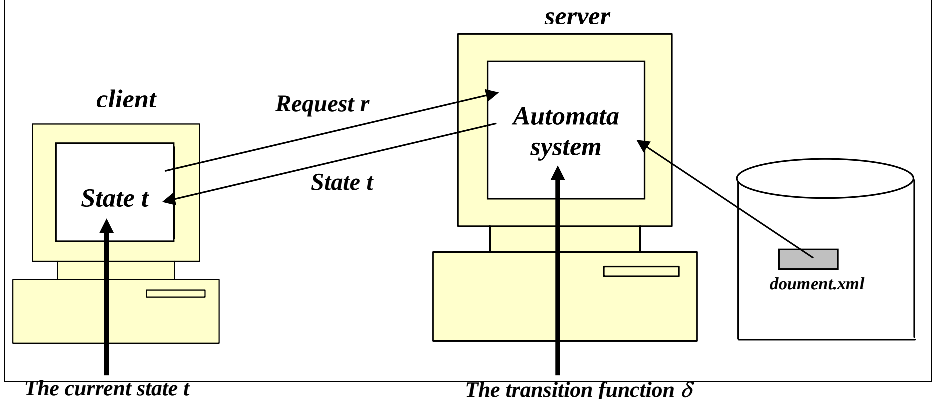
3.2 The automata client-server system

In the client-server architecture, an automata based platform comprises an automata client-server system and a directory of AML documents stored on the server disk. Then, to realize client-server systems, interactive automata are split into two parts where the transition function is the kernel of the system on the server side and only states must be presented on the client side as shown in Figure 2. The interaction in the client-server environment consists in two steps:

- step1 : the client sends a request, a label in the automata alphabet, to the server
- step2 : the server responds through a state sent to the client.

The automata client-server system has two parts, the client part and the server one. The client part deals with the states' presentation. Since states are implemented in XML, the states presentation is realized by XSL style sheets. The server part concerns the management of AML documents. It extracts the initial state from an AML document, presents it to the learner, takes the learner's request and makes a transition from the current state to a following one according to the request.
It is then composed of two important server modules, called initialization and transitions.

Figure 2. Interactive Automata in Client-Server Architecture



.

The 'initialization' module realizes the following operations:

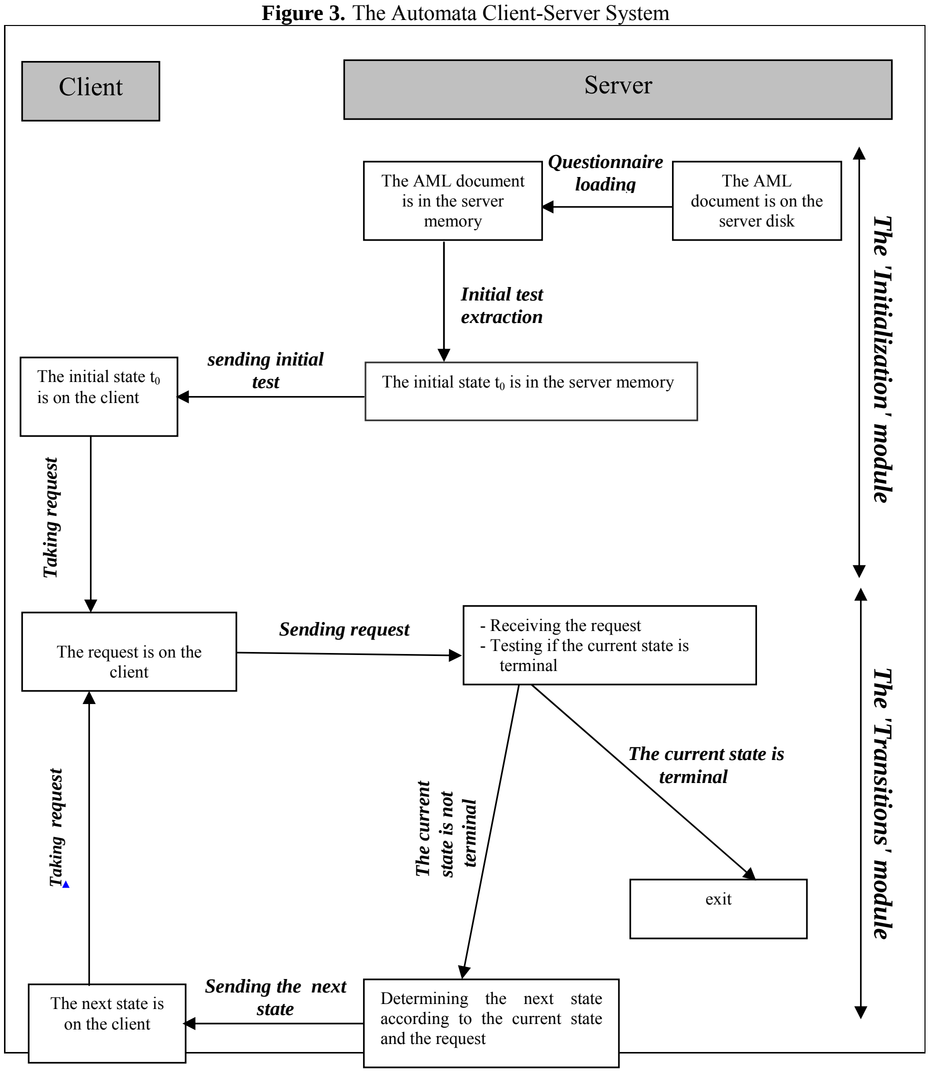
- o loads the AML document into the server memory,
- o extracts the initial state from the AML document,
- o presents the initial state to the client.

The 'Transitions' module conducts the following operations:

- o determines the next state in accordance with the current one and the client request
- o presents the next state to the client.

In Figure 3, all details for the two modules of *initialization* and *transitions* are given and distributed in the client-server architecture.

Figure 3. The Automata Client-Server System


Client
Server
The AML document is in the server memory
Questionnaire loading
The AML document is on the server disk
Initial test extraction
The initial state t_0 is on the client
sending initial test
The initial state t_0 is in the server memory
The 'Initialization' module
Taking request
The request is on the client
Sending request
- Receiving the request
- Testing if the current state is terminal
The current state is terminal
The current state is not terminal
exit
Taking request
The 'Transitions' module
The next state is on the client
Sending the next state
Determining the next state according to the current state and the request

Formatted: Font: Bold

4 THE PROTOTYPE

We have realized an assessment platform based on automata formalism. This platform is comprised of a directory of AML documents on the server disk and its kernel is an automata client-server application developed under the ASP technology. The AML documents represent questionnaires based on tests for networking. The automata client-server application loads a questionnaire (or AML document) in the server memory, extracts a test and presents it to a learner. It takes the learner's responses and determines the next test and so on. Therefore, it runs like an automata transition function where the test presented to a leaner plays the role of an automaton state.

5 CONCLUSION

In this paper, we have presented interactive automata and their adaptability for the conception of a client-server platform. Essentially, interactive automata use states to store useful information and implement them in the XML and client-server environments. Concrete applications of interactive automata have been given in the E-learning domain for assessment and learning processes.
Interactive automata allow the integration within the states' structure of many characteristics, like the timing control in the assessment process when learners have to respect a fixed duration for each test. But, in general, we have to classify characteristics and specify how to integrate them in the states, with automata output function, or with other automata extensions.

We are also working to develop a communication language between interactive automata based on the principle of messages. This language simplifies the realization of modular and automata-based platforms.

REFERENCES

[1] AHO, A., SETHI, R. & ULLMAN, J., "Compilers, Principles, Techniques and Tools", ADDISON-WESLEY, 2006.

[2] Belhadj, F. "Drones: Simulateur d'Environnement et Apprentissage", Des Quinzièmes Journées de L'A.F.I.G., Université Claude Bernard, LYON 1, December 2002.

[3] C. E. Dowling, & C. Hockmeyer., "Automata for the Assessment of Knowledge", IEEE Transactions on Knowledge and Data Engineering, vol 13, n° 3, may/june 2001.

[4] El-Hajj Barbar, M., Barbar, K., Monsef, Y., & Saleh, I., "A Three-level Model for Educational Process in a Client/Server Environment". ITHET04, Istanbul, June 2004.

[5] El-Hajj Barbar, M., Barbar, K., Monsef, Y., & Saleh, I., "Automata based Architecture for Layered E-learning Platforms". ITHET05, Sainto Domingo, July 2005.

[6] Issac, F. & Hû, O., "Formalism for Evaluation: Feedback on Learner Knowledge Representation", Computer Assisted Language Learning Volume 15, Number 2, pp 183 – 199, April 2002.

[7] Constant. M, and Maurel. D, 'Compiling Linguistic Constraints into Finite State Automata', Lecture Notes in Computer Science, Vol 4094/2006, PP. 242-252, August 2006

[8] Pavlos Antoniou, Jan Holub, Costas S. Iliopoulos, Bořivoj Melichar and Pierre Peterlongo 'Finding Common Motifs with Gaps Using Finite Automata', Lecture Notes in Computer Science, Vol 4094/2006, PP. 69-77, August 2006

[9] Mohri. M, 'On some Applications of Finite-state Automata Theory to Natural Language Processing', Natural Language Engineering, Vol 2, Issue 1, PP. 61-80

[10] Sipser. M, Introduction to the Theory of Computation, Second Edition, CENGAGE LEARNING, 2005