E-learning tools based on transition systems

DAN CHIRIBUCĂ, IULIAN PAH  
Department of Sociology,  
Babes-Bolyai University  
Cluj-Napoca, Bd.21 decembrie  
1989, no.128-130, 400604,  
Cluj-Napoca, ROMANIA

EMIL M. POPA, BOGDAN A. BRUMAR  
Department of Computer Science  
Faculty of Sciences  
Lucian Blaga University of Sibiu  
Str. Ion Ratiu, no. 5-7, 550012  
Sibiu, ROMANIA

Abstract: - Due to the dynamic nature of the computing objects represented, the semantics of the specification languages used to formalize systems will be a transition system performing computation actions, while the syntax will be the linguistic expression of the actions performed by the transition system. Computing objects manipulated in computer science bear a lot of similarities with the mathematical objects produced by the systems discussed so far they also are very different. The concept of language is regarded as a communication tool that allows language users to develop knowledge, while interacting with their universe of discourse, and to communicate with each other, while exchanging knowledge, in that case for e-learning systems. The results presented here have as initial point [1], [7], [13].

Key-Words: - system implementation language, system specification, system validation language, transition system, macro-operations.

1 Introduction

Systems represent computations with specific behavioral properties the major steps involved in this formalization are:

- **System specification**: defines the system as a computing object of the form **System** = <Specification, Behavior>.
- **System implementation**: expresses the behavior of the system such that it can be observed.
- **System validation**: shows that the behavior of the system has the requested properties. Each of these steps requires a specific language for its representation. A language in this context is a tuple L = <Sem, Syn, L:Sem→Syn> where:
  - Sem is the language semantics;
  - Syn is the language syntax;
  - L:Sem→Syn is a partial mapping that expresses computing objects c ∈ Sem by means of their linguistic expressions L(c) ∈ Syn in a way that there exists a total mapping ε : Syn→Sem such that ε(L(c)) = c whenever L is defined.

Due to the dynamic nature of the computing objects represented, the semantics of the specification languages used to formalize systems will be a transition system performing computation actions, while the syntax will be the linguistic expression of the actions performed by the transition system.

2 Transition systems

A transition system is a tuple **T** = <Π, Σ, T, Θ> where:

- Π is a set of typed symbols called the state variables. There are two kinds of variables in Π: data variables used to denote data values and control variables used to denote control values.
- Σ is a set of states. Each s ∈ Σ is an assignment of the variables in Π, i.e., s : Π→ D, where D is the domain of values of the variables in Π. If u ∈ Π then s[u] denotes the value assigned to u by s. The relation s:Π→ D can be uniquely extended to the valuation homomorphism V_s : W(Σ, Π) → A(Σ, D) where Σ is an operator scheme, W(Σ, Π) is the term generated by Σ in symbols of Π, and A(Σ, D) is the values generated by Σ on the domain D. If e ∈ W(Σ, Π) and s:Π→ D is an assignment then s[e] denotes V_s(e). If φ is an assertion and V_s(φ) = true then we write s |= φ and call s a φ-state.
- T is a finite set of transitions. Each τ∈T is a mapping τ : Σ → P(Σ) where P(Σ) is the power set of Σ. That is, for each s∈Σ, π(s) ⊆ Σ (π(s) can be the∅). If π(s)≠∅ and s′ ∈ π(s) then s′ is called a τ-successor of s. In addition, we assume that there is τ∈T such that for each
\[ s \in \Sigma, \pi(s) = \{ s \}. \] We call \( \pi \) the idling transition. The transitions in the set \( T_\pi = T \setminus \{ \pi \} \) are called diligent.

- \( \Theta \) is an assertion called the initial condition of \( T_\pi \).
- A state \( s \in \Sigma \) such that \( s \models \Theta \) is called an initial state of \( T_\pi \).
- Each transition \( \tau \in T_\pi \), is characterized by an assertion relating the values of the variables in states \( s : \Pi \rightarrow D \) and \( s' : \Pi \rightarrow D \) for each \( s' \in \pi(s) \). Denote this relation by \( \rho_\tau \).

formally, if \( s : \Pi \rightarrow D \) and \( s' \in \pi(s) \), then \( \forall u \in \Pi (s[u], s'[u]) \in \rho_\tau \). Using infix notation this is expressed by \( s[u] \rho_\tau s'[u] \). The assertion \( \rho_\tau \) allows us to refer to the values of the variables \( u \in \Pi \) as \( u \), the value \( u \) before the transition, and \( u' \), the value after the transition \( \tau \). \( u' \) is called the primed version of \( u \). That is, we may think of two copies \( \Pi \) and \( \Pi' \) of the variable of \( T_\pi \), and denote the transition relation of \( \tau \) by \( \rho_\tau(\Pi, \Pi') \). The transition relation \( \rho_\tau(\Pi, \Pi') \) has the form \( \rho_\tau = C_\Lambda(\Pi) \land (y_i = e_i) \land \ldots \land (y_i = e_k) \) where:

- \( C_\Lambda(\Pi) \) is an assertion called the enabling condition that depends only on the values of the variables in the state \( s \) before the transition. \( C_\Lambda(\Pi) \) states the condition under which \( s \) may have a \( \tau \)-successor, i.e., \( \pi(s) \neq \emptyset \).
- \( (y_i = e_i) \land \ldots \land (y_i = e_k) \) is the conjunction of the modifications performed by \( \tau \) when it takes place. Each \( y_i = e_i \) requires that the primed value of the variable \( y_i \) to be computed using the non-primed value of \( e_i \), i.e., \( \forall s' \in \pi(s)(s'[y_i] = s[e_i]) \), where \( y_1, y_2, \ldots, y_k \) are pair wise distinct.

Notation: The transition relation \( \rho_\tau(\Pi, \Pi') = C_\Lambda(\Pi) \land (y_i = e_i) \land \ldots \land (y_i = e_k) \) can be denoted by \( \rho_\tau : C_\Lambda \land (\tilde{y}' = \tilde{e}) \) where \( \tilde{y}' = (y_1, y_2, \ldots, y_k) \) and \( \tilde{e} = (e_1, e_2, \ldots, e_k) \). If \( \rho_\tau(T, \Sigma) \) and \( \pi(s) \neq \emptyset \) then we say that \( \tau \) is enabled on \( s \) and if \( \pi(s) = \emptyset \) we say that \( \tau \) is disabled on \( s \). Notice that if \( \rho_\tau : C_\Lambda \land (\tilde{y}' = \tilde{e}) \) then \( \tau \) is enabled on \( s \) if \( s \models C_\Lambda \). For a set of transitions, \( \tau \subset T \) and \( \rho_\tau(\Pi, \Pi') \) we say that \( \tau \) is enabled on \( s \) if \( s \models C_\Lambda \). For a set of transitions, \( \tau \subset T \) and \( \rho_\tau(\Pi, \Pi') \) we say that \( \tau \) is enabled on \( s \) if \( s \models C_\Lambda \). For a set of transitions, \( \tau \subset T \) and \( \rho_\tau(\Pi, \Pi') \) we say that \( \tau \) is enabled on \( s \) if \( s \models C_\Lambda \).

The pair \( (s_i, s_{i+1}) \) is called a \( \tau \)-computation step, or simply \( \tau \)-step.

- **Diligence**: either \( \sigma \) contains infinitely many diligent \( \tau \)-steps for \( \tau \in T \) or it contains a terminal state. Since \( \tau \)-steps of a terminal state leave that state terminal, a computation that contains a terminal state is called terminal.

### 3 System Specification

The universal language emerges into a specification language that is widely accepted in computer science. The specification language borrows form computer science the idea of using keywords in order to relate it to the natural language of its users. This language has already penetrated the field of computer science under the name of abstract data types.

An **action** specification is a program in the language used to express reactive systems and is provided in the specification by the keyword **Actn**. An action consists of two parts, the name, and the linguistic expression of the action. The name of the action is separated by double colon, ::, from its linguistic expression. The linguistic expression of the action is composed of a declaration part and an action part. Formally, an action specification is a linguistic expression of the form \( A :: [D][A_1 \parallel A_2] \ldots \parallel A_n \) where the following notation is used:

- \( A \) is the name of the action performed by the system.
- \( D \) is a sequence of typed lists of variables of the form **mode** List : **type** where **phi** where **mode** is one of **in**, **out**, **inout**, **local**, **type** is a type of value accepted in the system, and **phi** is an assertion satisfied by the variables in the List.

- \( A_1, A_2, \ldots, A_n \) are actions in terms of which the action \( A \) is specified. Each \( A_i \) is either a call to a previously defined action, or has the form \( [D_i] \); \( S \), where \( D_i \) is a declaration and \( S \) is a statement describing the action to be performed on the variables in \( D \cup D_i \). When \( A_i \) is an action call, its expression in \( A \) is \( A_i(arg) \) where \( arg \) is the list of variables used by \( A_i \) for its task. Arguments can be: **in**, **out**, **inout**. The **in** arguments are imported and not modifiable, the **out** arguments are exported and **inout** arguments are imported and modifiable.

**Definition 2.** A process is a tuple \( P = \langle Agent, Action, Status \rangle \) where **Agent** is a processor capable to perform statements composing the actions in **Action** and **Status** is the state of this performance.

In order to perform the statements of an action, the processor has a control mechanism that shows the label of the statement currently executed. Denote this control by \( \pi \). Statements are simple or composed. Each statement has the form \( l : body : I \), where \( l \) is a label that
identifies the statement by showing the entry point in the statement body and  is a label showing the exit point from the statement body. The simple statements are performed by the processor atomically. There are three types of simple statements in a system specification. They are called skip, await and assignment and are defined as follows:

✓ The skip statement has the form \( l : \text{skip} : \hat{i} \) and its performance means “skip”.

✓ The await statement has the form \( l : \text{await} \ e : \hat{i} \) where \( e \) is a boolean expression and its performance means “wait until \( e \) becomes true”.

✓ The assignment statement has the form \( l : (x_1, x_2, \ldots, x_n) := (e_1, e_2, \ldots, e_n) : \hat{i} \) denoted by \( l: \vec{x} := \vec{e} : \hat{i} \), where \( \vec{x} = (x_1, x_2, \ldots, x_n) \), \( \vec{e} = (e_1, e_2, \ldots, e_n) \), and for \( l = 1, 2, \ldots, n \), \( x_i \) and \( e_i \) have the same type. This is also called a multiple assignment.

The composed statement of the specification language is concatenation, branch, loop, choice, parallel and block. The statement composition generates redundant labels. To simplify this we group together all redundant labels in equivalence classes. A class of equivalence contains all labels that denote the entry point or the exist point of a statement. Each equivalence class is represented by one label. That is, we assume that each statement \( S \) has just one entry point and one exit point. The set of labels denoting the entry point of \( S \) is \( \text{Entry}(S) \) and the set of labels denoting the exit point of \( S \) is \( \text{Exit}(S) \).

The labeling of statements is however optional. The entry and exit points of a statement that has no labels are considered to be the empty string \( \varepsilon \). This language of actions can be freely extended in order to express various computation performed by different systems.

### 4 System specification language

Formally, a system can be defined as a pair \( \text{System} = (\text{TS}, A) \) where \( \text{TS} \) is a transition system and \( A \) is an action expression specifying the computations performed by the \( \text{TS} \). Using the systematic approach for a system construction we develop a system by successive iterations. At each iteration we construct a version of \( \text{TS} \) and then express it by an appropriate action \( A \) to obtain a process \( P = (\text{Agent}, A, \text{Status}) \). This process when active performs the computations specified by \( \text{TS} \). The formalization of this specification approach leads to a System Specification Language, \( \text{SSL} \), where \( \text{SSL} = (\text{SSL}_{\text{Sem}}, \text{SSL}_{\text{Syn}}, \text{L} : \text{SSL}_{\text{Sem}} \rightarrow \text{SSL}_{\text{Syn}}) \) where:

✓ The semantics \( \text{SSL}_{\text{Sem}} \) of the \( \text{SSL} \) are transition systems.

✓ The syntax \( \text{SSL}_{\text{Syn}} \) of the \( \text{SSL} \) are actions.

✓ The function \( L : \text{SSL}_{\text{Sem}} \rightarrow \text{SSL}_{\text{Syn}} \) is determined by the process that allows us to express transition systems by actions. The language evaluation function \( \varepsilon : \text{SSL}_{\text{Sem}} \rightarrow \text{SSL}_{\text{Syn}} \) is defined as follows: if \( A ::= [D][A_1 \parallel A_2 \ldots \parallel A_n] \) is an action in \( \text{SSL}_{\text{Syn}} \) then \( \varepsilon (A) \) is the transition system \( T_S A = (\Pi_i, \Sigma_i, T_A, \Theta_A) \) in \( \text{SSL}_{\text{Sem}} \) constructed as follows:

- \( \Pi_i \) is the set of all variables declared in \( A \) together with a control variable \( \pi \) that runs over the power set of the collection of labels \( L_A \) in \( A \).

- Each \( s \in \Sigma_i \) is an assignment \( s : \Pi_i = D_A \) where \( D_A = \bigcup_{x \in \Pi_A} \text{Type}(x) \cup L_A \).

- \( T_A \) is the set of transitions determined by the statements of \( A \). The idling transition is \( \rho : T \).

- The initial condition of \( \Theta_A \) is \( \Theta_A = \{ \pi = \text{Entry}(A_1) \cup \ldots \cup \text{Entry}(A_n) \} \) where \( \phi \) is the conjunction of all \( \text{where} \) assertions in the expression of \( A \).

A computation performed by the transition system \( T_S A \) is expressed by the sequence of transition \( \langle \pi, x_1, \ldots, x_n \rangle \xrightarrow{\rho_{S_1}} \langle \pi, x_1, \ldots, x_n \rangle \xrightarrow{\rho_{S_2}} \ldots \) where \( \{ \pi, x_1, \ldots, x_n \} = \Theta \) and \( S_1, S_2, \ldots \) are statements of the action performed by \( T_S A \).

An important question in our study is to determine when two computing objects are equivalent. We call two computing objects equivalent if the transition systems performing their behavior are equivalent. Two transition systems \( T_S A_1 \) and \( T_S A_2 \) are equivalent when they generate the same set of computations. Since every computation of a transition system in an infinite sequence of state transitions where the entire state is seen, this concept of equivalence is too discriminating. So, the equivalence of transition systems should be defined up to a set of state variables that are observable. The set of observable state variables should be specified by the user. That is, the computations generated by two transitions systems are considered to be the same if the values of the observable state variables are the same. Consequently, one can define the reduced behavior of a transition system to be the set of its computations where only the values taken by the observable variables are seen. If a reduced state is the observable part of that state then the reduced behavior \( \sigma' \) of a computation \( \sigma \) is determined as follows:

1. Replace each state \( s_i \) of \( \sigma \) by its observable part containing \( s_i \) to the observable variables.
2. Omit from the sequence of states of \( \sigma \) each state that coincides with its predecessor but differs from its successor.
The reduced behavior of a transition system $TS$ with respect to a given set of observable variables $O$ is denoted by $\mathcal{R}(TS, O)$.

Since actions are specified in terms of other actions this concept of equivalence should detect the situations where two actions can be used interchangeably. Let us assume that the variable $S$ runs over actions. Denote an action that depends on $S$ by $A(S)$ and by $A(A_1)$ the action that is obtained from $A(S)$ by replacing all occurrences of $S$ in $A$ by $A_1$. Then two actions $A_1$ and $A_2$ are called congruent, denoted by $A_1 = A_2$, if $TS_{S_1} = TS_{S_2}$ for every action $A(S)$. Example of congruent actions are provided by the associatively of the operators ";" (concatenation), “or” (choice), “||” (parallel) used to construct composed statements.

There are two relations among actions $A_1$, $A_2$ that allow the replacement of $A_1$ by $A_2$, emulation and implementation. Such a replacement is desirable when $A_2$ is expressed in terms of language constructs that can be run on a given computer. The action $A_1$ emulates the action $A_2$ if $\mathcal{R}(TS_{A_1}) = \mathcal{R}(TS_{A_2})$. The action $A_2$ implements the action $A_1$ if $\mathcal{R}(TS_{A_2}) \subseteq \mathcal{R}(TS_{A_1})$.

5 System implementation language
To achieve its goal, the objects and the operations used to specify a system should behave as the data and operations of an abstract machine which performs the computation task specified by the system. In other words, the system expression written in the system specification language should be transformed into a computation object of an abstract machine. The abstract machine used to express computing objects is a programming language implemented on an actual computer.

The programming language that allows us to express systems as computation objects is called the system implementation language. The process of mapping the system specification language into the system implementation language is called the system implementation. We use the C programming language as the system implementation language. C is regarded here mere as a tool for the software system designer.

6 System validation language
The validation of a system is the process of showing that the system performs the function for which it was designed. The validation process consists of actually using the system in appropriate applications according to the function that it performs.

The language used to express applications which use a system in order to validate the system is called the system validation language. Usually a system is validated using the system implementation language. Therefore, we use C as the software system validation language.

7 Communication tools for e-learning
E-learning systems contain plenty of communication tools, which enable teachers, communicate individually with selected students. These tools enable more intensive communication than in classical teaching, after a very precise specification of transition systems with the three steps for formalization: system specification, system implementation language and system validation. There are many languages and instruments to create the necessarily software’s. In this moment are already created many tools for e-learning, as a development of entire informatics society.

E-learning system utilizes mostly the following communication tools, frequently:

- **Electronic (e-mail).** Electronic mail is the most widespread Internet service. This enables sending text messages through the Internet. In spite of the fact that e-mail is intended for sending text based messages mainly, today it is possible to deliver any type of file. I.e. picture or files from table calculators etc.

- **Mailing list.** Mailing lists were created on the top of the e-mail. They are also called as discussion forums. In these groups, communicate groups of persons. Groups are created according to the interests of these groups. The majority of these groups are administered automatically, i.e. administration is done by programs. Some discussion forums could be moderated. It means that the forum is owned by a concrete person, moderator, who decides about the inclusion and then distribution of the particular message.

- **News groups** (groups for new messages). They are called also as network news or servers of messages. These are special servers or web pages, which accepts messages of a definite topic. These servers sends the messages through standard protocols (they are called newsreaders) to every participant in the group. The advantage of the newsgroups is the fact that they are created very easily. Newsreaders are easily used programs. The advantage is also the fact that answers are suited immediately after the message.

- **Discussion forum.** Typical example is the discussion about an article of special server where the readers explain their views. Contributions are well-arranged into a hierarchy. The advantages are as follows:
  - Anonymity
  - Time and place independence
- Discussion in any time
- Students are active, they may help each other in doing common projects
- Contributions are composed more carefully because students have time to work on them
- Every student has chance to be heard out
- Written expression is well-organized and based on facts
- is part of the assessment

• Chat. This asynchronous communication based on the exchange of short texts, which are entered through the windows of the application. The window is divided into several ranges. The largest portion of the window contains the whole communication process. The bottom part is a row, which is the input area for the text. This text is visible for all participants. Chat is realized in few minutes and the exchanged information could be collected during several days in other forms of communication. These tools are constantly in evolution and development on various operating systems platforms. Recently new modern communication means appeared e.g. ICQ.

• Response pads. This form of communication allows immediate voting in real time. This is utilized for the enhancement of the students’ activity, e.g. their participation in questionnaires.

• Whiteboard. Whiteboard equals to the classical blackboards. In the computer milieu this represents sharing area of the monitor for drawing. When somebody draws with special software (it remains paint utility), every student notes the drawings. For the efficient utilization of the whiteboards, every student may prepare his/her own clipart (set of pictures, symbols, arrows, lines, texts, which may be utilized in the given area). Very often, the participants are identified through the colors. It is not suitable to criticize somebody for his/her bad drawings. Every relation has to be summarized, what new ideas were created, what was solved and what was not solved.

• Shared screen. The shared screen enables to watch the screens of the instructor. Some types of programs enable to follow also the progress of students on their own screens. The majority of software with shared screens enables instructors to divide the screens into several parts in order to follow the progress of the students at once.

• Audio conference. Audio conference utilizes the computer network as telephone. This is called also as videoconference. This is mainly provided by the program, which is part of the internet browser that is why the quality depends on the selected program. Audio conferences are suitable for complex and emotional discussions.

• Videoconference. This is remote interactive communication between two or more remote participants in real time. This is providing mainly by program, which is not part of the internet browser and that is why the quality of the transfer depends on the selected program. Disadvantage is the high demand on the hardware and the network transfer rate. It is important to guarantee the speed rate through the whole session.

7 Conclusions

Personal contributions are linked to system analysis at specification, implementation and validation level, by the point of view of a computation process, with agents, actions and status.

Knowledge is obtained by language users in the process of their interaction with the universe of discourse and is used by language users in their communication process. Language definition needs to include the four fundamental components of a language: the universe of discourse (called semantics), the collection of symbolic expressions used to represent semantic objects (called syntax), the process of learning to generate knowledge by representing semantic objects using symbolic notations, and the process of using knowledge by mapping symbolic notations into the semantic objects they represent. This view of a language splits the language users into two classes: speakers, who are communicators producing language expressions, and auditors, who are communicators interpreting language expressions.

References:
[7] T. Rus, A Unified Language Processing Methodology, University of Iowa, Department of Computer Science, 2001