SUPERVISOR DESIGN FOR A CLASS OF DISCRETE EVENT SYSTEMS

Daniela Cristina CERNEGA

"Dunarea de Jos" University of Galati,
Str. Domneasca nr. 111, 800021 Galati, Romania
e-mail Daniela.Cernega@ugal.ro

Abstract: *This paper proposes a control problem statement in the framework of supervisory control technique for a class of discrete event systems with cyclic working. A desired behaviour of such a discrete dynamic event system is analysed. This behaviour is cyclic and the linguistic properties for the existence of a supervisor ensuring the desired closed loop specifications are verified. Next, a systematic design procedure of such a supervisor is presented. An example is also provided and solved by using a software tool implemented in Java.*

Keywords: *discrete event systems, supervisory control*

1. INTRODUCTION

A discrete event system (DES) is a dynamic system with a discrete state space. The state transitions of a DES are determined by *events*, which occur at generally unpredictable time instants as shown in Ramadge and Wonham, (1987). If the timing information is not crucial, one can ignore it and consider only the strings of events that the system responds at. Such a modelling approach leads to the so-called *logical* DES *models* as in Wonham and Ramadge (1987), where the events order practically specifies the state trajectory. The set of all the physically possible sequences of events describes *the possible behaviour* of the discrete event system. This behaviour may be modelled with a formal language *L*; consequently, a DES may be modelled as an automaton, ***G***, generator of the language *L*.

The control problem for DES consists in executing a pre-planned process, taking into account the mutual exclusion, the concurrence of tasks and the cyclic usage of resources. In the *supervisory control theory*, proposed by Wonham and his collaborators in (Ramadge and Wonham, 1987), (Wonham and Ramadge, 1987), (Ramadge and Wonham, 1989), the control role is played by the supervisor, which is an automaton connected with the controlled DES – i.e. the "plant" in the traditional control terminology – to form a closed loop system (Fig. 1). The supervisor must achieve a prescribed language for the system equipped with the supervisor.

To control a DES consists essentially in *disabling certain events* (i.e. preventing from occurring), which is somehow different from the classical control philosophy. The set of events, denoted by Σ, is thus partitioned into *controllable* and *uncontrollable*, i.e. $\Sigma=\Sigma_c \cup \Sigma_u$.

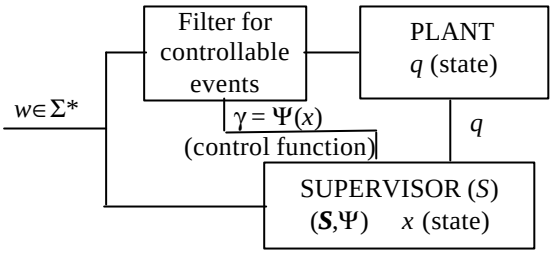

Fig. 1. Structure of a controlled DES

The events in Σ_c can be disabled at any time, they are subject to the control action, while those in Σ_u model events over which the control agent has no influence.
As shown in Fig. 1, a supervisor is a "controller" that decides in every state of the process which controllable event has to be enabled, and which has to be prevented from occurring in order to achieve the prescribed behaviour. The state transitions are generated by discrete events, and an entry signal for such a system is considered to be the string of events denoted by *w* in Fig.1.

A supervisor *S* is a pair $S=(\boldsymbol{S},\Psi)$, where $\boldsymbol{S}$ is an automaton equipped with a command function, Ψ. The command function has to establish for every state of the process which controllable event has to be disabled (because all the uncontrollable events are enabled by the control function). The supervisor does not act directly on the process, the action of disabling controllable events is done by the "filter for the controllable events" (situated on a lower control level), as shown in Fig. 1.

The supervisory control problem is fully solved when a supervisor forcing the closed loop specifications to be met *exists and it is constructible*. Generally, the existence of the supervisor is guaranteed if two conditions concerning some linguistic properties of language *L* are met, as it was proved in the paper Ramadge and Wonham, (1989).

The case study for the systematic design procedure proposed in this paper is a communication protocol conversion problem. This problem is justified by a lack of standardization in this domain. A model for the communication protocol conversion problem used for the supervisory control theory is proposed in (Kumar, et al. 1997) and in (Kumar 1991),. For example, the heterogeneity of the existing computer networks does not allow direct and consistent communication, leading to mismatch of protocols. In this paper this problem is solved through a modeling analogy with another discrete event system with cyclic behaviour: an assembly workstation, in order to use some previous theoretical results from Mînzu and Cernega (1999) and Cernega (2002).

The rest of the paper is structured as follows. In the next section, the supervisor control problem for a class of DES with cyclic working is stated and a systematic design procedure is proposed. In section 3 a DES model is deduced for the communication between two devices using two different protocols, by means of a converter and the supervisory control problem for such a system is solved. To illustrate the effectiveness of the approach, an application example is presented in section 4, which is solved by using a software program implemented in Java. Section 5 is dedicated to conclusions and ends this paper.

2. DESIGN OF THE DISCRETE EVENT SUPERVISOR

2.1 Supervisory control problem statement

In order to design a supervisor for a discrete event system the appropriate model to be used is the automaton or, equivalent the language recognized by this automaton.

Definition 1. An *automaton* $\boldsymbol{G}$ can be defined as follows:

$$\boldsymbol{G} = (Q, \Sigma, \delta, q_0, Q_m), \tag{1}$$

where:
Q is the *set of states*;
Σ is a finite set of symbols referred to as *event labels*;
δ: $Q\times\Sigma \rightarrow Q$ is (the partial) *transition function*;
q_0 is the *initial state*;
$Q_m \subseteq Q$ is the subset of *marked states*.

Let $L_m(\boldsymbol{G})$ be the set of strings leading to marked states; it represents the *marked behaviour* of $\boldsymbol{G}$.

Usually a controlled DES has the non-blocking property, i.e. $\overline{L_m(\boldsymbol{G})} = L(\boldsymbol{G})$ where $\overline{L_m(\boldsymbol{G})}$ denotes the prefix closure of the marked language, $L_m(\boldsymbol{G})$, defined as follows:
$\overline{L_m(\boldsymbol{G})} = \{u \mid \exists\, v \in \Sigma^*, \text{ such that } uv \in L_m(\boldsymbol{G})\}$.

A large class of DES from different domains may be modeled by automata having a unique marked state. Consequently, this class may be modeled by a single type of automaton, defined in Mînzu and Cernega (2002) and called *AWM* (*A*ssembly *W*orkstation *M*odel).

Definition 2. An automaton $\boldsymbol{M}$ defined by:

$$\boldsymbol{M} = (Q, \Sigma, \delta, q_0, Q_m), \tag{2}$$

is called AWM iff:
– each state of set Q is accessible and co-accessible (i.e. each state is accessible through a string which can be continued to the marked state);
– it has a unique marked state ($Q_m=\{q_m\}$), identical with the initial state ($q_m \equiv q_0$);
– each minimal cyclic sequence contains all the events from Σ;
– any cycle contains the initial state.

There exists a supervisor to ensure the closed loop admissible behaviour described by the formal language $K \subset L_m(\boldsymbol{G})$ if and only if (as stated in the general existence theorem in Ramadge and Wonham, (1987)):
– K is L_m-closed (i.e. $\overline{K} \cap L_m(\boldsymbol{G}) = K$, where $\overline{K}$ is the prefix closure of K);
– K is controllable (i.e. $\Sigma_u \cap L(\boldsymbol{G}) \subseteq \overline{K}$).

In the paper Mînzu and Cernega, (1999) a criterion for the L_m-closure of an admissible behaviour described by the formal language K in the case of an AWM automaton was proposed. The statement of the criterion requires some definitions.

Definition 3. A *cyclic sequence* is a string of events σ, having the property

$$\delta_r(\sigma, q_m) = q_m.$$

Definition 4. A cyclic sequence is called *minimal* if any event appears at most once.

Lemma 1: A sequence σ which satisfies the equality $\delta_r(\sigma, q_m) = q_m$, such that any intermediary state is different from q_m, is a minimal cyclic sequence.

The proof is based upon the fact that any transition is firable at most once between two passages through the marker state.

Definition 5. A *language* $K \subseteq L$ is *cyclic* when each element is a juxtaposition of cyclic sequences.

Lemma 2. For a given cyclic marked language, L_m, any language $K \subseteq L_m$ is cyclic.

Proof: For any $\sigma \in K$, it holds that $\sigma \in L_m$. Hence, the automaton passes through the marker state m times ($m \geq 1$). Applying lemma 1, it holds:

$$\sigma = s_1 s_2 \ldots s_m, \tag{3}$$

where s_i, $i=1,2,\ldots n$, are minimal cyclic sequences. Hence, K is cyclic.

Definition 6. A cyclic language $K \subseteq L$ is *cyclic prefix closed* if for any $\sigma \in K$, any prefix of σ which is a cyclic sequence belongs to K.

A necessary and sufficient condition for the L_m-closure of K is given hereafter.

Theorem 2 $\forall\ K \subseteq L_m$, K is L_m-closed iff K is cyclic prefix closed.
The proof of this theorem is in the paper Cernega and Mînzu (2002).

If the closed loop desired language, K, is L_m-closed, it only remains to verify its controllability. If the admissible language K is not controllable, then it will be computed the largest controllable language included in K, which is called the *supremal controllable language* ($\sup\mathbf{C}(K)$) of K.

2.2. Algorithm for Computing the Supremal Controllable Language of a Given Admissible Language

Before listing the steps of the algorithm, two definitions are needed.

Definition 7. Automaton $\boldsymbol{B}=(\Sigma, X_B, \xi_B, x_0, X_m)$ is called the *restriction of automaton* $\boldsymbol{A}=(\Sigma, X_A, \xi_A, x_0, X_m)$ if the following two conditions are met:
a) $X_{\boldsymbol{B}} \subset X_{\boldsymbol{A}}$,

b) $\xi_{\boldsymbol{B}}(\sigma,x)=\xi_{\boldsymbol{A}}(\sigma,x)$, $\forall\, x\in X_{\boldsymbol{B}}$ and $\forall\, \sigma\in\Sigma$ for which $\xi_{\boldsymbol{A}}(\sigma,x)$ is defined.

Definition 8. Let $\boldsymbol{A}=(\Sigma, X_{\boldsymbol{A}}, \xi, x_0, X_m)$ and $\boldsymbol{B}=(\Sigma, X_{\boldsymbol{B}}, \xi, x_0, X_m)$ be two automata, such that automaton $\boldsymbol{B}$ is a restriction of automaton $\boldsymbol{A}$. A state $x\in X_{\boldsymbol{B}}$ from automaton $\boldsymbol{B}$ is called *uncontrollable* state of automaton $\boldsymbol{B}$ in relation with automaton $\boldsymbol{A}$ if the following condition is met:

$$\exists\, u\in\Sigma_u \text{ for which } \xi_A(u,x)\in X_{\boldsymbol{A}}-X_{\boldsymbol{B}}.$$

Remark: If automaton $\boldsymbol{A}$ represents the physically possible behaviour of a DES and $\boldsymbol{B}$ represents the admissible behaviour, an uncontrollable state is a state from which the admissible behaviour can be exceeded when an uncontrollable event occurs. Hence, an uncontrollable state corresponds to the case in which the constraints' violation cannot be prevented from occurring using control action.

Given an AWM type automaton, $\boldsymbol{G}$, the possible behaviour, $L(\boldsymbol{G})$, the marked behaviour, $L_m(\boldsymbol{G})$, and the admissible language, K, the algorithm for computing the supremal controllable language of K is presented next.

Algorithm SCAWM (Supremal Controllable for AWM type automata)

Step 0. Let $\boldsymbol{S}_0$ be the automaton which is the recognizer of the language $L(\boldsymbol{G})$ (identical with $\boldsymbol{G}$):

$\boldsymbol{S}_0 = (\Sigma, X_0, \xi_0, q_m, \{q_m\})$,

where $X_0\equiv Q,$, $\xi_0\equiv\delta$.

Step 1. It is constructed the recognizer $\boldsymbol{S}_1$ for the language $K\subset L_m(\boldsymbol{G})$, defined by:

$\boldsymbol{S}_1 = (\Sigma, X_1, \xi_1, q_m, \{q_m\})$.

Step 2. i=1.

Step 3. It is computed the set of uncontrollable states of $\boldsymbol{S}_i$ in relation with $\boldsymbol{S}_{i-1}$, denoted by $\overline{C_i}$.

If $\overline{C_i}\neq\emptyset$, then go to *Step* 4,
else $\boldsymbol{S}=\boldsymbol{S}_i$ and STOP.

Step 4. It is constructed automaton $\boldsymbol{S}_{i+1}$ by removing from $\boldsymbol{S}_i$ the uncontrollable states and the transitions to them. Automaton $\boldsymbol{S}_{i+1}$ is defined by:

$\boldsymbol{S}_{i+1} = (\Sigma, X_{i+1}, \xi_{i+1}, q_m, \{q_m\})$,

where $X_{i+1}= X_i-\overline{C_i}$.

Automaton $\boldsymbol{S}_{i+1}$ is a restriction of $\boldsymbol{S}_i$.

Step 5. If $X_{i+1}\neq\emptyset$,
then $i=i+1$ and go to *Step* 3,
else STOP.

Remarks:
1. If language K is controllable, then the algorithm stops at *Step 1*.
2. Every automaton $\boldsymbol{S}_{i+1}$ is a restriction of automaton $\boldsymbol{S}_i$ computed at the previous iteration; therefore, $\boldsymbol{S}_{i+1}$ is a restriction of $\boldsymbol{S}_0$.

3. If the algorithm stops at *Step 5*, then there is no controllable language included in K.

Theorem 3 (in Cernega and Mînzu (2002)). For the AWM type automaton $\boldsymbol{G}$ and a given language $K\subset L_m(\boldsymbol{G})$, automaton $\boldsymbol{S}$ resulted from algorithm **SCAWM** is the recogniser of the supremal controllable language of K, sup$\mathbf{C}(K)$.

Proof (in Cernega and Mînzu (2002)) The automaton S is proved to be the recogniser for the supremal controllable language included in K_0 in two stages:
- first, it is shown that language K recognised by automaton S is a controllable language;
- then, it is shown that K is the supremal controllable language of K_0.

2.3. Systematic procedure of supervisor design

Given a discrete event system and a set of specifications, the systematic procedure to obtain the supervisor ensuring the desired behaviour consists in the following steps.

Step 0. The construction of the automaton model $\boldsymbol{G}$ for the discrete event system.

Step 1. *If* automaton $\boldsymbol{G}$ is of AWM type *then Step* 2, *otherwise* STOP.

Step 2. The specifications for the process $\boldsymbol{G}$ lead to the admissible language, K.

For the supervisor existence, the admissible language, K, has to be L_m-closed and controllable.
Step 3. The L_m-closure criterion for AWM automaton. *If* the admissible language, K, is L_m-closed, *then Step* 4, *otherwise Step* 2.

Step 4. **SCAWM** algorithm for K. *If* automaton $\boldsymbol{S}$, which recognizes the supremal controllable language contained in K, result of algorithm **SCAWM**, is nonempty, *then* Step 5, *otherwise* STOP.

Step 5. The supervisor S. The command function, Ψ, is added to automaton $\boldsymbol{S}$, in order to guarantee that the closed loop system does not exceed the admissible language. The supervisor is the pair $S = (\boldsymbol{S}, \Psi)$.

3. CASE STUDY. THE PROTOCOL CONVERSION PROBLEM

3.1. General Description

Generally, a protocol P consists of sending end protocol P_0 and the receiving end protocol P_1. Similarly, the protocol Q is composed of Q_0 and Q_1.

A protocol mismatch occurs when the sending end protocol P_0 of P tries to communicate with the receiving end protocol Q_1 of Q, and similarly when Q_0 tries to communicate with P_1. In Fig. 2 it is assumed that P_0 is the composition of the sender protocol and the sender's channel P_c, whereas the receiving end Q_1 consists of only the receiver protocol Q_r. For solving the protocol mismatch, a protocol converter, C, is interposed, for example, between P_0 and Q_1.

The events occurring at various interfaces for the example presented in Fig. 2 are the external events *accept* (*acc*) and *deliver* (*dlv*). The internal events occurring at the sender/receiver end are identified by lower/upper case letters.

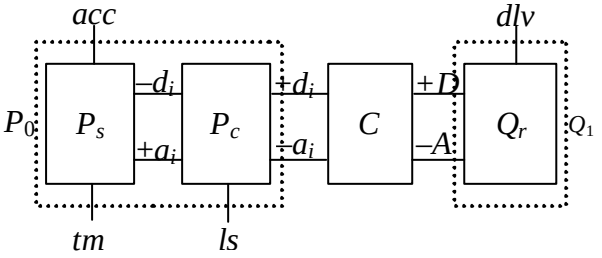

Fig. 2. A typical protocol conversion system

An event label has a negative/positive sign as its prefix denotes an event of sending/receiving. Since the converter is interposed between the channel and the receiver protocol, this convention identifies the events that occur at the converter interface. Thus, for instance, $-d_i$ represents the event of sending a data packet with label i (where i=0,1) at the sender protocol (and the event of receiving the same data at the channel), whereas $-A$ represents the event of sending an acknowledgement at the receiver protocol (and the event of receiving the same acknowledgement at the converter). Other events are the *timeout* (*tm*) and the *channel loss* (*ls*).

3.2. Systematic Supervisor Design Procedure for the Protocol Conversion Problem

The DES modeling of the above described communication system concerns its representation as an automaton.

Step 0
In Fig. 3. is represented the automaton $\boldsymbol{G}$ which models the mismatched protocols and the transmission channel; it is the "plant" that the supervisor must be coupled with.

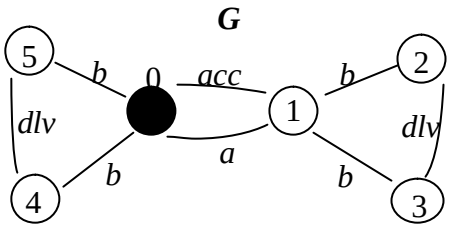

Fig. 3. Automaton G as model of the protocol conversion

In relation to Fig. 2, note that event $-a_i$ has been re-noted by a and event $+D$ has been re-noted by b.

Step1
For the automaton $\boldsymbol{G}$ in Fig. 3: Q={0,1,2,3,4,5}, Σ={a,b,*acc*,*dlv*}, q_0={0}. One can note the cyclic working of $\boldsymbol{G}$. It may be considered that $\boldsymbol{G}$ has a *single marked state*, which is identical to the initial one: Q_m={$q_0 \equiv 0$}. The elements of the marked language, $L_m(\boldsymbol{G})$, are the strings of events which determine transitions from the marked state to the marked state. Such a string is a cyclic sequence. A cyclic sequence containing every transition at most once is a minimal cyclic sequence.

Obviously, automaton $\boldsymbol{G}$ is of AWM type because all states are accessible and co-accessible. It has a unique marked state, 0, which is also the initial state. The minimal

cyclic sequences are: s_1 = *acc b dlv b a*, s_2 = *acc a b dlv b*, and s_3 = *b dlv b acc a*, and they contain all the events from Σ. Finally, all the state trajectories corresponding to these cyclic sequences contain the initial state.

Step 2

The supervisory controlled system has to transmit data through the channel, and this is possible if the events *acc* and *dlv* alternate. Also, it must satisfy a certain *progress* property, which requires that the external events be not blocked in the service specification.

The desired closed loop behaviour for this process (derived from a refined version of the service specification by synchronization with ***G*** as shown in Kumar, et al. (1997)) consists in the fact that the event *acc* is never blocked. This means that a new transmission must not be initiated until the channel becomes available. In terms of formal languages, this means that the sequence *acc a acc* must be prevented from occurring. The controllable events for this process are the converter output events (Fig. 2), $\Sigma_c=\{-a_i,+D\}\equiv\{a,b\}$, while all the other events are uncontrollable.

In Fig. 4 is represented the desired language *K*, which models the desired closed loop behaviour.

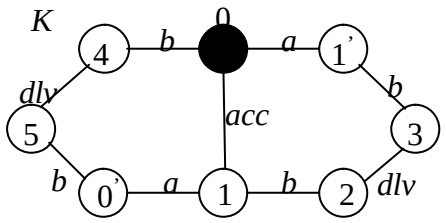

Fig. 4. Language *K*, modeling the desired closed loop behaviour

Step 3

For the problem of the protocol converter, the admissible language recognized by the automaton represented in Fig. 4 is L_m-closed because it is cyclic prefix closed (as stated by the L_m-closure criterion): all the strings leading to the marked state (0) are cyclic sequences $s_1s_2\ldots$, $s_1s_1\ldots$, $s_2s_2\ldots$ or $s_2s_1\ldots$, where s_1, s_2 are the minimal cyclic sequences s_1=*acc b dlv b a*, and s_2=*acc a b dlv b*, containing all the events at least once.

If the closed loop desired language, *K*, is L_m-closed, it only remains to verify its controllability. If the admissible language *K* is not controllable, then it will be computed the largest controllable language included in *K*, which is called the *supremal controllable language* (sup**C**(*K*)) of *K*.

Step4

For the communication system modelled as a DES by automaton ***G*** from Fig. 3, having as desired (admissible) closed loop behaviour the one described by language *K* from Fig. 4, algorithm **SCAWM** provides automaton ***S*** shown in Fig. 5.

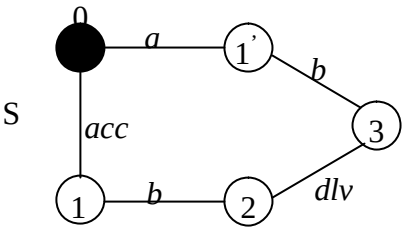

Fig. 5. Automaton *S* resulted from algorithm SCAWM for the protocol conversion problem

One can note that state 0' in *K* is uncontrollable, because from this state event *acc* may occur, which is uncontrollable and it is not possible to be disabled by control action.

Step 5

For the supervisor design, the command function, Ψ, is added to automaton ***S***, result of the **SCAWM** algorithm. The command function is important in state 0, where it is possible for the process to make a transition in state 0', and the controllable event *a* has to be disabled: $\Psi(a, 1)=0$.

The command function for all the other states enables all the controllable events and this is the reason why it does not need to be specified.

4. IMPLEMENTATION

The described algorithms are implemented in a Java based software named SYCDES, which runs in command mode and has some graphic facilities. The main types of objects used are *DES* and *supervisor*. The latter inherits the structure of the DES type and has as specific feature the command function.

Starting SYCDES has as effect the opening of the command window, from which a menu is also available. The functioning of SYCDES is next illustrated on the protocol conversion problem discussed above.

To initialise a new variable of DES type, for example `ProcessG`, one must type the command `newdes(ProcessG)` in the command window. Next, the edit window must be selected from the menu, which allows introducing the number of states, the table of transitions (in the form *initial state – event – final state*), the marked states, the admissible states and the initial states. Such a window is presented in Fig. 6, for automaton ***G*** from Fig. 3 (texts are in Romanian, the authors' native language).
The initialisation of a DES type variable can be also performed in command mode. The edit window serves only to visualize the content of an existent DES variable.

The specifications can be introduced in the same manner, that is, as the corresponding recognizing automaton. For example, the command `newdes(AdmK)` will allocate memory space for a DES type variable, intended to contain the description of the automaton which recognizes the admissible (desired) language. The SYCDES dedicated command for a restriction (admissible language) of a given process language is `AdmK=Res(ProcessG)`. For language *K*, whose recognizing automaton is depicted in Fig. 4, the edit window looks like in Fig. 7.

The command `Kc=supC(AdmK)` further provides the supremal controllable language of language *K* in the DES variable `Kc` (see Fig. 8). One can easy check that this corresponds to automaton *S* from Fig. 5.
SYCDES allows that the DES variables being introduced as the *graphs* representing the associated automata, saved in a graphic format. The graphical form is next converted into a DES variable.

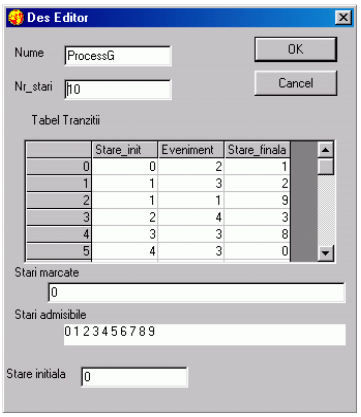

Fig. 6. Edit window for a DES type variable

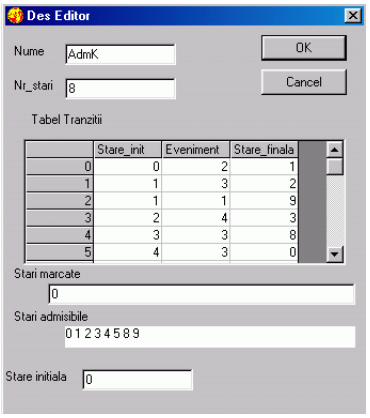

Fig. 7. The content of the DES variable `AdmK`, describing the desired closed loop behaviour from Fig. 4

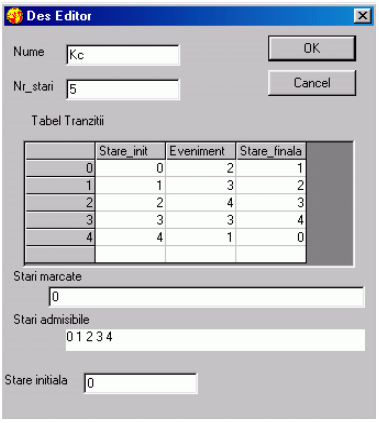

Fig. 8. The DES variable `Kc`, corresponding to the supremal controllable language of *K* (variable `AdmK`)

5. CONCLUSION

This paper presented a systematic procedure of supervising a communication protocol conversion modelled as a discrete event system, more precisely, as a DES fulfilling some properties. This kind of DES model is sufficiently general to allow the supervisor synthesis in order to meet some required closed loop specifications. The proposed procedure was implemented in a Java based software. Future work will aim at enriching this software with a simulation module to validate the supervisor effectiveness.

Also, another research direction is the extension of the proposed technique to complex systems, using the modular supervision theory or decentralized supervisory control when disposing of partial observations.

REFERENCES

[1] Cernega D. C. and Mînzu V. (2002), The Computation of the Supremal Controllable Language for an Assembly Workstation, in *Proceedings of the 9th IFAC Symposium on Large Scale Systems: Theory and Applications – LSS 2001*, Elsevier Science ISBN 0-08-0436919, ISSN 1474-6670, pp. 343-348.

[2] Kumar R. (1991), Supervisory Synthesis Techniques for Discrete Event Dynamical Systems, *Ph.D. Thesis, Texas University, Austin, U.S.A.*.

[3] Kumar R., Nelvagal S. and Marcus S. I. (1997) , A Discrete Event Approach for Protocol Conversion, *Grant no. T.R. 97–3, Institute for System Research*, University of Maryland, U.S.A.

[4] Mînzu V. and Cernega D. C (1999), Supervisory Control Technique for Assembly Workstation, in *Proceedings of IEEE International Symposium on Assembly and Task Planning – ISATP '99*, July 21–24 1999, Porto, Portugal, pp. 88–93.

[5] Mînzu V., Cernega D.C. and Henrioud J.–M. (2001), Linguistic Model and a Control Problem for Assembly Workstation, in *Proceedings of the 2001 IEEE International Symposium on Assembly and Task Planning – ISATP 2001*, May 28–29 2001, Soft Research Park, Fukuoka, JAPAN, pp. 381–386.

[6] Ramadge P. J. and Wonham W. M., (1987), Supervisory Control of A Class of Discrete Event Processes, *SIAM Journal of Control & Optimization*, vol. 25, no. 1, 1987, pp. 206–230.

[7] Ramadge P. J. and Wonham W. M. (1989), Modular Feedback Logic for Discrete Event Systems, *SIAM Journal of Control & Optimization*, vol. 25, no. 5, 1989, pp. 1202–1218.

[8] Wonham W. M. and Ramadge P. J. (1987), On the Supremal Controllable Language of a Given Language", *SIAM Journal of Control & Optimization*, vol. 25, no. 3, 1987, pp. 637–659.