J-DES: A Graphical Interactive Package for Analysis and Synthesis of Discrete Event Systems [1]

Xi Wang xxw117@psu.edu

Asok Ray axr2@psu.edu

Shashi Phoha sxp26@psu.edu

Junning Liu jxl390@psu.edu

The Pennsylvania State University
University Park, PA 16802

Keywords: *Finite-state automata; Regular Languages; Supervisor Synthesis; Software Tools*

Abstract

This paper describes the algorithmic and software design principles of an object-oriented, Java-based graphical interactive software package, J-DES, for analysis and synthesis of Discrete Event Systems (DES). The software is also capable of quantitatively evaluating different supervisory controllers in terms of a language measure.

1 Introduction

Finite state automata serve as a mathematical model for a variety of applications, such as digital systems, command and control (C^2) systems, communication systems, and manufacturing systems. The concept is applicable to any system that can be characterized by discrete symbols as inputs and outputs at a certain level of abstraction. These systems belong to the class of discrete event systems (DES). The supervisory control theory (SCT), initiated by Ramadge and Wonham [7], provides a systematic approach for automated synthesis of DES supervisors [9].

Software packages for analysis and synthesis of DES controllers using SCT have been developed in both timed (TTCT) and untimed (TCT) settings at the University of Toronto, Canada. UMDES-LIB developed at the University of Michigan is a supervisory controller synthesis package that can perform both standard operations of supervisory control and failure diagnosis in DESs. UK-DES [2] is yet another graphical DES toolkit that can perform analysis and synthesis of supervisors. Ostroff [6] has reported graphical design of real-time discrete event systems, using a timed transition model (TTM) with real-time temporal logic (RTTL) specifications. DESCO is a set of tools for manipulation of automata and Petri nets and exploration of SCT [3]. However, there are very few packages available that can interactively generate graphic displays of resulting automata. Some of the existing software packages are text mode only, for example, TCT/TTCT. Although other packages have a graphical interface that allows the users to interact with the software at the initial stage of the plant or specification construction, the resulting automata, after a series of operations, can only be generated into a text file. To interpret the resulting automata, the user has to manually convert the text file into a diagram, which can be tedious, erroneous, and time-consuming. In addition, it is difficult to implement new theoretical results such as quantitative evaluation of DES supervisors [8] and synthesis of optimal DES supervisor [4] in an existing software because of the changes in data structures required for representing an automata. For example, to quantitative evaluate the performance of DES supervisors [8], we proposed to assign weights to individual transitions which are state-dependent, unlike the controllability/observability of events in the alphabet, which is state-independent. Therefore, we implemented a more generic form of weighted finite state automata in J-DES's data structure in the sense that the user can assign weights not only to transitions, but also to states. Salient features of the new software J-DES are: (i) an interactive visual environment for analysis and synthesis of DES supervisors; (ii) capability to implement full functionality of existing SCT with its new data structure; and (iii) capability to incorporate new results of research in the software.

2 Supervisory Control Synthesis for DES

This section briefly introduces the basic concepts of supervisory control that forms the backbone of the J-DES software. The plant and control specifications are both modelled as Deterministic Finite State Automata (DFSA) as:

$$\text{Plant Model:} \quad G = (Q, \Sigma, \delta, q_0, Q_m)$$
$$\text{Specification Model:} \quad S = (X, \Sigma, \xi, x_0, X_m)$$

[1]This work has been supported in part by Army Research Office (ARO) under Grant No. DAAD19-01-1-0646 and NASA Glenn Research Center under Grant No. NAG-3-2448.

The synchronous composition [5] of G and S, representing the controlled plant behavior, is defined as:

$$G/S \equiv G \| S = (Q \times X, \Sigma, \delta \times \xi, (q_0, x_0), Q_m \times X_m)$$

and is implementable by synchronizing G and S via common events. It should be noted that synchronous composition defined in [5] also allows operation on non-identical alphabets that are used for the DES control problem of coordination or synchronization among several component systems. Controllability is a necessary and sufficient condition for the existence of a supervisor that achieves a desired prefix-closed controlled behavior for a given DES under the complete observation of events in alphabet. That is, given a nonempty prefix-closed sublanguage $K \subseteq L(G)$, there exists a supervisor S such that $L(S/G) = K$ if and only if K is controllable with respect to G and Σ_u, the set of uncontrollable events, *i.e.*, $pr(K)\Sigma_u \cap L(G) \subseteq pr(K)$. If this controllability condition is not satisfied, then the *supremal controllable sublanguage* of K, denoted as $K^\uparrow$, can be effectively computed under maximally permissive supervision. An algorithm can be found in [5] for the computation of $K^\uparrow$. For details of SCT, reader may refer to [5] [9].

3 Performance Measure of DES Supervisors

A measure of regular languages has been proposed by Wang and Ray [8] for quantitative performance evaluation of supervisors for a given plant. The J-DES formulates a signed measure of all sublanguages of the plant language $L(G)$ in two steps.

State transition cost is assigned to the plant model G as a function $\pi : Q \times Q \rightarrow [0, 1)$ such that $\forall q_i, q_j \in Q, \forall \sigma \in \Sigma$:

$$\pi_{ij} \equiv \pi[q_j | q_i] = \sum_{\delta(q_i,\sigma)=q_j} \tilde{\pi}[\sigma | q_i],$$
$$\{\delta(q_i, \sigma) = q_j\} = \emptyset \Rightarrow \pi_{ij} = 0.$$

where $\tilde{\pi}[\sigma|q_i] \in [0, 1)$ is the event cost to the transition σ defined on state q_i. We denote the resulting $n \times n$ state transition cost matrix as Π-matrix and π_{ij} is the j-th element of i-th row in Π-matrix.

Elements of the $n \times 1$ characteristic vector χ is defined corresponding to the n plant states as: good marked states (positive weight); bad marked states (negative weight); and (remaining unmarked states (zero weight).

$$\forall q \in Q, \quad \chi(L(q)) \in \begin{cases} [-1, 0), & q \in Q_m^- \\ \{0\}, & q \notin Q_m \\ (0, 1], & q \in Q_m^+ \end{cases} \qquad (1)$$

where Q_m^+, Q_m^- denote the good and bad marked state, respectively, and $L(q) = \{s \in \Sigma^* \mid \delta(q_0, s) = q \in Q\}$. The state weighting vector $\mathbf{X} = [\chi_0 \ \chi_1 \cdots \chi_{n-1}]^T$ is therefore formed where the i-th element χ_i of $\mathbf{X}$-vector

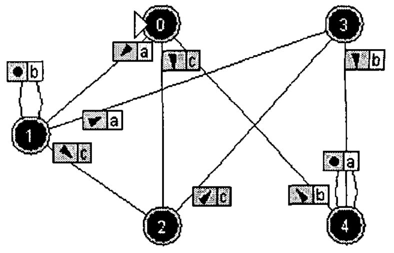

Figure 1: A Finite State Automaton (FSA) in J-DES

is the weight assigned to the corresponding state q_i. To compute the language measure of G, let $\mu_i \equiv \mu(L_i)$ denote the language measure of $G_i = (Q, \Sigma, \delta, q_i, Q_m)$, where L_i denote the marked language of G_i, then

$$\mu_i = \sum_j \pi_{ij} \mu_j + \chi_i \qquad (2)$$

In vector form, the solution to equation 2 is given by $\boldsymbol{\mu} = (\mathbf{I} - \mathbf{\Pi})^{-1}\mathbf{X}$. For details, reader may refer to [8].

4 Software Design

We have chosen `Java` as the language of J-DES software package for the following reasons. One advantage of `Java` over `C++` is that it permits a runtime environment for platform independence. The user has freedom to use the same package on Windows, Linux, Macintosh, etc. `Java` is also completely object oriented – even more so than `C++` – in the sense that everything in `Java`, except for a few basic types like numbers, is an object. The `Swing` package in `JDK1.2` or later provides a set of "lightweight" portable components that are extremely convenient for the design of graphical user interface. The basic data structure of J-DES is compatible with other relevant software packages to support analysis and synthesis of DES supervisory control algorithms while its extension further allows to incorporate new algorithms.

4.1 Finite State Automata Objects

From object-oriented software design point of view, there are several advantages for algorithms operating on finite state automata solely through abstract accessors and mutators (a. k. a. `get` and `set` methods). First, it masks the internal representation of an object. Second, it allows generic algorithms that operate on possible multiple objects. Third, it provides a mechanism for the implementation of on-demand operations. To illustrate this, consider the following accessor and mutator methods in classes `JP_FA` and `JP_FAState`, respectively:

- *fsa*.`getState(int stateID)`, which returns a `JP_FAState` object with the given `stateID`;

- *state*.`setFinal(boolean b)`, which specifies the current state to be `b`, *i.e.*, (un)marked.

There are a variety of ways of implementing these algorithms. For example, the states of an automaton can be stored as a vector, an array, or even a linked list. Hiding the internal implementation from the high level SCT algorithms enhances flexibility of the inner structure. The code remains operative as long as the algorithm's interfaces are kept unchanged. J-DES also takes advantage of `Java`'s ability to serialize objects. (Serialization is the process of storing the current state of an object in byte-code for future use.) In order for an object to be serialized, its defining class must implement a `Serializable` interface. Although there are no specific methodology for serialization, every referenced object is serialized. Consequently, each referenced object must have a defining class which implements `Serializable`. Java's serialization feature may not allow objects from older files to be serialized in the event of changes in the internal data structures of a DFSA. However, it should be noted that JDES's generalized form of weighted DFSA has a generic structure that a DFSA could possibly have. Although this feature may not be necessary in many cases (e.g., design of standard DES supervisors), the overhead introduced by the extension of data structures is small. In addition, J-DES has a function to export its DFSA file to ASCII text file for further conversion to the forms recognizable by other packages. In J-DES design, a DFSA object is serialized through three classes: `JP_FA`, `JP_FAState`, and `JP_FATransition`. Below we list the major attributes, methods, constructors of `JP_FA`.

```
class JP_FA extends Object implements
Cloneable, Serializable {
  public Vector States = new Vector();
  public Vector vAlphabet = new Vector();
  private String Alphabet;
  private String TransitionList;
  private int numberOfStates;
  private int numberOfTransitions;
  private int numberOfFinalStates;

  public JP_FA() { ... }
  public void addState(int x, int y,
         float chi_value) { ... }
  public JP_FAState getState(int stateID){...}
  public JP_FAState getStateWith(int x,
         int y){...}
  public Vector getParentList(int stateID){...}
  public String getFinalList(){...}
  ...
}
```

It is a common practice to make a local copy of an object, such as `JP_FA`, for high level operations(e.g., synchronous composition) and to modify that object while keeping the original object intact. In `Java`, it is convenient to do so by cloning objects. All of the above three classes for a DFSA object serialization implement the `Cloneable` interface to complete the cloneability of corresponding objects.

4.2 Description of the J-DES Software

The main GUI for J-DES is shown in Figure 1 with an example. The main menus consist of the **File** menu for manipulations on DFSA files; the **Edit** menu for the convenient modification of DFSA under construction; the **Operations** menu which is partitioned into three parts for standard operations, equivalent check, and various DES supervisory control operations, respectively; the **Options** menu that allows to switch between automatic transition layout and manual transition layout; the **Tools** menu that has several language measure-oriented functions; and the **Help** that gives on-line help for using J-DES. There is also a toolbar below the main menus which contains a set of icons. Clicking an icon is equivalent to select a corresponding menu item.

As one can see from the Figure 1, the graphical display of an DFSA provides the user with an intuitive understanding of the automaton structure such as the initial state, (un)marked states, (un)controllable and/or (un)observable events. States are represented as double circles filled with different colors. The state 0 (by default) with a triangular arrowhead filled with yellow pointing to it is the initial state. The initial state can be changed to be an arbitrary state in the DFSA by right clicking on top of a state and selecting the "make marked" item from the pop-up menu window. The marked states are filled with red color while the unmarked states are filled with blue color. By default, a state is unmarked. A state may also be moved over the canvas once it is created so that the overall layout of the automata is easily readable and convenient to recognize. A transition is displayed in different colors to indicate if it is (un)controllable and/or (un)observable. By default, a transition is green, indicating both controllable and observable. Light gray indicates a controllable and unobservable transition, light brown indicates an uncontrollable and observable transition, and pink indicates an uncontrollable and unobservable transition. *Changing the controllability/observability of one transition will change the controllability/observability of all transitions indicated by the same event accordingly* since in SCT controllability/observability is associated with the event, not transition. Transitions are grouped together if they are defined from the same source state to the same destination state. User has the flexibility to manipulate each state and each transition of a state either individually or together. A standard transition table of the current DFSA can be obtained by clicking "**Tools | Automata Properties**". Instead of graphically displaying the weights to states and transitions, the user can specify the numerical values of the $\tilde{\pi}$ and χ in a table by clicking "**Tools | Au-**

tomata Properties" and selecting the "Pi-Chi Table" tab from the tabbed pane. (Note that weights assigned to the same event defined on different states are not necessarily the same.) J-DES has the capability for editing a table that directly maps an event or a state of the DFSA to its physical identity (`*.phd` files) by clicking "**Tools | Automata Properties**" and selecting the "Physical Meaning" tab from the tabbed pane. In addition, the user can have a natural language description (`*.spn` files) of the physical process and the control specification by first clicking "**Tools | Automata Properties**" and then selecting the "**Control Specification**" tab. This helps a user to easily understand others' design or refer to a previous design. Lack of this feature is a common deficiency in many existing DES software.

4.3 Available Operations in J-DES

Currently, the following operations are available in J-DES package.
Rational operations: Union, concatenation, Kleene closure, complement, and projection;
Compositions: Synchronized composition (i.e., intersection of the plant and supervisor alphabets), parallel composition(intersection of the plant and supervisor alphabets defined in [1];
Equivalence transformations: ϵ-removal, determinization, minimization, inaccessible state removal;
SCT-oriented operations: Accessibility, completion, trimness, prefix-closure, controllability, observability, supermal controllable sublanguage;
Special operations: Language measure, optimal control synthesis, creation of regular expressions, etc.
In addition, there is a set of utilities available for transforming a DFSA to some other formats for further investigation. For example, the Π-matrix and **X**-vector together can be transformed into a `*.mat` file and then loaded it in MATLAB for language measure computation and verify the results generated by J-DES. One can also export an FSA structure into a plain text file for other purposes such as DES simulation and real-time control code generation.

4.4 SCT Algorithms Efficiency

For supervisor synthesis, we follow the procedures outlined on page 74 and 75 in [5]: accessibility of plant DFSA $G \rightarrow$ trimness of specification DFSA $S \rightarrow$ prefix closure of $S \rightarrow$ relative closure of $S \rightarrow$ controllability of S. The details of these algorithms are available in the JDES user manual for complete description. In this paper, we present two algorithms and their complexity analysis for example. In the accessibility algorithm, we examine each state only once. For each state, we check all the transitions going out of that state also once. Thus the computational complexity of the algorithm is $\mathcal{O}(nk)$ where n is the number of states of the DFSA and k is the number of total transitions. The controllability check of S with respect to plant G can be determined in $\mathcal{O}(mn)$ time, where m, n denote the number of states in G and S respectively, if we ignore other parameters.

```
Algorithm: Accessibility
1. Let the set of reachable states $R = \{ q_0 \}$,
       where $q_0$ is the starting state
2. for all $p \in R$
       begin
3.         Find the set of states $R' = \{ q \in Q \mid$
               $\exists \sigma \in \Sigma$ s.t. $\delta(p,\sigma) = q$ and $q \neq p \}$
4.         for all $q \in R'$ and $q \notin R$, insert state $q$ in $R$
       end
5. if $R = Q$, then $G$ is accessible
6. else $G$ is not accessible
```

Figure 2: Algorithm for accessibility check

```
Algorithm: Controllability
1. Specify the set of uncontrollable events $\Sigma_u$
2. Compute $\overline{S}$ by calling Completion()
3. Compute $G||\overline{S}$ by calling SynComp()
4. for all state $z = (q,x) \in Z$ of $G||\overline{S}$, set flag = true
       begin
5.         if $\exists \sigma \in \Sigma_u$, s.t., $\overline{\psi}(z,\sigma) = z_D = (\cdot, x_D)$,
           then
               flag = false
               add $\overline{\psi}(z,\sigma) = z_D$ to a list $\mathcal{L}$, where $\mathcal{L}$
                 denotes those uncontrollable transitions
                 that directly lead to the dump state $z_D$
                 in $G||\overline{S}$
       end
6. if(flag) then $K = L_m(S)$ is controllable
7. else $K$ is uncontrollable, print the list $\mathcal{L}$
```

Figure 3: Algorithm for controllability check

5 An Example for Illustration of J-DES Usage

The well-known example of dining philosophers [1] is chosen to demonstrate some of J-DES's features. This should help users to understand the detailed operations of J-DES.

> Each of the two philosophers may either think or eat. There are two plates of food, one in front of each philosopher, and two forks, one on each side between the plates. To switch from the "thinking" state to the "eating" state, each philosopher needs to pick up both forks from the table, one at a time, in either order. After a philosopher has finished eating, he/she places both forks back on the table and returns to the "thinking" state.

Figure 4 and 5 shows the automaton models of the two philosophers, denoted by P_1 and P_2, constructed in J-DES. The events of P_1 are defined in Table 1 which is also constructed in J-DES. Events of P_2 have the similar meanings to those of P_1 hence having been omitted for display. To further capture the constraint that a fork can only be used by one philosopher at a time, we create two more automata, one for each fork, as shown in Figure 6 and 7. Therefore, the complete uncontrolled system behavior, denoted by G, is the parallel composition of all four automata, P_1, P_2, F_1, and F_2. This automaton is shown in Figure 8. Due to the common events between the philosopher automata and the fork automata, only 9 out of 64 possible states are accessible in G.

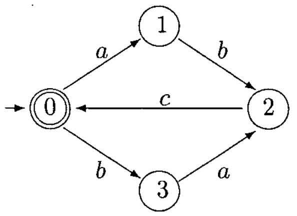

Figure 4: P_1

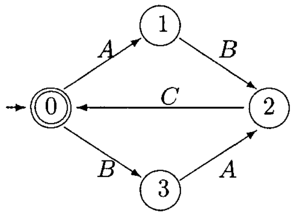

Figure 5: P_2

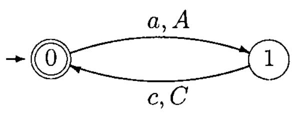

Figure 6: F_1

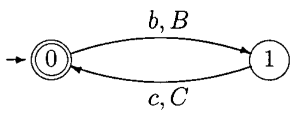

Figure 7: F_2

Table 1: Even Definition for P_1

Event	Physical Description	Controllability
a	philosopher 1 pick up fork 1	true
b	philosopher 1 pick up fork 2	true
c	philosopher 1 put both forks	false

```
2  0  0.24  0.24  0     0.24  0.24  0     0     0     0     0     3
   0  0     0     0.48  0     0     0     0     0.48  0     0
   0  0     0     0.48  0     0     0     0.48  0     0     0
   0  0     0     0     0     0     0     0     0     0.96  0
   0  0     0     0     0     0     0.48  0.48  0     0     0
   0  0     0     0     0     0     0.48  0     0.48  0     0
   0  0     0     0     0     0     0     0     0     0     0.96
   0  0     0     0     0     0     0     0     0     0     0
   0  0     0     0     0     0     0     0     0     0     0
4  0  0.24  0.24  0     0.24  0.24  0     0     0     0     0     5
   0  0.24  0.24  0     0.24  0.24  0     0     0     0     0
```

There are two deadlock states in G, state 7 and state 13. The automaton is blocking when each philosopher is holding one fork and waiting for the other fork to be available. To avoid this deadlock, we would have to design a supervisor to prevent a philosopher from picking up an available fork if the other philosopher is holding the other fork. Suppose the control specification K is given as follows: No restriction on whether Philosopher 1 or Philosopher 2 eats first; and each philosopher is allowed to eat consecutive courses of meals. The control specification K describes the constraints that we wish to impose on the plant's behavior is also modeled by a finite state automata S_1 shown in Figure 9. As is well-known, a supervisor exercises control over the plant by

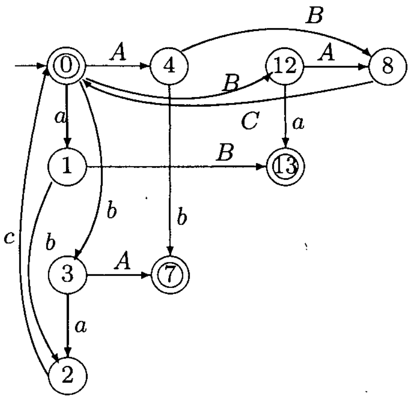

Figure 8: Plant Model $G = P_1 || P_2 || F_1 || F_2$

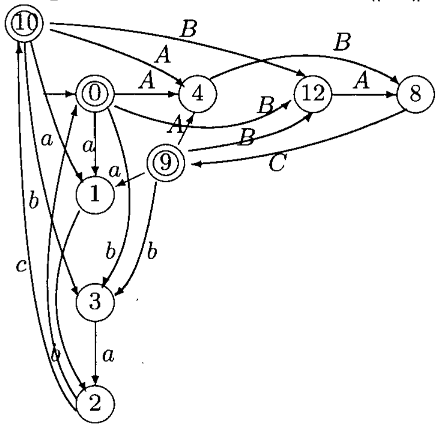

Figure 9: Specification Automata S_1

dynamically disenabling a minimal set of controllable events (maximal permissiveness) to achieve the desired specification. In practice, a controller extracted from the supervisor only select, if possible, at most one controllable event from these enabled by the supervisor. By doing so, the controller guides the system (possibly infinitely often) to reach the "good" states while trying to avoid "bad" states(e.g., deadlock state).

In J-DES, after constructing the plant G and specification S_1 automata, we check "Trimness" of both G and S_1 by selecting these operations under the menu "**Operations**". To synthesize a supervisor, we first perform "Completion" operation on S_1 by selecting "**Operations | Completion**". The resultant FSA $\overline{S_1}$ has one more state than S_1, called the "dump" state q_D. All the undefined transitions at each state in S_1 are added to indicate a transition from that state to the dump state q_D, therefore the generating language of FSA $\overline{S_1}$ equals Σ^*. Now we perform "Synchronized Composition" operation between G and $\overline{S_1}$ by selecting "**Operations | Synch Composition**", as shown in Figure 10. Since the resultant DFSA $G||\overline{S_1}$ created in J-DES preserves the property of event controllability which is specified when G or S_1 is created, it is ready to check the controllability of the specification with respect to Σ_u and $L(G)$. To do so, we select

"**Operations | Controllability**". If it turns out the specification is uncontrollable, we need to compute the supremal controllable sublanguage which is also under the menu "**Operations**"; otherwise the specification automata S_1 is the supervisor. In our example, S_1 is controllable, i.e., $K = L(S_1)$ where S_1 is the supervisory controller that restrict the plant to the desired behavior K. Alternatively, one can do so by visual checking if there is any brown(indicating uncontrollable) transition leading to the dump state q_D.

For the purpose of performance evaluation of supervisor S_1, we need both the $\mathbf{\Pi}$-matrix of the uncontrolled plant G, which is assumed to be known or needs to be derived from experimental observations, and the characteristic (i.e., signed weighting of plant states) vector $\mathbf{X}$ of G, which is selected by the designer. These values are shown in the matrix. In G, there are three marked states as seen in Figure 8. The one good marked state, 0, is assigned positive χ values 1, and the two bad states, 7 and 13 (that are deadlock) are assigned negative χ values -1. As the goals of the supervisor S_1 for plant control are: (i) to increase the possibilities of the philosophers' return to the good marked state, i.e., going back to a thinking state after eating; and (ii) to decrease the possibilities of reaching one of the bad marked states, i.e., being indefinitely stuck in a deadlock state of fork holding without eating. Based on

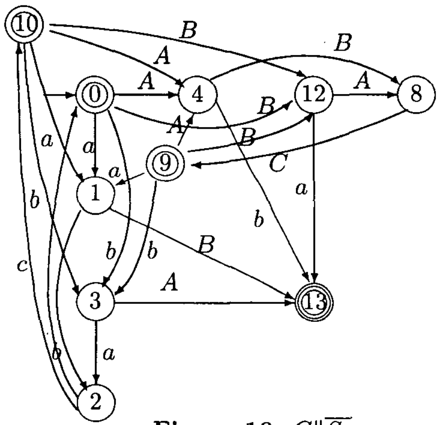

Figure 10: $G\|\overline{S_1}$

these data, the language measures of the uncontrolled plant G and controlled plant S_1/G in Figures 8 and 9, respectively, are computed as:

$$\mu(L_m(G)) = 1.3801; \quad \mu(L_m(S_1/G)) = 1.7933$$

by selecting "compute" button in the "Pi-Chi Table" tab. The rationale for superior performance of S_1/G relative to that of the uncontrolled plant G is that S_1 disables the controllable events a, A, b and B leading to the bad marked states 7 and 13. If more than one supervisor is designed, then we can quantitatively measure the performance of the closed-loop system under different supervisors.

6 Summary and Future Work

This paper reports the design of a Java-based software package, called J-DES, for analysis and synthesis of supervisory control algorithms for discrete-event systems that are modelled by finite state automata.

J-DES is built upon the foundations of supervisory control theory and is not restricted to any specific class of applications. The software architecture of J-DES has flexible data structures and is based on highly modular algorithms for execution of a variety of logical operations on regular languages. The software is formulated for efficient (e.g., with reduced memory overhead and execution time) analysis and synthesis of DES supervisory control systems; it also provides quantitative evaluation of the control system performance by taking advantage of language measures. The major advantages of J-DES over other DES design software tools are mainly due to its interactive graphical features and accommodation of integrated supervisory control algorithms, specifically, the DES language measure and optimal control algorithms. Current research efforts focus on inclusion of new fault diagnosis algorithms within the J-DES software.

References

[1] C. G. Cassandras and S. Lafortune, *Introducrion to discrete event systems*, Kluwer Academic, 1999.

[2] V. Chandra, B. Oruganti and R. Kumar, *Ukdes: A graphical software tool for the design, analysis & control of discrete event systems*, IEEE Transactions on Control Systems Technology (Submitted).

[3] M. Fabian and A. Hellgren, *Desco - a tool for education and control of discrete event systems*, Proceedings of WODES (2000), 471–472.

[4] J. Fu, A. Ray and C. Lagoa, *Unconstrained optimal control of regular languages*, 2002 IEEE Conference on Decision and Control (Submitted).

[5] R. Kumar and V. Garg, *Modeling and control logical discrete event systems*, Kluwer Academic, 1995.

[6] J. S. Ostroff, *A visual toolset for the design of real-time discrete-event systems*, IEEE Transactions on Control Systems Technology **5** (1997), no. 3, 320–337.

[7] P. J. Ramadge and W. M. Wonham, *Supervisory control of a class of discrete event processes*, SIAM J. Control and Optimization **25** (1987), no. 1, 206–230.

[8] X. Wang and A. Ray, *Signed real measure of regular languages*, American Control Conference (2002).

[9] W. M. Wonham, *Discrete event system control theory and application*, University of Toronto, 2001.