Journal of Advanced Computer Science and Technology, 2 (1) (2013) 9-27
©Science Publishing Corporation
www.sciencepubco.com/index.php/JACST

Identifying the effect of model modifications in State-Based models and systems

Luay Tahat[1], **Nada Almasri**[2]*

Management Information Systems Department, Gulf University for Science and Technology
[1] *E-mail: tahat.l@gust.edu.kw*
[2] **Corresponding author E-mail: almasri.n@gust.edu.kw*

Abstract

System modeling is a widely used technique to model state-based systems. System models are often used during the development of a software system, e.g., in partial code generation and in test generation. Several modeling languages have been developed to model state-based software systems, e.g., EFSM, SDL, and State Charts. Although state-based modeling is very useful, system models are usually large and complex, and they are frequently modified because of specification changes. Identifying the effect of these changes on the model and consequently on the underlying system is usually challenging and time-consuming. In this paper, we present an approach to automatically identify the effect of modifications made to the model. The goal is to identify those parts of the model that may exhibit different behaviors because of the modification. These are usually critical parts of the system that should be carefully tested. In this approach, the difference between the original model and the modified model is identified, and then the affected parts of the model are computed based on model dependence analysis. An empirical study on different EFSM models is performed in order to identify the affected parts of the model after a modification. The results of the study suggest that our approach could considerably reduce the amount of time and efforts spent to validate the model after a modification.

Keywords: *EFSM, Model Dependence Analysis, State-Based Systems, System Modeling.*

1 Introduction

The demand for large and complex software systems has been steadily increasing over time. The development and maintenance of these systems are difficult and costly due to the increased complexity of the systems. In recent years, several new technologies have emerged, which have made a significant impact on new ways of software development. One of these technologies is a development of modeling techniques to model state-based software systems. System models are often used during the development of a software system, mainly in partial code generation and in test generation for model-based testing. Typically, state-based systems can be modeled using formal description languages. Several modeling languages have been developed to model state-based software systems such as: State Charts [19, 11, 12, 13], Extended Finite State Machines (EFSM) [20, 14, 15], Specification Description Language (SDL) [21, 16, 15], Virtual Finite State Machine (VFSM) [18], and ESTELLE [19]. In recent years, several model-based test generation [20, 22, 23, 24-26] and test suite reduction [9, 13] techniques have been developed based on these modeling languages. System modeling reduces ambiguity, misunderstanding, and misinterpretation of system specifications. However, modern systems tend to be very large and complex [27, 47] and, as a result, they are hard to understand, difficult to modify and debug [28, 29, 27, 47].

During software maintenance of large and evolving software systems, their specification and implementation are modified to fix defects, to enhance or change functionality, to add new functionality, or to delete the existing functionality. Modifications in specifications frequently lead to modifications in system models. After a modification is made to the model, the developer may be interested in parts of the model that affect the modification (modified parts of the model) and parts of the model affected by the modification. Typically, modified models are only used to understand the modifications made in the model. However, developers may have difficulties understanding the effect of modifications on the system model and consequently on the system especially for large system models. Thus, there is a need for techniques that can support understanding of model modifications and identifying their effect on the system.

In this paper, we present an approach of identifying the effect of model-based modifications. The goal is to identify those parts of the model that are related to the modification and may exhibit different behavior because of the modification. We concentrate on EFSM models; however, the presented approach may be applied to other types of state based modeling methods such as: SDL and State Charts. The approach uses the original model and the modified model, and it automatically identifies a difference between these models, where the difference between two models is a set of elementary model modifications [7, 43]. The approach using model-based dependence analysis identifies parts of the model that affect the modification and parts of the model that are affected by the modification. Our initial experience with the approach shows that it significantly helps in understanding and identifying the effect of modifications on the system.

The rest of the paper is organized as follows: Section 2 provides an overview of state based modeling. Section 3 introduces model-based dependence analysis. Section 4 presents our approach in identifying the effect of model modifications, and it introduces an algorithm to compute the parts affected by the modification as well as the parts affecting the modifications. In Section 5, an empirical study is performed, and the results of the study are presented. Section 6 outlines the related work on the application use of system models. In Section 7, conclusions and future research directions are discussed.

2 State-Based modeling

The process of structuring and formulating software specifications is normally guided by modeling techniques. System models for state-based systems describe the system behavior by a set of states and system actions represented as transitions between states. The languages used for modeling state- based systems are often graphical, which make modeling techniques easy to comprehend and utilize. Modeling techniques have received wide industry acceptance, especially in the fields of telecommunication, embedded systems, and computer networking [14,], where state-based systems are prevalent.

The most popular formal description techniques (languages) used for modeling of state-based systems are Finite State Machine (FSM) [17], Extended Finite State Machine (EFSM) [7, 15, 14, 34, 36, 44], Specification Description Language (SDL) [16, 15], State charts [11, 12, 13], Virtual Finite State Machines (VFSM) [18], and ESTELLE [19]. Formal modeling techniques provide the basis to validate the system design [39, 40], to generate system level test suites [38, 41, 40, 37, 14, 31, 32], to simulate the system behavior by executing the model [42, 18] and to determine properties of the system. Although modeling techniques can help eliminate complexity of many of today's systems, the resulting models are difficult to understand, analyze, and modify [28, 29], especially for large and complex systems [27, 47] where there are a large number of states and transitions. During software maintenance developers are frequently interested in a partial system model related to an element of interest, e.g., a requirement, a transition, or a feature [28, 30, 31, 32, 33, 34], that they need to analyze, understand, or modify, e.g., slices of models [34, 35]. However, the existing techniques do not address the problems of understanding modifications of system models. Thus, there is a need to develop techniques that can help in understanding model modifications.

In this paper, we concentrate on the EFSM system models, however, our approach can be extended to other modeling languages such as SDL, State charts. EFSM [7, 15, 14, 34, 36, 43] is very popular technique for modeling state-based systems like communications and control systems.

An EFSM consists of a set of states (including a start state and an exit state) and transitions between states. A transition is triggered at its originating state when an event occurs (e.g., an input is received) and an enabling condition (e.g., a Boolean expression) associated with the transition is satisfied. When the transition is triggered, a sequence of actions may be performed (which may manipulate variables and produce an output) and the system is transferred to the terminating state of the transition. The following elements are associated with a transition: an event, a condition, and a sequence of actions. Figure 1 shows a graphical representation of an EFSM transition. We distinguish three types of actions: an input action (read), an output action (write), and an assignment action. In our model assignment, actions have syntax of assignment statements and enabling conditions have syntax of conditional statements of C language.

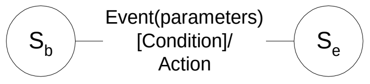


Fig. 1: EFSM Transition

An EFSM M is expressed formally as a 7 tuple: $M = (\Sigma, Q, Start, Exit, V, O, R)$ where:
Σ is the set of events,
Q is the set of states,
$Start \in Q$ is the start state,
$Exit \in Q$ is the exit state,

V is a finite set of variables,
O is the set of actions,
R is the set of transitions, where each transition *T* is represented by the tuple: $T = (E, C, A, S_b, S_e)$ where:
$E \in \Sigma$ is an event,
  C is an enabling condition defined over *V*,
  A is a sequence of actions, $A = <a_1, a_2, \ldots\ldots, a_j>$, where $a_i \in O$,
  $S_b \in Q$ is the transition's originating state,
  $S_e \in Q$ is the transition's terminating state.

In addition, the following notation related to a transition T is introduced:
$S_b(T)$ is the originating state of transition *T*,
$S_e(T)$ is the terminating state of transition *T*,
$C(T)$ is the enabling condition (a Boolean expression) associated with transition *T*,
$E(T)$ is the event associated with transition *T*,
$A(T)$ is a sequence of actions associated with transition *T*.

In *M*, Σ is a set of events, each of which is an external stimulus (input) that may be associated with a list of arguments; i.e., an event $E \in \Sigma$ is represented by $E(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_k)$. States in *Q* are passive elements in the EFSM model. States are just snapshots of the system and they are not involved in any kind of decision-making or computation. The states *Start* and *Exit* are where the system starts and terminates, respectively. The variables in *V* provide storage for values that is accessible by enabling conditions and actions in transitions. An action $a_i \in O$ is one of the following types: assignment action, *output* action, or function call. An *assignment* action assigns a value to a variable. An *output* action displays a variable or a constant to the external environment. A *function call* to some function $f(v_1, v_2, \ldots, v_k)$ returns the evaluated value.

A transition *T* in *R* is triggered when the system is in the originating state $S_b(T)$, the event $E(T)$ occurs, and the enabling condition $C(T)$ is evaluated to TRUE. When transition *T* is triggered, the $A(T)$ sequence of actions is performed and the system is transferred to the terminating state $S_e(T)$. If a transition *T* is specified at a state with no enabling condition, no other transition from that state can be associated with $E(T)$.

EFSM models may be depicted as graphs where states are represented by nodes and transitions by directed edges between states. A simple EFSM model of an ATM system is shown in Figure 2 [9, 17, 27]. This ATM system supports two types of accounts: checking account and savings account and three types of transactions: withdrawal, deposit and balance. Before ATM transactions can be performed, user must enter a valid pin that is matched against the pin stored in an ATM card. A user is allowed a maximum of three attempts to enter the valid pin. For example, transition T_2 is triggered when the model is in state S_1, event *PIN*(*p*) is received, the value of parameter *p* does not equal to variable *pin*, and the value of variable *attempts* is less than three. When the transition is triggered, an error message is displayed, the value of variable *attempts* is incremented, and the user is prompted to enter PIN. Notice that in this example, for transition T_2, $S_b(T_2) = S_1$, $S_e(T_2) = S_1$, $C(T_2) = (\text{p != pin})$ and $(\text{attempts} < 3)$, $E(T_2) = \text{PIN(p)}$.

In this paper, we assume that the EFSM model is executable [18, 42, 43], i.e., enough detail is provided in the model so that the model executor can execute the model based on the model specification (or an executable program corresponding to the model can be generated from the model specification). In order to support model execution, some actions may not be implemented (they are represented by "empty" actions). However, all actions are implemented during the development of the system. An input to the EFSM is a sequence of events with values for arguments associated with the events. For example, consider the following input for the EFSM of the ATM system of Figure 2:

t = Card(1234,100,200), PIN(1234), Savings(), Deposit(20), Receipt(), Withdrawal(50), Receipt(), Done(), Exit().

When the model of Figure 2 is executed on the sequence of events *t above*, the following sequence of transitions is executed: $\tau(t) = <T_1, T_4, T_9, T_{11}, T_{12}, T_{10}, T_{12}, T_8, T_6>$.

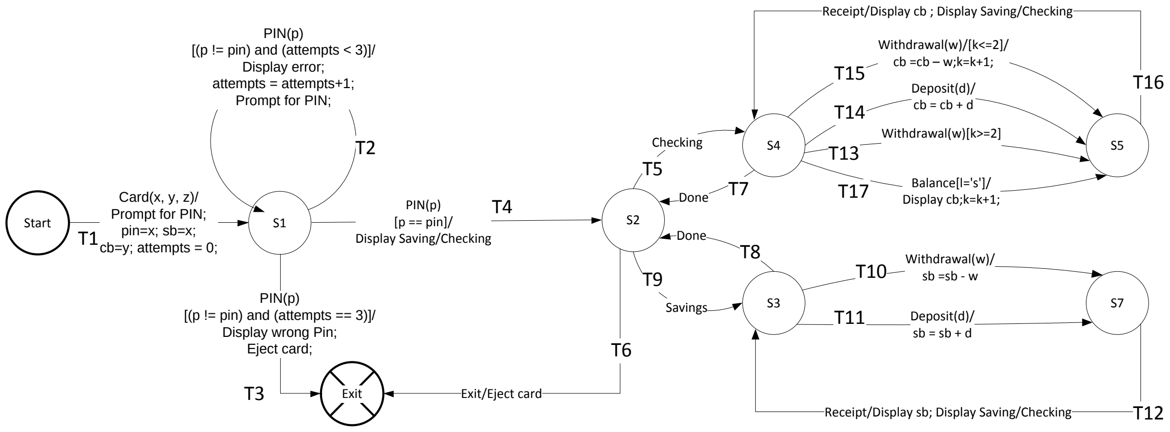


Fig. 2: Sample ATM Model

In this paper, we assume that the EFSM model is deterministic, i.e., for every event $E_i(x_i)$ where x_i = arg$_1$, arg$_2$, …, arg$_k$, in t there is one and only one possible execution of model M (at most one transition is executed for a given event $E_i(x_i)$). When model M is executed for a given sequence of events t = <$E_1(x_1)$, $E_2(x_2)$, …, $E_n(x_n)$>, a sequence of transitions τ (t) = <T_{i1}, T_{i2}, …, T_{im}> is executed.

Modifications in specifications frequently lead to modifications in system models. Traditionally, a modified model is only used to understand the modifications made in the model. However, developers may have difficulties understanding the effect of modifications on the model and consequently on the system. For example, consider a model of Figure 2, the model is modified by adding a balance transition to the savings account: transition T_{18}. The modified model is shown in Figure 3.

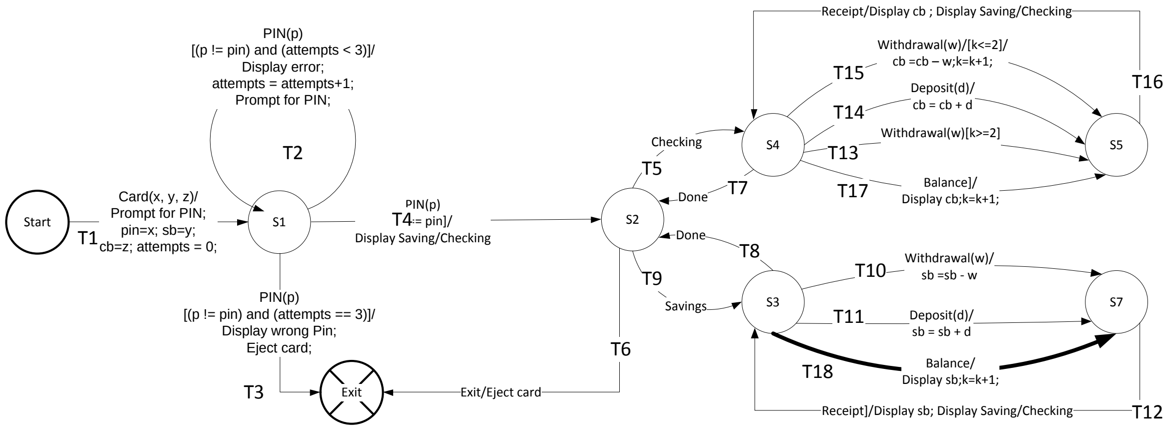


Fig. 3 Modified ATM model of Figure 2 (transition T_{18} is added)

The modification seems to be benign and should not have any effect on the system. Clearly, the developer may have difficulties understanding actual effect of this modification on the system, and, as a result, some undetected problems may be passed to the next phases of the development. Typically, the developer after making a modification to the system model may be interested in answers to the following questions:

- Which parts of the model affect the modification?
- Which parts of the model are affected by the modification?

In the first question, the developer may be interested in model transitions, referred to as *affecting transitions* that affect the modified part of the model. This may be important to understand whether the modification interacts with expected transitions of the model. For example, Figure 4 shows parts of the model (affecting transitions) that affect the modification, i.e., transition T_{18} in the modified model of Figure 3. The affecting transitions are shown in bold lines in Figure 4. From Figure 4, the developer may easily identify that transitions T_1, T_4, T_9, T_{10} and T_{11} affect the modification as expected. However, transitions T_5, T_{15} and T_{17} unexpectedly affect the modification. To a developer, it is not evident how transitions of the checking account may have an effect of the savings accounts.

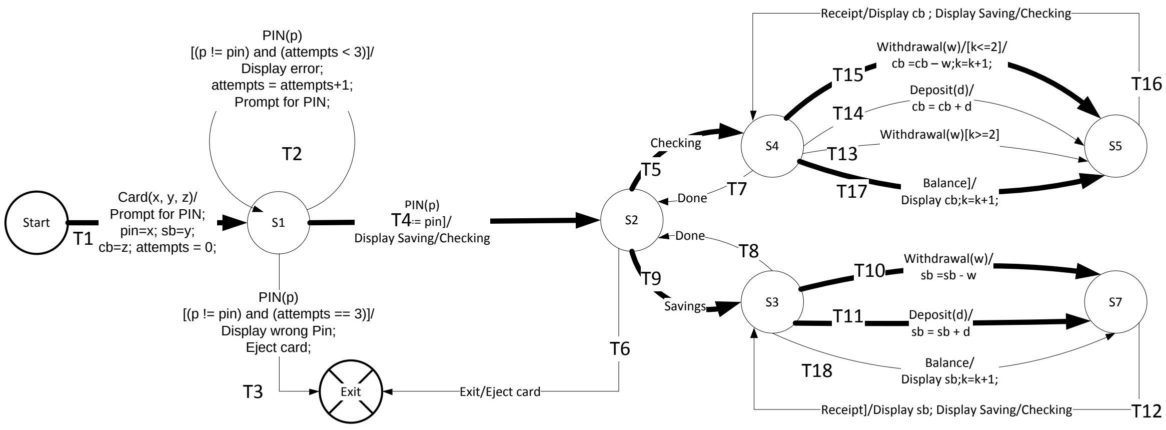


Fig. 4 Model transitions affecting the modification

In the second question, the developer may be interested in transitions, referred to as *affected transitions* that may be affected by the modification. As a result, the developer may have a better understanding whether the intended transitions of the model are affected. For example, Figure 5 shows parts of the model (affected transitions) that are affected by the modification in the modified model of Figure 3. The affected transitions are shown in bold lines. The developer most likely expects that the modification should not affect any parts of the model. However, the modification unexpectedly affects almost all transitions related to the checking account (T_{13}, T_{14}, T_{15}, T_{16} and T_{17}). In addition, the modification affects transition T_{12}.

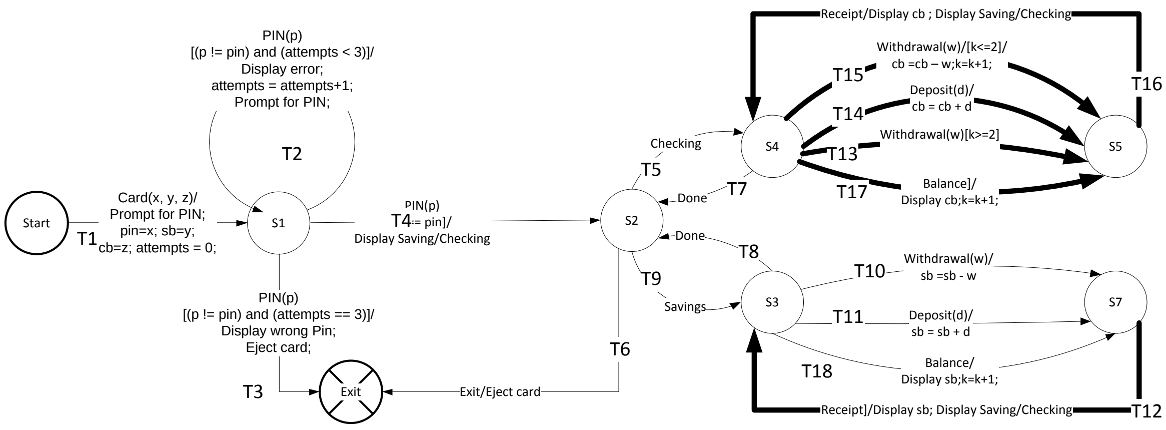


Fig. 5 Model transitions affected by the modification

3 Model dependencies

In order to identify affecting and affected transitions for a model modification, we use model dependence analysis. In this section, we introduce model dependencies that may exist in the system model, specifically in the EFSM model [7, 33]. We define two types of dependencies between transitions ("active" elements of a model): data dependence and control dependence. Note that states are "passive" elements of the model. These dependencies capture the notion of potential "interactions" between transitions in the model.

3.1 Data dependence

Model dependence analysis with respect to data dependence focuses on occurrences of variables within the system model. Each variable occurrence is classified as being a variable definition or a variable use. We refer to these as *definition* and *use*, respectively. A definition of a variable v in a transition is any occurrence of v at which v is assigned a value. A transition can define a variable v by defining v as a part of the action(s) (e.g., $v = x + 5$). A use of a variable v in a transition is any occurrence of v that references the value of v. A transition can reference a variable v in a Boolean expression associated with the transition (*e.g.*, [$\boldsymbol{v} < 0$]) or by using v in action(s) associated with the transition (*e.g.*, $x = v + 5$).

Let T be a transition. The following concept related to transition T is introduced:

- D(T) is a set of variables defined by transition T, i.e., variables defined by an action(s) or by a triggering event of T.
- U(T) is a set of variables used in transition T, i.e., variables used in a condition and an action(s) of T.

For example, in the EFSM model of Figure 2, for transition T_1, D(T_1) = {*pin*, s*b*, *cb*, *attempts*} and U(T_1) = {x, y, z}.

Data dependence captures the notion that one transition defines a value of a variable and another transition may potentially use this value. There exists a *data dependence* between transitions T_i and T_k if transition T_i modifies the value of variable v, transition T_k uses v, and there exists a path (transition sequence) in the model from T_i to T_k along which v is not modified [33]. More formally, there exists *data dependence* between transitions T_i and T_k if there exists a variable v such that: (1) $v \in$ D(T_i), (2) $v \in$ U(T_k), and (3) there exists a path (transition sequence) in the EFSM model from T_i to T_k along which v is not modified; such a path is referred to as a *definition-clear path.* For example, there exists a data dependence between transitions T_1 and T_{11} in the model of Figure 2. This is because transition T_1 assigns a value to variable s*b* in the action "s*b* = *y*", transition T_{11} uses variable s*b* in action "sb=sb+1", and there exists a path from T_1 to T_{11} along which s*b* is not modified (sequence of transitions T_1, T_4, T_9, T_{11}). Notice that there is no data dependence between T_1 and T_{12} because along the path from T_1 to T_{12}, s*b* is modified by transitions T_{10} or T_{11}.

3.2 Control dependence

Control dependence was originally defined for a program's Control Flow Graph (CFG) [45]. Control dependence captures the notion that one node in the control graph may affect the execution of another node. In this paper, we extended the concept of program control dependence to the EFSM model [9]. Control dependence in an EFSM exists between transitions and it captures the notion that one transition may affect traversal of another transition. Control dependence between transitions is defined similarly to control dependence between nodes of a CFG [45], i.e., in terms of the concept of post-dominance. Let Y and Z be two states (nodes) and T be an outgoing transition (edge) from Y. State Z *post-dominates* state Y if Z is on every path from Y to the exit state of the EFSM. State Z post-dominates transition T if Z is on every path from Y to the exit state of the EFSM through transition T. Transition T_k is control dependent on transition T_i iff: (1) $S_b(T_k)$ does not post-dominate $S_b(T_i)$ and (2) $S_b(T_k)$ post-dominates transition T_i. Notice that the definition of control dependence presented in this paper captures the same view as the definition of control dependence between nodes in a CFG [45].
For example, transition T_4 has control dependence on transition T_9 in the model of Figure 2 because state S_2 does not post-dominate state S_1 (condition 1 of control dependence definition is true) and state S_2 post-dominates transition T_4 (condition 2 is TRUE). The issue of control dependence in EFSMs is discussed in more detail elsewhere [46, 48].

3.3 Model dependence graph

Data and control dependence in the model can be graphically represented by a directed graph where nodes represent model transitions and directed edges represent model data and control dependencies.
More formally, let $M = (\Sigma, Q, Start, Exit, V, O, R)$ be an EFSM model and let $G=(R, E)$ be a model dependence graph of model M:

Where:

R is a set of nodes (set of transitions)

E is a binary relation on R, $E \subseteq R \times R$, referred to a set of directed edges where: edge $(T_i, T_k) \in E$, if there exists data or control dependence between transitions T_i and T_k.

Throughout this paper, we will be consistent on using the statement "there exists dependence between transitions T_i and T_k". It will always mean that T_k depends on T_i (not the opposite), so in the dependence graph, the relation will be represented by the directed edge (T_i, T_k).

Due to space limitation, Figure 6 shows only a partial model dependence graph of the model of Figure 2. Variables associated with data dependencies are also not shown. Note that data dependencies are shown as solid edges and control dependencies are shown as dashed edges.

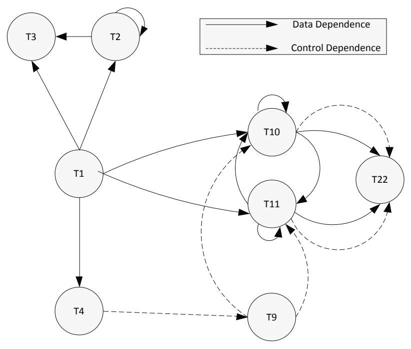


Fig. 6 Partial Model Dependence graph of the ATM model

4 Effect of modifications

Our approach to support identifying the impact of modifications in system models is based on the observation that usually not the whole system model is affected by the modified part of the model (a modification). Frequently, only a relatively small part of the model is affected by the modification. When a software system is modified two types of analysis can be performed to support understanding the model modifications: identifying the effect of the model on the modification (affecting transitions), and identifying the effect of the modification on the remaining part of the model (affected transitions). In order to identify the effect of a model modification, we compare changes in dependencies between the original model and the modified model. In this section, we formally define "Model Modification", "Affects Relationship", "Affecting Transition" and "Affected Transitions", and then we introduce an algorithm to compare the difference in dependencies between the original model and the modified model. The end results of the algorithm are the set of affecting transitions and the set of affected transitions.

Definition 4.1: *For a Model M_o, an elementary modification represents one of the following actions:*

1. *Deleting an existing transition T where $T \in R$*
2. *Adding a new Transition T to R* (1)

In this context, editing an existing transition $T \in R$ is equivalent to deleting the transition T and adding a new transition T' .

Definition 4.2: *For an original Model M_o, a model modification (MF) is represented by two sets: R_d and R_a, where R_d is the set of transitions to be deleted from M_o, and R_a is the set of transitions to be added to M_o. We call the resulting model after applying the model modification on M_o: "the modified model M_m".* (2)

Definition 4.3: *Let G=(R, E) be the dependence graph of the model M. A transition T in R "affects" another transition T' in R if and only if there is a non-null path from T to T' in G.* (3)

Since the dependence relationship itself is not transitive, then we can look at the "affects" relationship as the transitive relation of the dependence relation between transtions. For example, if transition T depends on transition Q, and transition Q depends on transion S, then S "affects" T.

Definition 4.4: *Let G=(R, E) be the dependence graph of the model M. The set of affecting transitions for a transition T in G is the set of all transitions T' that "affects" the transition T. Formally, we define this set as:*

AG(T) = R′, where R′ $\subseteq$ *R, and T′∈ R′ if and only if "T′ affects T " on R.* (4)

Definition 4.5: *Let G=(R, E) be the dependence graph of the model M. The set of affected transitions for a transition T in G is the set of all transitions T′, where T "affects" T′. Formally, we define this set as:*

AD(T) = R′, where R′ $\subseteq$ *R, and T′∈ R′ if and only if "T affects T′ " on R.* (5)

Definitions 4.4 and 4.5 present an understanding of "affected transitions" and "affecting transitions" as a function of a specific transition of interest T. In this paper however we are more interested in understanding the effect of the whole modification of the model rather than the effect of a single transition. Therefore, in the remaining of this section we formalize our understanding of affecting and affected transitions as a funciton of model modification *MF* instead of a transition *T*.

4.1 Identifying affecting transitions

Introducing a model modification to an original model M_o produces a modified model M_m. To understand the effect of the modification, we analyze the dependence graph of the original model $G_0=(R_o, E_o)$, and compare it with the dependence graph of the modified model $G_m=(R_m, E_m)$.

We have defined model modification in definition 4.2 as two sets: the set of added transitions R_a, and the set of deleted transition R_d. To understand what parts of the model have an impact on the modifed part of the model (*MF*), we focus on the transitions affecting the added transitions. As for deleted transtions, they can no longer be affected by other transitions in the model since they have been deleted, however, their deletion may affect the model. Consequently, we can derive the set of transtions affecting the model modification by looking at R_a. We investigate the impact of the deletion of transtions in R_d as part of the effect of the modication on the model.

Definition 4.6: *Let $G_m=(R_m, E_m)$ be the dependence graph of the modified model M_m. The set of affecting transitions for a model modifcation MF on M_o is the Union of AG(T) for all transitions $T \in R_a$. Fromally, we define the affecting transitions for a model modification as follows:*

$$AG(MF) = \bigcup_{T \in R_a} AG(T) \qquad (6)$$

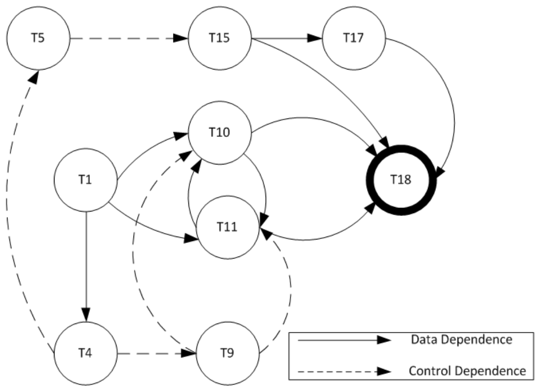


Fig. 7: Model dependence sub-graph that affects added transition T_{18}

For example, suppose a developer is interested in understanding the modification of adding transition T_{18} in the model of Figure 2, which results in the model of Figure 3. Figure 7 shows a dependence sub-graph with respect to added transition T_{18} of the model of Figure 3. All transitions in this sub-graph affect T_{18} and are highlighted in the modified model as shown in Figure 4.

4.2 Identifying affected transitions

Identifying transitions affected by the modification is more complex than identifying transitons affecting the modification. This is true mainly because we need to look at both R_a and R_d in order to understand the impact of the modification on the original model M_o. In this section, we identify five cases covering all possible changes in the dependence graph that may be caused by the addition of a new transtion or the deletion of a transition. The first three cases discuss the possible impact of adding a new transtion, while the last two cases discuss the possible impact of deleting a transition.

Case 1: When a new transition is added, it will have a direct impact on the transitions that will be involved in a dependece relation in G_m with the added transition. In other words we say that an added transition T has an impact on a transition Q, if there exists an edge(T, Q) $\in E_m$, where $Q \in R_o$. For example, Figure 8 shows a dependence sub-graph with respect to added transition T_{18} of the model of Figure 3 after forward traversal starting from T_{18}. All transitions in this sub-graph are affected by T_{18}.

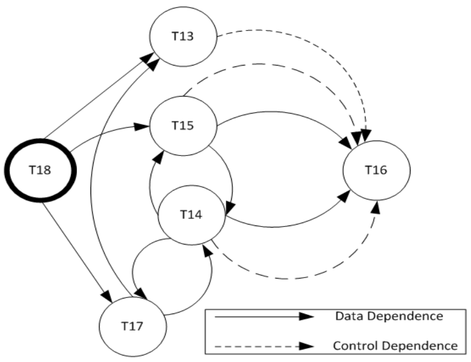


Fig. 8 Model dependence sub-graph that is affected by added transition T_{18}

Case 2: When a new transition is added, new dependencies between existing transitions may be introduced. For example, the new transition may create a definition-clear path between other existing transitions. This path will lead to new dependencies in G_m that didn't exist in G_o. In this case, the added transition will have an indirect impact on the transitions invovled in the new dependcies. Mainly, an added transtion T has an impact on a transition Q if there exists an edge(S,Q) $\in E_m$ where edge(S,Q) $\notin E_o$ and S, $Q \in R_o$. For example, in the original model of Figure 2, there is no dependence between T_1 and T_{12}. However, when transition T_{18} is added into the model of Figure 3, transition T_{12} becomes data dependent on transition T_1 in the modified model.

Case 3: When a new transition is added, it may break a dependency relation between two transitions. As a result, a previously existing dependence relation in G_o may cease to exist in G_m. Consequently, transitions which were involved in a dependence relation in G_o, but are no longer involved in this dependence relation in G_m are indirectly affected by the addition of the new transition. An added transtion T has an impact on a transition Q if there exists an edge(S, Q) $\in E_o$ where edge(S, Q) $\notin E_m$ and S, $Q \in R_m$. For example, assume that figure 9 is the original ATM model, and suppose that transition T_{17} is added to the model resulting in the model of Figure 3. Adding transitions T_{17} breaks the dependency between T_1 and T_{16} in the original model of Figure 9. We notice that this dependency between T_1 and T_{16} doesn't exist anymore in the modified model of Figure 3. Clearly, T_{16}, the dependent transition, may behave differently because of the modification.

Case 4: When a transition is deleted, it will have direct impact on the transitions that were previously invovled in a dependence relation with the deleted transition. These transitons may have dependence relation on other transtions in M_m. Thus, a deleted transition T will have an impact on a transition Q if there exists an edge(T, Q) $\in E_o$ where $Q \in R_m$ and $T \in R_d$. For example, consider the model of Figure 3. Suppose transition T_{17} is deleted from this model, resulting in the model of Figure 9. The modification is a deletion of transition T_{17}. In the original model of Figure 3, there is a data dependence between transition T_{17} and T_{13}, T_{15} and T_{18}. Clearly, these transitions are affected by deletion of T_{17}.

Case 5: When a transition is deleted, it may delete a definition-clear path between two transitons. In this case the dependence relation between existing transtions may cease to exist after applying the modification. This case has the same effect as case 3 although the cause is different. A deleted transtion T has an indirect impact on a transtion Q if there exists an edge(S,Q) $\in E_o$ where edge(S,Q) $\notin E_m$ and S,$Q \in R_m$. For example, consider the original model of Figure 2, suppose that transition T_{17} is deleted from this model resulting in the model of Figure 9. The modification consists of the deletion of transition T_{17}. We notice that the dependence relation between T_1 and T_{16} in the original model M_o cease to exist in modified model M_m. Clearly, T_{16} may behave differently because of this modification.

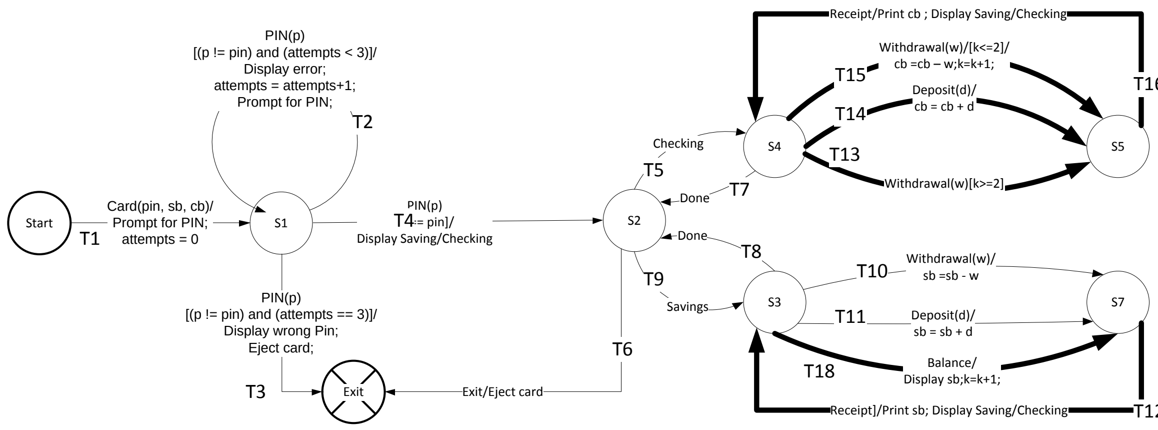


Fig. 9 Modified Version of the ATM model

The above five cases identify all possible sets of transitions on which the model modifcation will have a direct or an indirect impact. The model modification doesn't only affect the transitions identified in these five cases, but it also affects all transitions affected by theses identifed transitions as well. The formal definition of the set of affected transitons for a model modification is:

Definition 4.7. *Let C_1, C_2, C_3, C_4, and C_5 represnt the sets of transitions identifed in the above five cases. Let G=(R, E) be the dependence graph of the model M. The set of affected transitions for a model modifcation AD(MF) on M is the Union of AD(Q) for all transitions Q $\in\{C_1 \cup C_2 \cup C_3 \cup C_4 \cup C_5\}$.*

$$AD(MF) = \bigcup AD(Q), \text{ for all } Q \in \{C_1 \cup C_2 \cup C_3 \cup C_4 \cup C_5\}. \quad (7)$$

4.3 Algorithm to compute affected and affecting transitions

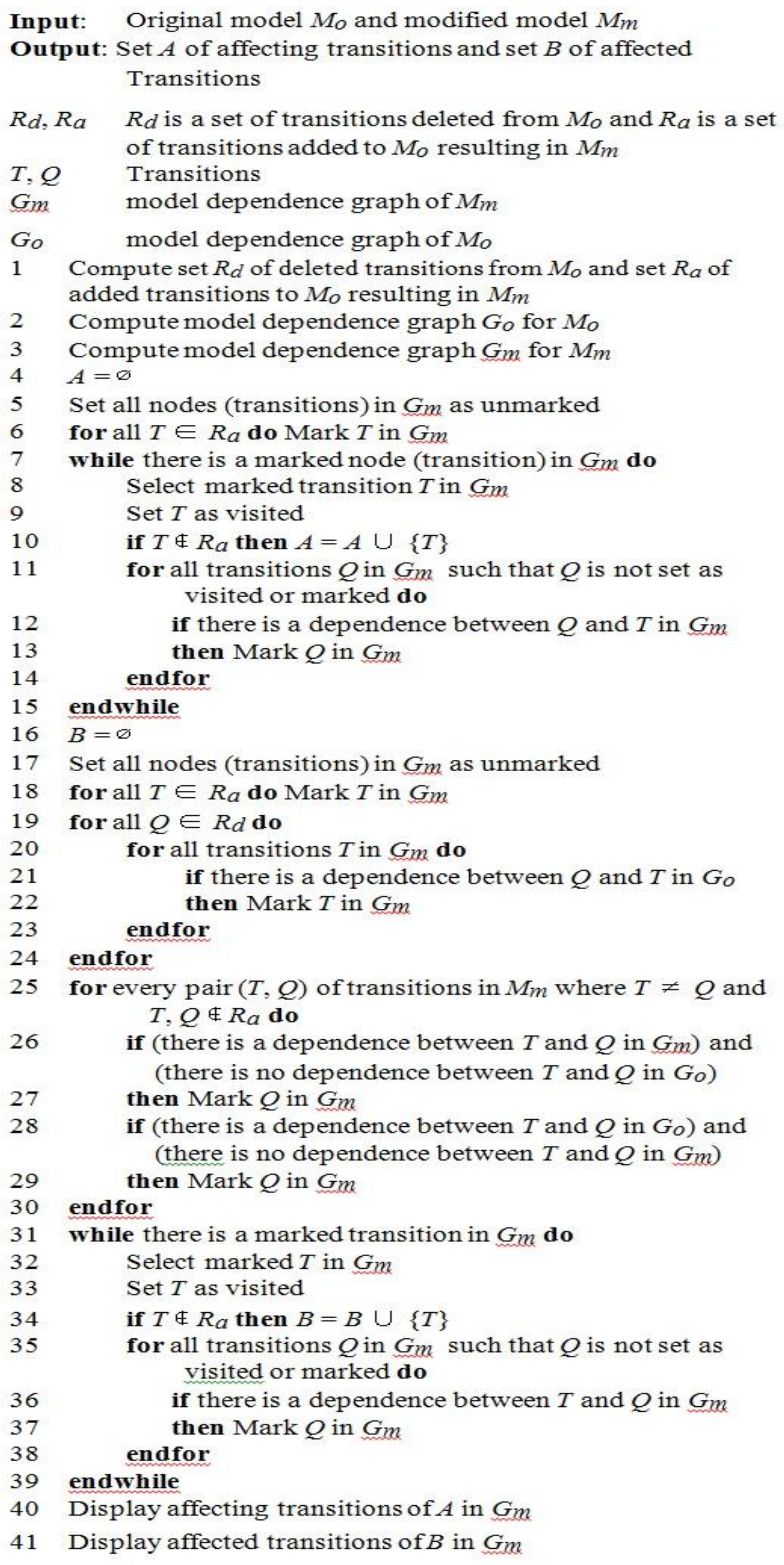
In this section, we present an algorithm to compute affecting transitions and affected transitions for any complex model modification. The algorithm is shown in Figure 10. The algorithm uses the original model M_o and the modified model M_m and automatically identifies the difference between these models, where the difference between two models represents the model modification MF. This difference between the two models M_o and M_m is represented by the set R_a of added transitions and the set R_d of deleted transitions. A transition addition may occur between existing states or may involve an introduction of a new state when a transition is added to the model. Similarly, a transition deletion may, in some cases, result in the deletion of a state. Notice, however, that an addition of a new state and a deletion of a state are always associated with a transition addition and a transition deletion. Therefore, addition of a new state or a deletion of a state is not considered as an elementary modification. For example, the difference between the original model of Figure 2 and the modified model of Figure 3 is $R_a=\{T_{18}\}$ and $R_d=\{\}$. On the other hand, the difference between the original model of Figure 2 and the modified model of Figure 9 is the following elementary modifications: deletion of transition T_{17} , and addition of transition T_{18}, i.e., $R_a=\{T_{18}\}$ and $R_d=\{T_{17}\}$. After the difference between models is identified, the algorithm uses the dependence graph of both the original model and the modified model to compute the affecting transitions for all added transitions in R_a. The algorithm then computes the set of affected transitions based on both R_a and R_d.

In line 1, the algorithm computes the difference between the original model M_o and the modified model M_m, i.e., sets R_a and R_d. The algorithm used to compute the difference between the two models is straight forward [17, 28, 43], hence modifications in terms of the added and deleted transitions can be devised as follows. Let R_o be a set of transitions of M_o and R_m be a set of transitions of M_m. The algorithm then amounts to taking two set differences between R_o and R_m provided that the state and transition names are preserved across versions of the models. That is, only previously unused state and transition names appear in the modified model M_m for the added states and transitions, then, the algorithm becomes the following:

$$R_a = R_m - R_o$$

$$R_d = R_o - R_m$$

The complexity of this algorithm is at most, 4*(| R_o |+ | R_m |)-1 comparisons provided that the sets R_o and R_m are sorted in the same order over the transition names [50].



```
Input:   Original model $M_o$ and modified model $M_m$
Output: Set $A$ of affecting transitions and set $B$ of affected
         Transitions

$R_d$, $R_a$   $R_d$ is a set of transitions deleted from $M_o$ and $R_a$ is a set
         of transitions added to $M_o$ resulting in $M_m$
$T$, $Q$     Transitions
$G_m$       model dependence graph of $M_m$
$G_o$       model dependence graph of $M_o$
1   Compute set $R_d$ of deleted transitions from $M_o$ and set $R_a$ of
    added transitions to $M_o$ resulting in $M_m$
2   Compute model dependence graph $G_o$ for $M_o$
3   Compute model dependence graph $G_m$ for $M_m$
4   $A = \varnothing$
5   Set all nodes (transitions) in $G_m$ as unmarked
6   for all $T \in R_a$ do Mark $T$ in $G_m$
7   while there is a marked node (transition) in $G_m$ do
8         Select marked transition $T$ in $G_m$
9         Set $T$ as visited
10        if $T \notin R_a$ then $A = A \cup \{T\}$
11        for all transitions $Q$ in $G_m$ such that $Q$ is not set as
              visited or marked do
12            if there is a dependence between $Q$ and $T$ in $G_m$
13            then Mark $Q$ in $G_m$
14        endfor
15  endwhile
16  $B = \varnothing$
17  Set all nodes (transitions) in $G_m$ as unmarked
18  for all $T \in R_a$ do Mark $T$ in $G_m$
19  for all $Q \in R_d$ do
20        for all transitions $T$ in $G_m$ do
21              if there is a dependence between $Q$ and $T$ in $G_o$
22              then Mark $T$ in $G_m$
23        endfor
24  endfor
25  for every pair ($T$, $Q$) of transitions in $M_m$ where $T \neq Q$ and
            $T, Q \notin R_a$ do
26        if (there is a dependence between $T$ and $Q$ in $G_m$) and
             (there is no dependence between $T$ and $Q$ in $G_o$)
27        then Mark $Q$ in $G_m$
28        if (there is a dependence between $T$ and $Q$ in $G_o$) and
             (there is no dependence between $T$ and $Q$ in $G_m$)
29        then Mark $Q$ in $G_m$
30  endfor
31  while there is a marked transition in $G_m$ do
32        Select marked $T$ in $G_m$
33        Set $T$ as visited
34        if $T \notin R_a$ then $B = B \cup \{T\}$
35        for all transitions $Q$ in $G_m$ such that $Q$ is not set as
              visited or marked do
36            if there is a dependence between $T$ and $Q$ in $G_m$
37            then Mark $Q$ in $G_m$
38        endfor
39  endwhile
40  Display affecting transitions of $A$ in $G_m$
41  Display affected transitions of $B$ in $G_m$
```

Fig. 10 Algorithm to compute affecting and affected transitions

In lines 2 and 3, the model dependence graph G_o of the original model and the model dependence graph G_m of the modified model are computed respectively.

In lines 4-15, the algorithm computes set A of affecting transitions defined as *AG(FM)* in definition 6. Initially, all added transitions in R_a are marked in the model dependence graph G_m (line 6). In the while-loop (lines 7- 15), the algorithm traverses backwards the model dependence graph G_m for each marked transition. In lines, 11-14, transitions Q for which there exists data or control dependence between Q and T in G_m are marked as well. At the termination of the while loop (lines 7-15) set A contains affecting transitions *AG(MF)*.

In lines 16-39, the algorithm computes set B of affected transitions defined as *AD(FM)* in definition 7. In this part of the algorithm we can identify two phases, the marking phase (lines 17-30), and the forward traversal phase (lines 31-39). In the marking phase, the algorithm identifies all transitions on which the model modifcation will have a direct or an indirect impact (as specified in the five cases presented in section 5.2). In the forward tranversal phase, the algorithm tranverses all marked transitions in order to compute *AD(FM)* as defined in definition 7.

In the marking phase, the algorithm starts by marking all transitions in R_a (line 18), this will take care of all transitions related to case 1. Forward traversal of these marked transitions will identify the set of affected transtions for each transtion Q in R_a ($\bigcup AD(Q)\ where\ Q\ \in C1$). In lines (21-22), the algorithm marks all transitions Q if there exists data or control dependence between Q and at least one deleted transition in R_d. This part of the algorithm takes care of the transitions related to case 4. The forward traversal of these transtions will identify the set of affected transtions for each transtion Q$\in C4$. In lines (26-27), the algorithm marks transitions involved in new dependencies that do exist in the modified model but do not exist in the original model. These are the transitions related to case 2.

Forward traversal of these transitions will identfiy the set of affected transitions for each transition Q $\in C2$. In lines (28- 29), the algorithm marks transitions involved in dependencies that cease to exist, i.e., dependencies that do exist in the original model but do not exist in the modified model. These transitions are related to case 3 and case 5. . Forward traversal of these transitions will identfiy the set of affected transitions for each transition Q $\in \{C3 \cup C5\}$ By the end of the forward traversal phase, set B contains all transitions affected by the model modification AD(MF).

5 Empirical study

The goal of this empirical study is to verify that the average size of the set of affecting transitions *AG(MF)* and affected transitions *AD(MF)* for a model modification is relatively smaller than the size of the original model in terms of number of transitions. The smaller the size of these sets is, the more practical they can be used by the developer. In this case, after a model is modified, it will be enough for the developer to validate only the transitions within these sets without having to look at all transitions of the modified model. Notice that in the worst case scenario the size of *AG(MF)* is the same as the size of *M*, and the size of *AD(MF)* is the same as the size of *M*. We want, however, in this experiment to verify that the average size of these sets is reasonably smaller than the size of *M*.
For this experiment, we have used six EFSM models. Due to the unavailability of system models for real world commercial software, we used EFSM system models that are in the public domain for the empirical study. These EFSM models are: an ATM model [10, 11, 53], a cruise control model [45], a fuel pump model [46], the Transfer Control Protocol-communication dialer (TCP) [44], Print-Token [24], and the Integrated Service Digital Network (ISDN) protocol [47]. The sizes of models range from 5 to 20 states and 20 to 89 transitions. The details about the models are shown in Table 1.

Table 1: System models used in the experiment

Model Name	TN	SN	NV
ATM	28	8	8
Cruise Control	20	5	18
Fuel Pumps	16	13	12
TCP-Dialer	50	17	31
ISDN	92	20	4
Print Token	98	11	5

TN:	number of transitions
SN:	number of states
NV:	number of variables

Our approach in this experiment is as follows: For a given model M of size n, we run n iterations of what-if-analysis using a tool that we developed for the purpose of this experiment. In each iteration, we consider a single transition in the model, and we analyze what-if this transition is edited. In this context, we consider editing a transition equivalent to deleting the transition, and adding a new transition. Therefore, for a single iteration of the analysis, R_d will contain one deleted transition, and R_a will contain one added transition. The result of a single stage of analysis is the set of affecting transitions AG(MF) and the set of affected transitions AD(MF).

The analysis we apply distinguishes between data dependence and control dependence. As a result, for each iteration of the analysis nine sets of transitions are generated: AD_d, AD_c, AD_{cd}, AG_d, AG_c, AG_{cd}, AGD_d, AGD_c, AGD_{cd} where:

- AD_d is the set of affected transitions, AD(MF), considering only data dependence,
- AD_c is the set of affected transitions, AD(MF), considering only control dependence,
- AD_{cd} is the set of affected transitions, AD(MF), considering both data and control dependence,
- AG_d is the set of affecting transitions, AG(MF), considering only data dependence,
- AG_c is the set of affecting transitions, AG(MF), considering only control dependence,
- AG_{cd} is the set of affecting transitions, AG(MF), considering both data and control dependence,
- AGD_d is the union of affecting and affected transitions, considering only data dependence ($AD_d \cup AG_d$),
- AGD_c is: the union of affecting and affected transitions, considering only control dependence ($AD_c \cup AG_c$)
- AGD_{cd} is: the union of affecting and affected transitions, considering both data and control dependence ($AGD_d \cup AGD_c$)

By the end of all *n* iterations of the analysis, we obtain a size matrix, *S*, summarizing the *n* assumed model modifications. The size matrix is $n\times9$, where *n* is the number of transitions in the input model, and 9 is the number of sets obtained in each iteration of the analysis. One row in this matrix represents one iteration of the analysis. A row has 9 values, each value represents the size of one of the resulting sets. For example, in the size matrix *S*, the value of the entry $S_{3,5}$ will represent the size of the fifth resulting set of the third iteration of the analysis. In this case, the value is the size of the set of affecting transitions for control dependence: $|AD_c|$, assuming a modification in transition T_3 of the input model. Table 2 represents the size matrix obtained for the fuel pump model.

Table 2 Size of affecting and affected transitions for fuel pump model

	Affecting AG(MF)			**Affected AD(MF)**			**AG(MF) U AD(MF)**		
	AG_d	AG_c	AG_{cd}	AD_d	AD_c	AD_{cd}	AGD_d	AGD_c	AGD_{CD}
T1	0	0	0	0.31	0	0.31	0.31	0	0.31
T2	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0
T4	0	0	0	0.25	0.38	0.56	0.25	0.38	0.56
T5	0	0.13	0.13	0	0	0	0	0.13	0.13
T6	0.06	0	0.06	0.31	0.38	0.63	0.31	0.38	0.63
T7	0.06	0.13	0.19	0.19	0.19	0.38	0.25	0.31	0.56
T8	0.06	0.13	0.19	0.19	0.19	0.38	0.25	0.31	0.56
T9	0	0.25	0.25	0.19	0	0.19	0.19	0.25	0.44
T10	0.44	0	0.44	0.19	0	0.19	0.56	0	0.56
T11	0.13	0.25	0.25	0	0	0	0.13	0.25	0.25
T12	0.44	0	0.44	0	0	0	0.44	0	0.44
T13	0.44	0	0.44	0	0	0	0.44	0	0.44
T14	0	0.25	0.25	0	0	0	0	0.25	0.25
T15	0	0	0	0	0	0	0	0	0
T16	0	0	0	0	0	0	0	0	0
Avg	**0.1**	**0.07**	**0.16**	**0.1**	**0.07**	**0.16**	**0.2**	**0.14**	**0.32**

The size is expressed as a percentage of the total size of the model. Looking at transition T_4, we notice that the values of the first three columns are zeros, which means that there are no transitions affecting a model modification involving only transition T_4. On the other hand, 25% of the transitions in the model will be affected by the modification (considering data dependence), and 38% of the transitions in the model will be affected by the modification (considering control dependence). This leads to a combined total of 56% of the transitions affected by the modification (either based on data dependence or control dependence). We notice that the size of AG_{cd} is not exactly equal to the sum of $|AG_c|$ and $|AG_d|$ since some transitions are included in both sets (AG_d and AG_c). Looking at column AGD_{cd}, the worst case modification scenario is for transition T_6 with 63% of the transitions either affecting the modification or affected by the modification. Namely, 31% (5 out of 16) of the transitions are affecting the modification, and 38% (6 out of 16) of the transitions are affected by the modification.

It is worth mentioning here that the average of affecting and affected transitions for the whole model is the same since the "affects" relationship is a binary bidirectional relation. So, whenever the relation "T_i affects T_k" appears in the model, we get T_i in the set of affecting transitions for T_k, and we get T_k in the set of affected transitions for T_i.

To better understand the analysis results, box-plots of the average size values for each model modification of each EFSM model are shown in figure 11. Additionally, the last box-plot BP#7 represents the cumulative average size values for all modifications in all EFSM models.

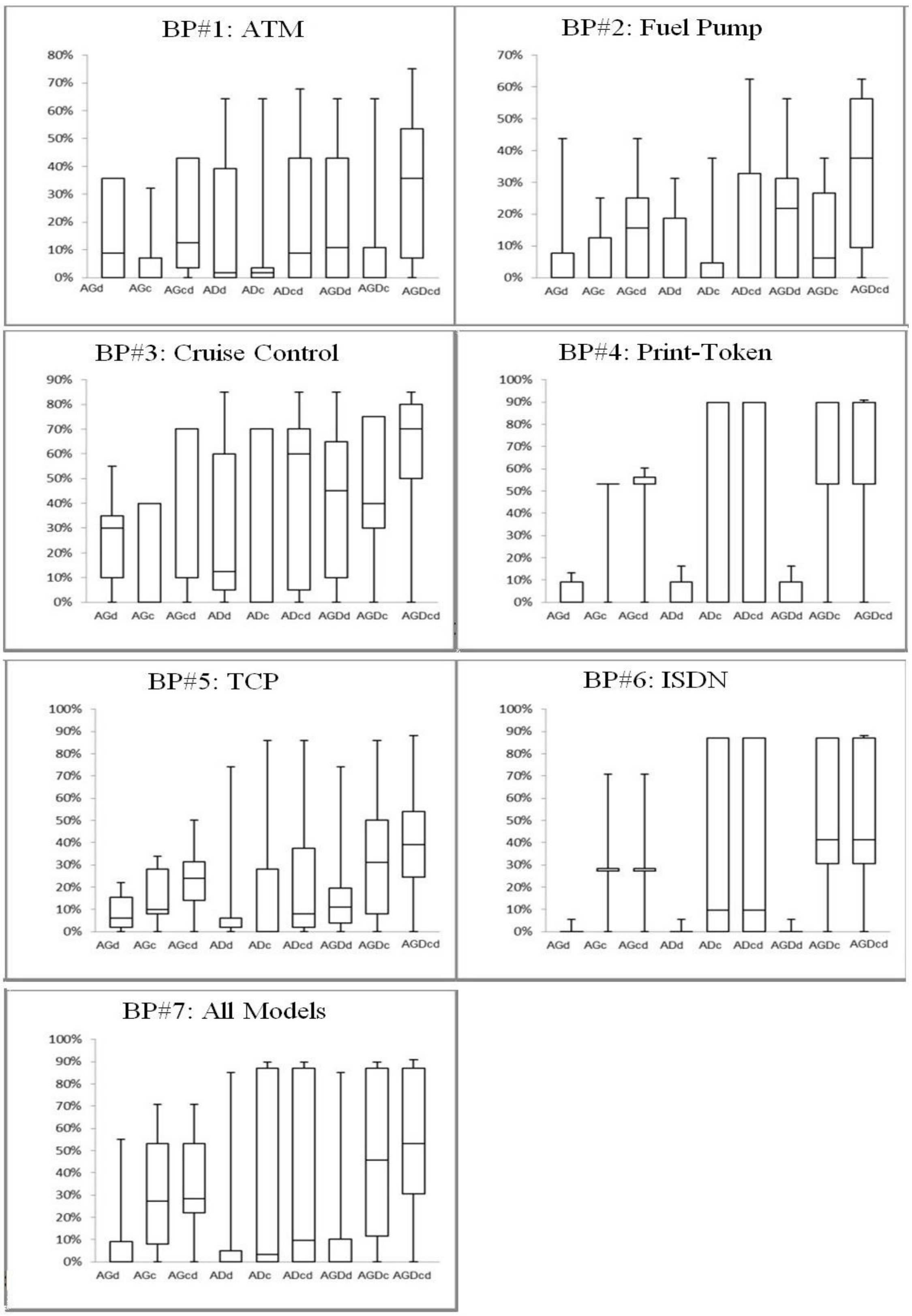


Fig. 11Box-plots of the affected parts of the models

The analysis results represented in all of the seven box plots indicate that our approach significantly help developers identifying the effect of the model on the modification AG(*FM*), and identifying the effect of the modification on the remaining part of the model AD(*FM*). Looking at BP#7, AG_{cd} shows that only 25% (the third quartile) of the model modifications are affected by more than 53% of the transitions in *M*. Another 25% (the second quartile) of the modifications are affected by more than 28% of the transitions in *M* (but less than 53%), and the remaining 50% of the modifications are affected by less than 28% of the transitions in *M*.

Looking at AD_{cd}, we notice that 25% of the model modifications (the third quartile) are affecting more than 86% of the transitions on *M*, another 25% of the model modification (the second quartile) are affecting less than 86% of the transitions in *M*, and the remaining 50% of the modifications are affecting less than 10% of the transitions in M. These numbers indicate that, excluding the top 25% transactions of a model, the developer can look at a considerably smaller dependence graph when testing the model after a modification.

Table 3 summarizes the average size of the sets of affecting and affected transitions for all model modifications applied on all six models.

Table 3: Average size of the sets of affecting and affected transitions for all models

	Affecting AG(MF)			**Affected AD(MF)**			**AG(MF) ∪ AD(MF)**		
Models	**AG_d**	**AG_c**	**AG_{cd}**	**AD_d**	**AD_c**	**AD_{cd}**	**AGD_d**	**AGD_c**	**AGDcd**
ATM	0.16	0.06	0.21	0.16	0.06	0.21	0.21	0.11	0.32
Fuel Pump	0.1	0.07	0.16	0.1	0.07	0.16	0.2	0.14	0.32
Cruise Control	0.25	0.28	0.48	0.25	0.28	0.48	0.4	0.44	0.59
Print Token	0.03	0.48	0.49	0.03	0.48	0.49	0.04	0.67	0.68
TCP	0.08	0.16	0.22	0.08	0.16	0.22	0.14	0.32	0.41
ISDN	0	0.28	0.28	0	0.28	0.28	0	0.48	0.48
Average	**0.1**	**0.22**	**0.31**	**0.1**	**0.22**	**0.31**	**0.17**	**0.36**	**0.47**

Figure 12 represents Box plot for the data in table 3. From AGD_{cd}, we can conclude that on average, 25% of the modifications (third quartile) are affecting/affected by 56% to 68% of the transitions in *M*. Another 25% of the modifications (second quartile) are affecting/affected by 44% to 55% of the transitions in *M*. Another 25% of the modifications are affecting/affected by 31% to 43% of the transitions in *M*. The last 25% of the modifications affect/are affected by less than 31% of the transitions in *M*.

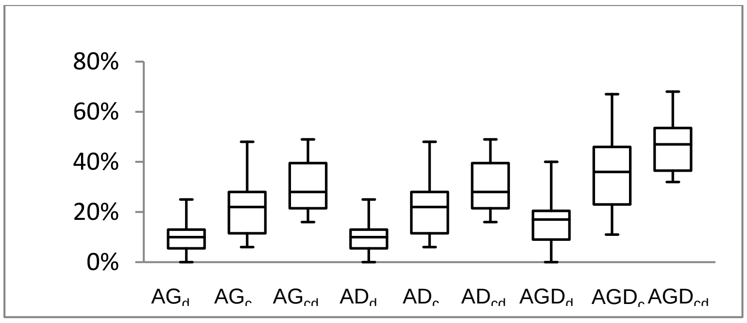


Fig. 12 Average size of affecting/affected transitions for all six EFSM models

The significance of our approach becomes more evident when we categorize transitions of an EFMS model into four categories: high-impact transitions, medium-impact transitions, low-impact transitions, and no impact transitions. These categories are based on the number of affecting transitions and affected transitions expressed as: |AG(MF) ∪ AD(MF)|. Table 4 declares the boundaries of each category. According to table 4, a transition is categorized as high-impact transition if modifying the transition generates a set of affecting/affected transitions >=60% of the original size of the model.

Table 4 High-impact, medium-impact, low-impact, and no-impact transitions

Category	Criteria
High-impact	|AG(MF) U AD(MF)| > = 0.60 * |M|
Medium-impact	|AG(MF) U AD(MF)| > = 0.30 * |M| And |AG(MF) U AD(MF)| < 0.60 * |M|
Low-impact	|AG(MF) U AD(MF)| > 0 And |AG(MF) U AD(MF)| < 0.30 * |M|
No-impact	|AG(MF) U AD(MF)| = 0

Figure 13 categorizes all transitions of the six models according to the impact of the modification. We can clearly see that cumulatively, less than 40% of the transitions are categorized as high-impact transitions, which means that for the remainder of the transitions the developer will be able to validate a change by looking at a sub-graph of less than <60% of the original size of the model.

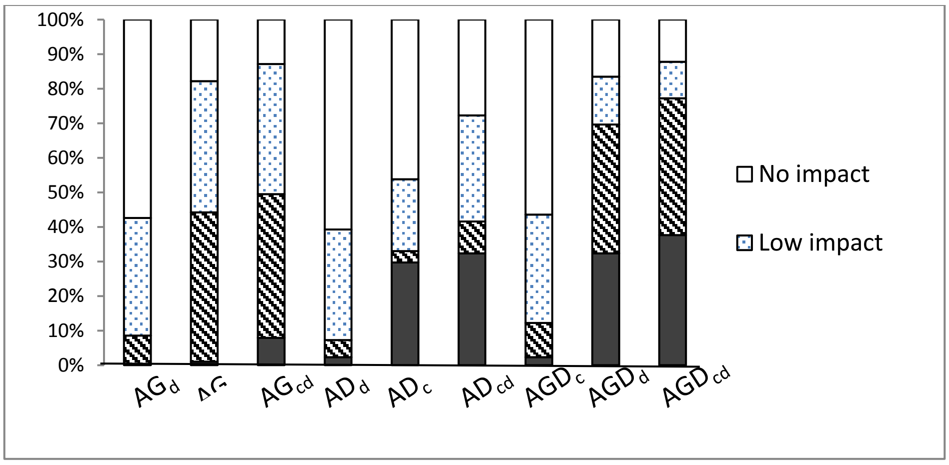


Fig. 13: Size of each of the categories of transitions based on the modification impact

6 Related research

There has been a significant amount of research that uses system model for several applications such as: managing requirements changes [48], regression testing [3, 4, 7, 26], model slicing [34, 46, 47], and test prioritization and reduction [36, 50].

Korel and Tahat [43] presented an approach toward understanding modifications on model-based systems. the approach uses the original and the modified models to automatically identify the difference between them. The modification in the model might be as a result of maintenance, error correction, or a change in functionality driven by a change in requirements. This paper is an extension to Korel et al's work on understanding model modifications. In this paper, we have formally defined "model modification", and we have formally defined the relationship "affects" which is the core concept toward understanding the effect of model modifications. We have explored different possible cases of model modifications, where each case focuses on a specific effect of the modification. Finally, we performed an empirical study involving six EFSM system models to show the effectiveness of our approach.

Lin et al. [48] introduced a technique for a requirement change management. Requirements change management is one of the most difficult problems to deal with in requirements tracking. They proposed a set of algorithms for managing all possible automatic requirements changes. Their approach used state machines to model and manage requirements changes.

Briand et al. [1] introduced a Regression Test Selection Tool which used systems modeled in UML. More recently, this work has been developed and extended into a comprehensive study of the regression test selection problem for UML [2]. Orso et al. [3] presented two regression test selection techniques for component-based systems, using component metadata to support the identification of selective subsets of tests to be used in efficient re-testing strategies. Their approach was illustrated in terms of component-based systems specified as UML statecharts and was evaluated using two real world java systems. Their approach used state based models, similar to our approach. Farooq et al. [4] also recently presented a regression test selection approach based on changes identified in both the statecharts and class diagrams of the UML.

Wu and Offutt [5] presented retesting strategies for UML, based on a differencing approach that identified the modified parts of the new model. Similar to our work, Wu and Offutt also incorporated the tracing of data dependence based changes in their work. Pilskalns et al. [6] presented a safe and efficient re-test strategy based on UML model level changes, illustrating their approach with a case study of an open source system called Batik. Their work is concerned with regression testing the model itself.

Korel et al. [7] presented methods of test prioritization based on the state-based model of the system under test. These methods assumed that the modifications were made on both the system under test and its model. The existing test suite was executed on the system model and information about this execution was used to prioritize tests. Korel et al. extended their research on model-based prioritization for a class of modifications for which models were not modified (only the source code is modified) [8, 9]. Several model-based test prioritization heuristics were introduced. Their major motivations for these heuristics were simplicity and effectiveness in early fault detection. The results of their study suggested that system models might improve the effectiveness of test prioritization with respect to early fault detection. Korel et al. [10] proposed simple model-based test prioritization heuristics. The major stress was on simplicity. These simple heuristics have shown promise when a large number of transitions was modified. However, for small modifications the performance of these heuristics could be equivalent to the selective prioritization -Version II.

Korel et al. [34] presented a technique for slicing EFSMs that used data and control dependence for EFSM. They first constructed a dependence graph by using dependence relations. Then the algorithm started from the node in the dependence graph representing the slicing criterion and nodes (i.e. transitions) that were backward reachable from the slicing criterion in the dependence graph were marked in the slice. Once the transitions in the slice have been marked, Korel et al. [34] have implemented different algorithms for automatically reducing the size of an EFSM slice.

7 Conclusion and future work

In this paper, we have presented an approach toward understanding modifications in model-based systems. The approach can be used for any modification of the system model. The goal is to identify these parts of the model that may exhibit different behavior because of the modification. The approach uses the original model and the modified model and automatically identifies a difference between these models. Our introduced approach uses model-based dependence analysis to identify the affected parts of the model and the parts that may affect the modified transitions. Our empirical study shows that this approach helps developers in understanding the effect of model modification by providing them with a set of affecting transitions and a set of affected transitions with a significantly smaller number of transitions compared to the total number of transition in the model. During evolutionary software maintenance, system models are modified to fix defects, to enhance or change functionality, to add new functionality, or to delete the existing functionality. The presented approach does not only help in understanding the impact of the modification on the system, but also facilitates isolating parts of the modified model that may contribute to a faulty behavior. We have developed a tool to compute the parts of the model that are affected by the modification and the parts of the model that affect the modification. Additionally, the tool generates subsets of affected/affecting transitions based on the type of dependency.

The results of the empirical study indicate that our presented approach may significantly help in understanding the effect of modifications on the system. In the future, we plan to perform an experimental study in which we will investigate advantages and limitations of the presented approach in understanding modifications of large state-based models. In addition, we plan to investigate the mapping between the source code of the system and the model.

References

[1] Lionel C. Briand, Yvan Labiche and G. Soccar. Automating impact analysis and regression test selection based on UML designs. International Conference on Software Maintenance (ICSM), Montréal, Canada, pages 252-261, 2002.

[2] Lionel C. Briand, Yvan Labiche and S. He. Automating regression test selection based on UML designs. Information and Software Technology. 51(1):16-30, 2009.

[3] Alessandro Orso, Hyunsook Do, Gregg Rothermel, Mary Jean Harrold, David S. Rosenblum. Using component metadata to regression test component-based software. Software Testing Verification and Reliability (STVR). 17(2):61-94, June 2007.

[4] Qurat-ul-ann Farooq, Muhammad Zohaib Z. Iqbal, Zafar I Malik andAamer Nadeem. An approach for selective state machine based regression testing. Proceedings of the 3rd International ACM Workshop on Advances in Model-Based Testing, London, United Kingdom, pages: 44 – 52, 2007.

[5] Ye Wu and Jeff. Offutt. Maintaining evolving component-based software with UML. Seventh European Conference on Software Maintenance and Reengineering(CSMR), pages 133- 142, 2003.

[6] Orest Pilskalns, Gunay Uyan, Anneliese Andrews. Regression Testing UML Designs. 22nd IEEE International Conference on Software Maintenance, Philadelphia, Pennsylvania, USA, pages: 254 – 264, 2006.

[7] B. Korel, L. Tahat, M. Harman. Test Prioritization Using System Models. Proc. IEEE International Conference on Software Maintenance. Budapest, Hungary, pp. 559-568, 2005.

[8] B. Korel, G. Koutsogiannakis, L. Tahat. Model-Based Test Prioritization Heuristic Methods and Their Evaluation. IEEE International Conference on Software Maintenance. ICSM 2008, Beijing, China, September. 2008, pp: 247-256.

[9] B. Korel, G. Koutsogiannakis. Experimental Comparison of Code-Based and Model-Based Test Prioritization, 5th Workshop on Advances in Model Based Testing, A-MOST 2009, Denver, April 2009, IEEE digital library."

[10] B. Korel, G. Koutsogiannakis, L. Tahat. Prioritization Algorithms for Regression Testing in Model Based Systems. Proc. the 3rd ACM Workshop on Advances in Model Based Testing (A-MOST), London, United Kingdom, pp. 34-43, 2007.

[11] Douglass B. P. "UML Statecharts", ESP Jan-1999. I-Logix.

[12] Harel D. "Statecharts: A Visual Formalism for Complex Systems". Science of Computer Programming. Vol. 8 (1987), pp. 231-274.

[13] Harel D. "From Play-In Scenarios to Code: An Achievable Dream," Proc. Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science, Vol. 1783, 2000, pp. 22-34.

[14] Savage, P., Walters, S., Stephenson, M., "Automated Test Methodology for Operational Flight Programs," Proceedings of IEEE Aerospace Conference, vol. 4, pp. 293 - 305, 1997.

[15] Dssouli, R., Saleh, K., Aboulhamid, E., En-Nouaary, A., Bourhfir, C., "Test Development For Communication Protocols: Towards Automation," Computer Networks, 31, pp. 1835 – 1872, 1999.

[16] Bourhfir, C., Aboulhamid, E., Khendek, F., Dssouli, R., "Test Case Selection from SDL Specifications," Computer Networks, 35(6), pp. 693 – 708, 2001.

[17] Holzman, G., Design and Validation of Protocols, Prentice-Hall, 1990.

[18] F. Wagner, "VFSM Executable Specification," Proc. CompEuro, 1992, pp. 226-231.

[19] Lee, D., Lee, J., "A Well-Defined Estelle Specification for Automatic Test Generation," IEEE Trans. on Communications, 40, pp. 526 - 542, 1991.

[20] K. Cheng, A. Krishnakumar, Automatic Functional Test Generation Using The Extended Finite State Machine Model, Proc. ACM/IEEE Design Automation Conf.,, Dallas, TX, USA, pp. 86-91, 1993.

[21] ITU-T. Recommendation Z.100 Specification and description language (SDL). International Telecommunications Union, Geneva, Switzerland, 1999.

[22] J. Dick, A. Faivre, Automating the Generation and Sequencing of Test Case from Model-Based Specification, Proc. International Symposium on Formal Methods, San Francisco, CA, USA, pp. 268-284, 1992.

[23] R. Dssouli, K. Saleh, E. Aboulhamid, A. En-Nouaary, C. Bourhfir, Test Development For Communication Protocols: Towards Automation, Computer Networks, 31, pp.1835-1872, 1999.

[24] H. Ural, K. Saleh, A.W. Williams. Test generation based on control and data dependencies within system specifications in SDL, Computer Communications 2000; 23(7): 609□627.

[25] Y. Duale, M. U. Uyar. A method enabling feasible conformance test sequence generation for EFSM models. IEEE Transactions on Computers, 53(5):614–627, 2004.

[26] Vaysburg, L. Tahat, B. Korel, Dependence Analysis in Reduction of Requirement Based Test Suites, Proc. ACM International Symposium on Software Testing and Analysis, Rome, Italy, pp. 107-111, 2002.

[27] Friedman, G., Hartman, A., Nagin K., Shiran, T., "Projected State Machine Coverage for Software Testing," Proc. of the ACM Intern. Symposium on Software Testing and Analysis, pp. 134 – 143, 2002.

[28] Heimdahl, M., Whalen, M., "Reduction and Slicing of Hierarchical State Machines," ACM SIGSOFT Software Engineering Notes, 22(6), pp. 450 – 467, 1997.

[29] Heimdahl, M., Thompson, J., Whalen, M., "On Effectiveness of Slicing Hierarchical State Machines: A Case Study," Proc. of the 24th Euromicro Conference, pp. 435 – 444, 1998.

[30] Oda, T., Araki, K., "Specification Slicing in Formal Methods of Software Engineering," Proc. of the 7th Intern. Computer Software and Applications Conference, 1993.

[31] Tahat, L., Vaysburg, B., Korel, B., Bader, A., "Requirement-Based Automated Black-Box Test Generation," Proc. of the 25th Annual IEEE Intern. Computer Software and Applications Conference (COMPSAC), pp. 489 - 495, 2001.

[32] Vaysburg, B., Tahat, L., Korel, B., Bader, A., "Automating Test Case Generation from SDL Specifications," Proc. of 18th Intern. Conf. on Testing Computer Software, pp. 130 – 139, 2001.

[33] Vaysburg, B., Tahat, L., Korel, B., "Dependence Analysis in Reduction of Requirement Based Test Suites," Proc. of the ACM Intern. Symposium on Software Testing and Analysis, pp. 107 – 111, 2002.

[34] Korel, B., Singh, I., Tahat, L., Vaysburg, B., "Slicing of State -Based Models," IEEE Intern. Conf. on Software Maintenance, pp. 34-43, 2003.

[35] Lyle, J., Weiser, M., " Experiments on Slicing-based Debugging Tools," 1st Conference on Empirical Studies of Programming, pp. 187 - 197, 1986.

[36] Korel, B., Tahat, L., Vaysburg, B., "Model Based Regression Test Reduction Using Dependence Analysis," Proc. of the Intern. IEEE Conf. on Software Maintenance, 2002, pp. 214-223.

[37] Dick, J., Faivre, A., "Automating the Generation and Sequencing of Test Case from Model-Based Specification," Proc. of the Industrial Strength Formal Methods, 5th Intern. Symposium on Formal Methods, pp. 268 – 284, 1992.

[38] Carver, R., H., Tai, K., C., "Use of Sequencing Constraints for Specification-Based Testing of Concurrent Programs," IEEE Trans. on Software Engineering, 24(6), pp. 471 – 490, 1998.

[39] Apfelbaum, L., "Spec-based Tests Make Sure Telecom Software Works," IEEE Spectrum Magazine, 34(11), pp. 77 – 83, 1997.

[40] Dalal, S., Jain, A., Karunanithi, N., Leaton, J., Lott, C., Patton, G., Horowitz, B., "Model-based Testing and Practice," in Proc. of the Intern. Conference on Software Engineering (ICSE), pp. 185 – 194, 1999.

[41] Cheng, K., Krishnakumar, A., "Automatic Functional Test Generation Using The Extended Finite State Machine Model", The 30th ACM/IEEE Design Automation Conf., pp. 86 – 91, 1993.

[42] Budkowski, T., Dembinski, P., “An Introduction to Estelle: A Specification Language for Distributed Systems,” Comp. Netw. & ISDN, 14(1), pp. 3-24, 1987.

[43] Luay Tahat, Bogdan Korel, Mark Harman, and Hasan Ural,"Regression test suite prioritization using system models", Journal of Software Testing, Verification, and Reliability, in press 2011.

[44] Bogdan Korel, Luay Tahat, “Understanding Modification in State-Based System”, Proceeding of the 12th IEEE International Conference on Program Comprehension (IWPC’04), London, UK, September 2004: pages:246-250.

[45] Ferrante K., Ottenstein K., Warren J., “The Program Dependence Graph and its Use in Optimization,” ACM Transactions on Programming Languages and Systems, 9(5), pp. 319 – 349, 1987.

[46] K. Androutsopoulos, et al. *Control Dependence for Extended Finite State Machines*. Proc. Fundamental Approaches to Soft. Eng. (FASE '09) , York, UK, 22nd-29th March, 2009. Springer LNCS volume 5503, pages 216-230.

[47] K. Androutsopoulos, N. Gold, M. Harman, Z. Li, and L. Tratt. *A Theoretical and Empirical Study of EFSM Dependence*. Proc. 25th IEEE International Conference on Software Maintenance (ICSM 2009), Edmonton, Alberta, Canada, 23rd-26th September 2009, pp.287-296.

[48] L. Lin, S. J. Prowell2, J. H. Poore1, "The impact of requirements changes on specifications and state machines ", Software: Practice and Experience Volume 39, Issue 6, pages 573–610, 25 April 2009.

[49] Tahat L, Korel B., Hartman M, Ural H., “Regression Test Suite Prioritization Using System Models”, Software Testing, Verification, and Reliability Journal (STRV), Wiley Inter science, special edition on Model-Based Testing, in press 2011, DOI: 10.1002/stvr.461

[50] Basili, V. R., “Viewing Maintenance as Reuse-Oriented Software Development”, IEEE Software, 7(1):19-25, 1990.