Towards Minimization of Test Sets for Coverage Testing of Interactive Systems

Fevzi Belli, Christof J. Budnik

University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany
{belli, budnik}@adt.upb.de

Abstract. A model-based approach for minimization of test sets for interactive systems is introduced. Test cases are efficiently generated and selected to cover the behavioral model and, complementarily, the fault model of the system under test (SUT). Structural features of the SUT are exploited in order to decrease the complexity of the introduced algorithms. The approach is extended to UML statecharts to increase its applicability.

1. Introduction

Testing is the traditional validation method in the software industry. There is no justification, however, for any assessment on the correctness of the SUT based on the success (or failure) of a single test, because there can potentially be an infinite number of test cases, even for very simple programs. To overcome this shortcoming of testing, formal methods have been proposed, which introduce models that represent the relevant features of the SUT. The modeled, relevant features are either functional behavior or the structural issues of the SUT, leading to *specification-oriented testing* or *implementation-oriented testing*, respectively. This paper is on specification-oriented testing; i.e., the underlying model represents the system behavior interacting with the user's actions. The system's behavior and user's actions will be viewed here as *events,* more precisely, as *desirable events* if they are in accordance with the user expectations. Moreover, the approach includes modeling of the faults as *undesirable events* as, mathematically spoken, a complementary view of the behavioral model.

Based on [Be01], this paper introduces a novel, graphical representation of both the behavioral model and the fault model of the SUT. Algorithms are introduced for the coverage of these models by a minimal set of test cases (*minimal spanning set for coverage testing*) which are of less complexity as the ones known in the literature. The next section summarizes the related work before Section 3 introduces the fault model and the test process. The optimization of the test suite is discussed in Section 4.

Section 5 considers the structure of the SUT to avoid unnecessary and/or infeasible tests. This is necessary to considerably decrease the complexity and to avoid the state explosion problem, which requires also the model be hierarchically structured and consist of a tractable number of nodes. Section 6 extends the approach to UML statecharts. as they are solid, graphical means that are also both popular and in accordance with the principle of software engineering, e.g., modularization ("divide et impera"). The achieved results are summarized in Section 7. Section 8 concludes the paper and sketches the research work planned.

2. Related Work

Methods based on finite-state automata have been used for almost four decades for the specification and testing of system behavior, e.g., for specification of software systems [Ch78], as well as for conformance and software testing [Bi00, ADL91, Sa89, OSA03]. Also, the modeling and testing of interactive systems with a state-based model has a long tradition [Pa69, JNC03, SS97, WA00]. These approaches analyze the SUT and model the user requirements to achieve sequences of *user interaction* (*UI*) which then are deployed as test cases. [WA00] introduced a simplified state-based, graphical model to represent UIs; this model has been extended in [Be01] to consider not only the desirable situations, but also the undesirable ones. This strategy is quite different from the combinatorial ones, e.g., *pairwise testing*, which requires that for each pair of input parameters of a system, every combination of these parameters' valid values must be covered by at least one test case. It is, in most practical cases, not feasible to test UIs [TL02]. A similar fault model as in [Be01] is used in the mutation analysis and testing approach which systematically and stepwise modifies the SUT using mutation operations [DLS78]. Although originally applied to implementation-oriented unit testing, mutation operations have also been extended to be deployed at more abstract, higher levels, e.g., integration testing, state-based testing, etc. [DMM01]. Such operations have also been independently proposed by other authors, e.g., "state control faults" for fault modeling in [BP94], or for "transition-pair coverage criterion" and "complete sequence criterion" in [OSA03]. However, the latter two notions have been precisely introduced in [Be01] and [WA00], respectively, earlier than in [OSA03]. A different approach, especially for graphical user interface (GUI) testing, has been introduced in [MPS00]; it deploys methods of knowledge engineering to generate test cases, test oracles, etc., and to deal with the test termination problem. All of these approaches use some heuristic methods to cope with the state explosion problem

This paper also presents a method for test case generation and test case selection. Moreover, it addresses test coverage aspects for test termination, based on [Be01], which introduced the notion of "minimal spanning set of complete test sequences", similar to "spanning set", that was later also discussed in [MB03]. The present paper considers existing approaches to optimize the round trips, i.e., the Chinese Postman Problem [ADL91], and determines algorithms of less complexity for the spanning of walks, rather than tours, related to [We96, NT81].

Statecharts [Har87] have become very popular in software construction. Several approaches formalize their semantics, e.g., by extended finite state machines (EFSMs), or flow graphs [HKC00]. Based on resulting, mathematically sound models, test cases can be generated [HKC00, OA99]. The present paper extends these approaches by integrating the complementary view into the fault model.

3. Fault Model and Test Process

This work uses *Event Sequence Graphs (ESG)* for representing the system behavior and, moreover, the facilities from the user's point of view to interact with the system. Basically, an *event* is an externally observable phenomenon, such as an environmental or a user stimulus, or a system response, punctuating different stages of the system activity.

3.1 Preliminaries

Definition 1. An Event Sequence Graph ESG=(V,E) is a directed graph with a finite set of *nodes* (*vertices*) $V \neq \emptyset$ and a finite set of *arcs* (*edges*) $E \subseteq V \times V$.

For representing user-system interactions, the nodes of the ESG are interpreted as events. The operations on identifiable components of the UI are controlled/perceived by input/output devices, i.e., elements of windows, buttons, lists, checkboxes, etc. Thus, an event can be a user input or a system response; both of them are elements of V and lead interactively to a succession of user inputs and system outputs.

Definition 2. Let V, E be defined as in Def. 1. Then any sequence of nodes $\langle v_0,\ldots,v_k\rangle$ is called an (legal) *event sequence* (*ES*) if $(v_i, v_{i+1}) \in E$, for $i=0,\ldots,k-1$.

Furthermore, α (*initial*) and ω (*end*) are functions to determine the initial node and end node of an ES, i.e., $\alpha(ES)=v_0$, $\omega(ES)=v_k$. Finally, the function l(*length*) of an ES determines the number of its nodes. In particular, if $l(ES)=1$ then $ES=\langle v_i\rangle$ is an ES of length 1. An $ES=\langle v_i, v_k\rangle$ of length 2 is called an *event pair* (*EP*). The assumption is made that there is an ES from the single node ε to all other nodes, and from all nodes there is an ES to the single node γ ($\varepsilon, \gamma \notin V$). ε is called the *entry* and γ is called the *exit* of the ESG.

The entry and exit, represented by '[' and ']', respectively, are not included in V. They enable a simpler representation of the algorithms to construct minimal spanning test case sets (Section 4). ESGs visualize the functionality and external behavior of the system. Thus, they should be produced during design of the system, long before it is implemented. However, the approach can also be applied to any formally sound visualization means, e.g., statecharts, as demonstrated in Section 6.

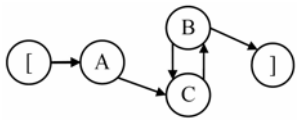

Figure 1: An ESG with [as entry and] as exit

Definition 3. An ES is called a *complete ES* (*Complete Event Sequence*, *CES*), if $\alpha(ES)=\varepsilon$ is the entry and $\omega(ES)=\gamma$ is the exit.

CESs represent *walks* from the entry "*[*" of the ESG to its exit "*]*".

Definition 4. Given an ESG, say $ESG_1 = (V_1, E_1)$, a *refinement* of ESG_1 through vertex $v \in V_1$ is an ESG, say $ESG_2 = (V_2, E_2)$. Let $N^+(v)$ be the *set of all successors* of v, and $N^-(v)$ be the *set of all predecessors* of v. Also let $N^-(ESG_2)$ be the *set of all EPs from start ([) of* ESG_2, and $N^+(ESG_2)$ be the *set of all EPs from* ESG_2 *to exit (]) of* ESG_2. Then there should be given an *one-to-many mapping* from ESG_2 to ESG_1, $N^+(ESG_2) \rightarrow N^+(v)$ and $N^-(ESG_2) \rightarrow N^-(v)$.

Figure 2 shows a refinement of vertex *a* in ESG_1 given as ESG_2, and the resulting new ESG_3.

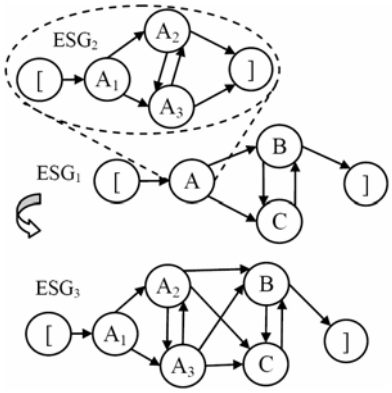

Figure 2: A Refinement of the vertex *a* of the ESG given in Fig. 1

3.2 Fault Model and Test Terminology

Definition 5. For an *ESG=(V, E)*, its *completion* is defined as $\widehat{ESG} = (V, \widehat{E})$ with $\widehat{E} = V \times V$.

Definition 6. The *inverse* (or *complementary*) *ESG* is then defined as $\overline{ESG} = (V, \overline{E})$ with $\overline{E} = \widehat{E} \setminus E$ (\: set difference operation).

Note: Entry and exit are not considered while constructing the $\overline{ESG}$.

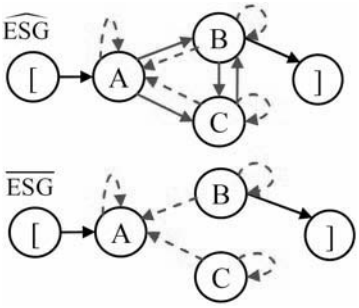

Figure 3: The completion $\widehat{ESG}$ and inversion $\overline{ESG}$ of Figure 1

Definition 7. Any EP of the $\overline{ESG}$ is a *faulty event pair* (*FEP*) for *ESG*.

Definition 8. Let $ES=\langle v_0, \ldots, v_k \rangle$ be an event sequence of length $k+1$ of an ESG and $FEP=\langle v_k, v_m \rangle$ a faulty event pair of the according ESG. The concatenation of the ES and FEP forms then a *faulty event sequence* $FES=\langle v_0, \ldots, v_k, v_m \rangle$.

Definition 9. An *FES* will be called *complete* (*Faulty Complete Event Sequence*, *FCES*) if $\alpha(FES)=\varepsilon$ is the entry. The ES as part of a FCES is called a *starter*.

3.3 Test Process

Definition 10. A *test case* is an ordered pair of an input and expected output of the SUT. Any number of test cases can be compounded to a *test set* (or, a *test suite*).

Once a test set has been constructed, tests can be run applying the test cases to the SUT. If it behaves as expected, the SUT *succeeds* the test, otherwise it *fails* the test. The approach introduced in this paper uses event sequences, more precisely CES, and FCES, as test inputs. If the input is a CES, the SUT is supposed to proceed it and thus, to succeed the test. Accordingly, if a FCES is used as a test input, a failure is expected to occur. The latter case represents an exception that must be properly handled by the system, i.e., the SUT is supposed to refuse the proceeding and produce a warning. The test process is sketched in Algorithm 1.

Algorithm 1. Test Process

n:= number of the functional units (modules) of the system that fulfill a well-defined task
length:= required length of the test sequences
FOR function 1 TO n DO
  Generate appropriate ESG and $\overline{\text{ESG}}$
  FOR k:=2 TO length DO //Section 4.2
    Cover all ESs of length k by means of CESs subject to minimizing the number and total length of the CESs //Section 4.1
  Cover all FEPs of by means of FCESs subject to minimizing the total length of the FCESs //Section 4.3
  Apply the test set to the SUT.
  Observe the system output to determine whether the system response is in compliance with the expectation.

To determine the point in time in which to stop testing, a criterion is necessary to systematize the test process and to judge the efficiency of the test cases. The approach converts this problem into the *coverage of the ES and FES of length k of the* $\widehat{ESG}$. The test costs are given by the minimized total length of the CESs and FCESs. The length of the ESs can be increased stepwise. This enables a scalability of the test costs which are proportional to the length of the ESs.

4. Minimizing the Spanning Set

The union of the sets of CESs of minimal total length to cover the ESs of a required length is called *Minimal Spanning Set of Complete Event Sequences (MS^{2}CES)*. If a CES contains all EPs at least once, it is called an *entire walk*. A legal entire walk is *minimal* if its length cannot be reduced. A minimal legal walk is *ideal* if it contains all EPs exactly once. Legal walks can easily be generated for a given ESG as CESs, respectively. It is not, however, always feasible to construct an entire walk or an ideal walk which are convenient if the system cannot be reset.

4.1 An Algorithm to Determine Minimal Spanning Complete Event Sequence

The determination of MS^{2}CES represents a derivation of the *Directed Chinese Postman Problem (DCPP)*, which has been studied thoroughly, e.g., in [ADL91, Th03]. The

MS^{2}CES problem introduced here is expected to have a lower complexity grade, as the edges of the ESG are not weighted, i.e., the adjacent vertices are equidistant. In the following, some results are summarized that are relevant to calculate the test costs and enable scalability of the test process.

For the determination of the set of minimal tours that covers the edges of a given graph, the algorithm described in [Th03] requires this graph be strongly connected. This can be reached for any ESG through an additional edge from the exit to the entry that resets the system and is (or, should be) always possible in interactive systems. The idea of transforming the ESG into a strongly connected graph is depicted in Figure 4 as a dashed arc. The figures within the vertices indicate the balance of these vertices as the difference of the number of outgoing edges and the number of the incoming edges. These balance values determine the minimal number of additional edges from "+" to "-" that will be identified by searching the all-shortest-path and solving the optimization problem [AMO93] by the Hungarian method [Kn93]. The required additional edge for the ESG in Figure 4 is represented as a dotted arc. The problem can then be transferred to the construction of the Euler tour for this graph [We96]. Each occurrence of the ES=*]]* in the Euler tour identifies another separate test case. The algorithm to determine minimal spanning set of complete event sequences (MS^{2}CES) consists of three sections:

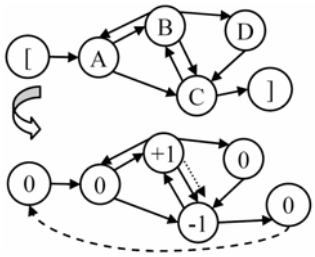

Figure 4: Transferring walks into tours and balancing the nodes

- Determination of all-shortest-paths by Floyds algorithm with the complexity O(|V|^3) [We96]. However, because the ESG is a non-weighted digraph, the complexity can be decreased by using the Breadth-First-Search (BFS) down to O(|V|·|E|).
  This results from the |V| loop invocations of the BFS algorithm which determines the shortest path from one node to all the others in O(|E|) because the ESG is connected and |E|>|V|+1. In worst case when the ESG is fully connected then |E| equals |V|^2 so that the complexity is the same as Floyds algorithm.
- The optimizing problem, which is solved in accordance with [Kn93] by the Hungarian method, with the complexity O(|V|^3).
- Computation of an Euler tour with the complexity of O(|V|+|E|) [We96].

To sum up, the MS^{2}CES can be solved in O(|V|^3) time. Instead of this "perfect" solution (determined by the Hungarian method), sub-optimal solutions can be delivered by balancing the nodes via searching shortest paths using also the BFS; the overall complexity is then reduced down to O(|V|·|E|).

Example 1 lists a minimal set of the legal walks (i.e., CESs) for the example given in Figure 4 to cover all event pairs. Note that no entire walk exists for this example. Therefore, an ideal walk cannot be constructed.

Example 1. Euler tour=*[ABACBDCBC][* $\rightarrow$ MS^{2}CES=*ABACBDCBC*

4.2 Minimal Spanning Set for the Coverage of Faulty Event Sequences

The union of the sets of FCESs of the minimal total length to cover the FESs of a required length is called *Minimal Spanning Set of Faulty Complete Event Sequences (MS²FCES)*. In comparison to the interpretation of the CESs as legal walks, *illegal walks* are realized by FCESs that never reach the exit. An illegal walk is *minimal* if its starter cannot be shortened. Assuming that an ESG has n nodes and d arcs as EPs to generate the CESs, then at most $u := n^2 - d$ FCESs of minimal length, i.e., of length 2, are available. Those FCESs emerge when the node(s) after entry is (are) followed immediately by a faulty input. The number of FCESs is precisely determined by the number of FEPs, which are of constant length 2; thus, they cannot be shortened. It remains to be noticed that only the starters of the remaining FEPs can be minimized, e.g., using the algorithm given in [ED59]. A further algorithm to generate FESs of length > 2 is not necessary because such faulty sequences will be constructed through the concatenation of the appropriate starters with the FEPs.

5. Exploiting the Structural Features

The approach has been applied to the testing and analysis of the GUIs of different kind of systems, leading to a considerable amount of practical experience. A great deal of test effort could be saved considering the structural features of the SUT. Thus, there is further potential for the reduction of the cost of the test process.

5.1 A Practical Example

Figure 5 depicts a small part of the GUI of an MS WordPad-like word processing system. The optional events are abbreviated in the Figure 6 with capital letters. The described components are used to traverse through the entries of the menu and sub-menus, creating many combinations and accordingly, many applications. The GUI represented in Figure 5 is transferred to an ESG (Figure 6). Figure 5 is easy to understand, but an informal and imprecise presentation of the GUI, while Figure 6 is a formal presentation that neglects some aspects, e.g. the hierarchy, while still being precise. The conversion of Figure 5 into Figure 6 is the most abstract step of the approach that must be done manually. Example 2 lists the FCESs to cover the FEPs of the ESGs Main/ Open given in Figure 6.

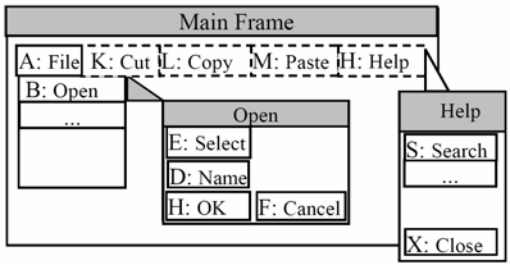

Figure 5: Top-level GUI of WordPad

Example 2. *AD*, *AE*, *AF*, *AH*, *ABA*, *ABB*, *ABH*, *ABDA*, *ABDB*, *ABEA*, *ABEB*, *ABFB*, *ABFF*, *ABFE*, *ABFD*, *ABFH*, *ABEHA*, *ABEHB*, *ABEHD*, *ABDHE*, *ABDHF*, *ABDHH*

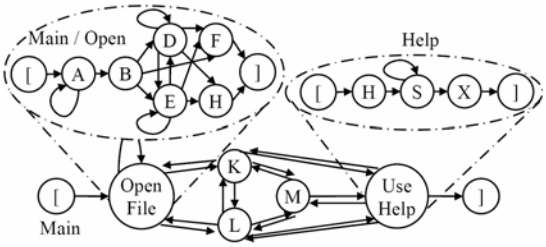

Figure 6: ESG of the GUI represented in Figure 5.

5.2 Modal and Modeless Windows

Analysis of the structure of the GUIs, e.g., the example GUI in Figure 5, delivers the following features:

- Windows of commercial systems are nowadays mostly hierarchically structured, i.e., the root window invokes children windows that can invoke further (grand) children.
- Some children windows can exist simultaneously with their siblings and parents; they will be called *modeless* (or *non-modal*) windows. Other children, however, must "die", i.e., close, in order to resume their parents (*modal* windows).

For the main frame of the WordPad, the child window `Help` is a modeless window; the other child window, `Open`, is a modal one. Figure 7 represents these windows as a "family tree". In this tree, a unidirectional edge indicates a modal parent-child relationship. A bidirectional edge indicates a modeless one. Because modal windows must be closed before any other window can be invoked, it is not necessary to consider the FESs of the parent and children. This is true only for the FCESs and MS^{2}FCESs as test inputs considering the structure information might impact the structure of the ESG, but not the number of the CESs and MS^{2}CESs as test inputs. Thus, similar to the strong-connectedness and symmetrical features [SS97], the modality feature is extremely important for testing since it avoids unnecessary test efforts. Figure 8 represents the modified ESG of the WordPad. The modification, which separates the events *A* and *B* from Open, takes the modality into account that avoids unnecessary FEPs. Example 3 lists the MS^{2}FCESs to cover FEPs of the sub-graph Open given in Figure 8.

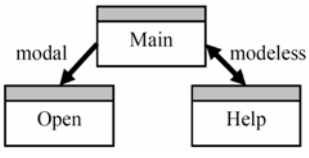

Figure 7: Modal windows vs. modeless windows

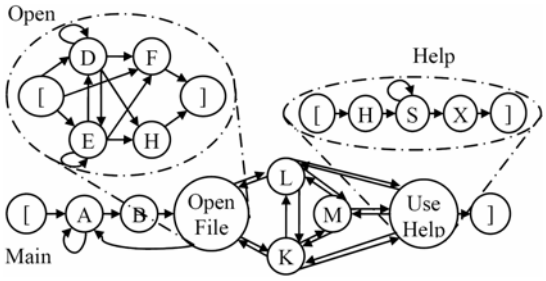

Figure 8: Modified ESG of the GUI in Figure 7

Example 3. *EFD*, *EFE*, *DFF*, *DFH*, *EHF*, *EHD*, *DHF*, *DHH*

Already this example, i.e., the comparison of Example 1 (22 FEPs) with Example 2 (8 FEPs), demonstrates the efficiency increase through the exploitation of the structural features of the SUT.

6. Extension of the Approach to Statecharts

The approach described in section 3 and 4 can be applied to „high-level“ visualization tools, e.g., statecharts, which, extend conventional state-transition diagrams by aspects of hierarchy, concurrency and communication [HAR87]. A statechart diagram describes sequences of states and transitions through which the system can proceed during its operation. The syntax and semantics of a modified class of Harel's statecharts are de-facto-standardized in the *OMG Unified Modeling Language Specifications* [OMG03].

For modeling the faulty system behavior, the approach introduced in this paper complements statechart diagram by an *error state*. In any non-error, i.e., correct, state any other event than the legal transition transfers to the error state and forms a *faulty transition*. As an example, in Fig. 9, from state s_1 only the event *c* triggers a (legal) transition. Therefore, events *a* or *b* causes faulty transitions from state s_1 to the error state. The test criteria introduced in section 3.3, i.e., coverage of legal event pairs (EP) and faulty event pairs (FEP), are to be reconsidered, because a single event pair can represent more than one transition pair (TP) [OA99]. This leads to the sequentialization of the TPs. Accordingly, faulty transition pairs (FTP) can be introduced (in analogy to the FEPs). This leads to following test criteria.

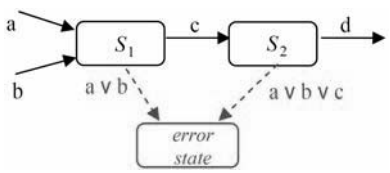

Figure 9: Error state and faulty transition

Transition Pair Coverage: *For any state, incoming and outgoing transitions are to take place sequentially.*

Faulty Transition Pair Coverage: *For any state, incoming and outgoing faulty transitions are to take place sequentially.*

7. Validation

The approach introduced in this paper has been applied to several case studies. Three of these case studies are briefly sketched in this section. As no system specification was available, user manuals were studied to produce ESGs or/and statecharts. This modeling effort avoided a "trial and error" way of testing and enabled detection of many, partly intricate faults. Note that the SUTs of these case studies are commercial systems that are widely used over many years. Accordingly, they have been frequently updated, matured and well-established.

For the first case study, a significant, non-trivial function of the personal music management system RealJukebox (RJB), Version 2, of RealNetworks, has been selected. This function enables the user to load a CD, select a track, and play it. The user can then change the mode, replay the track, or remove the CD, load another one, etc. For a comprehensive black-box testing, several strategies have been developed with varying characteristics of the test inputs, i.e.,

- the length and number of the test sequences,
- the type of the test sequences, i.e., CES- and FCESs-based and
- modeling the system with ESGs vs. statecharts.

12 ESGs were constructed, leading to total of 255 nodes. The study delivered following findings:

- The most faults have been detected by the test cases covering EPs and FEPs, i.e., ESs of the length 2. Coverage of ESs of the length>4 did not lead to detecting substantial more faults. Some of the faults have repeatedly been detected.
- The test cases covering ESs of the length 4 were more effective in revealing dynamic, intricate faults than the test cases of the lengths 2 and 3. They were, however, considerably more expensive in terms of costs per detected fault.
- The CES-based test cases as well as the FCES-based cases were effective in detecting faults, without revealing a tendency.

For the construction of test cases our group developed the tool "GenPath" [BHM04]; the structural information has been obtained with the capture-playback facility of a commercially available tool. A single tester, who acted also as oracle, carried out 1166 tests semi-automatically over a period of 2 days, working, on average, 8 hours per day, thus spending a total of 78560 seconds. These figures result in approximately 67 seconds per test. A total of 32 faults were detected. The results of the research for minimizing the spanning set of the test cases (MS^2CES and MS^2FCES), as described in Section 4, has been applied to the testing of the selected function. Table 1 summarizes that the minimization could save about 60 % of the test costs, while the exploitation of the structural information of the SUT could save up to almost 20 %.

Table 1: Reducing the number of test cases achieving the same number of detected faults

Length	#CES	#MS^2CES	Cost Reduction ES
2	40	15	62.5 %
Length	#MS^2FCES without structural information	#MS^2FCES with structural information	Cost Reduction MS^2FCES
2	75	58	22.7 %

Two additional case studies were performed to compare the fault detection capability of ESG modeling vs. statechart modeling. For the next case study (#1) the same tester consecutively constructed the ESGs and statecharts; in another case study (#2) different testers carried out the modeling job by separately constructing the ESGs and statecharts. Table 2 compares the both strategies.

Table 2: Comparison of the fault detection capability of statecharts vs. ESG

ESGs and statecharts constructed	Faults detected only by ESG	Faults detected both by ESG and statecharts	Faults detected only by statecharts
#1: consecutively	2	32	-
#2: separately	12	11	5

Expectedly, constructing the statecharts and ESG separately by different testers lead to a smaller total number of faults detected by both models. Concerning the number of de-

tected faults, ESG modeling considerably dominates statechart modeling which can easily be explained: ESGs are simpler to be handled, and thus, the tester could work more efficiently, i.e., produce more and better detailed ESGs than statecharts, and accordingly, a better analysis and testing job could be performed. A more detailed discussion of the case studies and their findings is given in [BNB04].

8. Conclusion and Future Work

This paper has introduced an integrated, black-box approach to coverage testing of interactive systems, incorporating modeling of the system behavior with fault modeling and minimizing of the test sets for the coverage of these models. The framework is based on the concept of "event sequence graphs (ESG)". Event sequences (ES) represent the human-computer interactions. An ES is complete (CES) if it produces desirable, well-defined and safe system functionality. The notion of complete faulty event sequences mathematically complements this view. To increase its applicability potential, the approach is extended to UML statecharts which are nowadays de facto the industrial standard. The objective of testing is the construction of a set of CESs of minimal total length that covers all ESs of a required length. A similar optimization problem arises for the validation of the SUT under exceptional, undesirable situations which are modeled by faulty event sequences (FESs) and complete FESs (FCESs). The paper modified algorithms known from the literature and applied to these problems. Furthermore, it was shown how the structure of interactive systems can be exploited to reduce the test sets by eliminating infeasible and/or unnecessary test cases, leading to a considerably less complexity, which is a novelty introduced in this paper. Comparison of the fault detecting capability of the ESG with the fault detecting capability of the statecharts could not point out any significant tendency but validate the efficiency of the approach when applied to different modeling methods.

The goal for present work is to design defense actions, which form appropriately enforced sequences of events, in order to prevent faults that could potentially lead to failures. Further planned work concerns cost reduction through automatic test execution. Starting point is to integrate different self-developed tools and use them as an add-on to a commercially available test tool.

Literature

[ADL91] A. V. Aho, A. T. Dahbura, D. Lee, M. Ü. Uyar, "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours", *IEEE Trans. Commun.* 39, pp. 1604-1615, 1991

[AMO93] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, "Network Flows-Theory, Algorithms and Applications", Prentice Hall, 1993.

[Be01] F. Belli, "Finite-State Testing and Analysis of Graphical User Interfaces", Proc. 12th ISSRE, pp. 34-43, 2001

[BHM04] Ch. J. Budnik, A. Hollmann, R. Moge, „GenPath – A Tool to Generate Paths of different Lengths of an Event Sequence Graph", TR 2004/9, Univ. Paderborn, 2004

[Bi00] R.V. Binder, "*Testing Object-Oriented Systems*", Addison-Wesley, 2000

[BNB04] F. Belli, N. Nissanke, Ch. J. Budnik, "A Holistic, Event-Based Approach to Modeling, Analysis and Testing of System Vulnerabilities"; TR 2004/7, Univ. Paderborn, 2004

[BP94] G. V. Bochmann, A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing", *Softw. Eng. Notes, ACM SIGSOFT*, pp. 109-124, 1994

[Ch78] Tsun S. Chow, "Testing Software Designed Modeled by Finite-State Machines", *IEEE Trans. Softw. Eng.* 4, pp. 178-187, 1978

[DLS78] R.A. DeMillo, R.J. Lipton, F.G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer", *Computer* 11/4, pp. 34-41, 1978

[DMM01] M.E. Delamaro, J.C. Maldonado, A. Mathur, "Interface Mutation: An Approach for Integration Testing", *IEEE Trans. on Softw. Eng.* 27/3, pp. 228-247, 2001

[ED59] Edsger. W. Dijkstra, "A note on two problems in connexion with graphs.", Journal of Numerische Mathematik, Vol. 1, pp. 269-271, 1959

[HAR87] D. Harel, "Statecharts: A Visual Formalism For Complex Systems", Science of Computer Programming 8, S. 231-274, 1987

[HKC00] H. S. Hong, Y. G. Kim, S. D. Cha, D. H. Bae, H. Ural: "*A test sequence selection method for statecharts*"; Software Testing, Verification and Reliability 2000: 10; John Wiley & Sons; pp. 203-227; 2000

[JNC03] J. Jorge, N.J. Nunes, J.F. Cunha (Eds.), "Interactive Systems – Design, Specification, and Verification", LNCS 2844, Springer-Verlag, 2003

[Kn93] D.E. Knuth, "The Stanford GraphBase", Addison-Wesley, 1993

[MB03] M. Marré, A. Bertolino, "Using Spanning Sets for Coverage Testing", *IEEE Trans. on Softw. Eng.* 29/11, pp. 974-984, 2003

[MPS00] A. M. Memon, M. E. Pollack and M. L. Soffa, "Automated Test Oracles for GUIs", *SIGSOFT 2000*, pp. 30-39, 2000

[OA99] J. Offutt, A. Abdurazik: "*Generating Tests from UML Specifications*"; UML'99 - The Unified Modeling Language; Springer; pp. 416-429; 1999

[NT81] S. Naito, M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", *Proc. FTCS*, pp. 238-243, 1981

[OMG03] OMG Unified Modelling Language Specification, Version 1.5, formal/03-03-01, 2003

[OSA03] J. Offutt, L. Shaoying, A. Abdurazik, and Paul Ammann, "Generating Test Data From State-Based Specifications", *The Journal of Software Testing, Verification and Reliability, 13(1):25-53, Medgeh 2003*.

[Pa69] D.L. Parnas, "On the Use of Transition Diagrams in the Design of User Interface for an Interactive Computer System", Proc. 24th ACM Nat'l. Conf., pp. 379-385, 1969

[Sa89] B. Sarikaya, "Conformance Testing: Architectures and Test Sequences", *Computer Networks and ISDN Systems* 17, North-Holland, pp. 111-126, 1989

[SS97] R. K. Shehady and D. P. Siewiorek, "A Method to Automate User Interface Testing Using Finite State Machines", in *Proc. Int. Symp. Fault-Tolerant Computing FTCS-27*, pp. 80-88, 1997

[Th03] H. Thimbleby "The Directed Chinese Postman Problem", School of Computing Science, Middlesex University, London, 2003

[TL02] K. Tai, Y. Lei, "A Test Generation Strategy for Pairwise Testing", *IEEE Trans. On Softw. Eng.* 28/1, pp. 109-111, 2002

[We96] D.B. West, "Introduction to Graph Theory", Prentice Hall, 1996

[WA00] L. White and H. Almezen, "Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences", in Proc ISSRE, IEEE Comp. Press, pp. 110-119, 2000