Property–Oriented Testing: A Strategy for Exploring Dangerous Scenarios

Olfa Abdellatif-Kaddour, Pascale Thévenod-Fosse and Hélène Waeselynck

LAAS-CNRS, 7 Avenue du Colonel Roche
31077 Toulouse Cedex 4 – France
Phone: +(33) 5 61 33 62 00
Fax: +(33) 5 61 33 64 11
{kaddour, thevenod, waeselyn}@laas.fr

ABSTRACT

Property-oriented testing uses the specification of a property to drive the testing process. The aim is to validate a program with respect to a target property, that is, to exercise the program and observe whether the property is violated or not. The paper defines a test strategy for safety properties in cyclic control systems. It consists of the stepwise construction of test scenarios. Each step explores possible continuations of the dangerous scenarios found at the previous step, using black-box sampling techniques. The feasibility of the strategy is illustrated on a steam boiler case study. The target property is the "non explosion" of the boiler in presence of faults in the physical devices. The experimental results are promising since four different explosive scenarios have been identified.

Keywords

Software testing, high-level safety property, test data generation, case study.

1. INTRODUCTION

This paper investigates a strategy for property-oriented testing of cyclic real-time control systems. Property-oriented testing uses the specification of a property to drive the testing process [4]. The aim is to validate a program with respect to a target property, that is, to exercise the program and observe whether the property is violated or not. The type of property we are interested in is any high level requirement related to the most critical failure modes of the control system, as identified in a preliminary risk analysis. Such a high-level property is required from the whole system, rather than from some delimited part of it. Formal verification would need a detailed model reproducing the interactions between the control program and its controlled environment, accounting for the physical devices, the physical laws governing the controlled process, as well as the possible occurrences of physical faults in the devices. Such an exact analysis of system behavior is generally unfeasible, and testing may be seen as a pragmatic alternative, provided that the testing environment is as close as possible to the operational environment (i.e., the control program is run on the target hardware and connected to a simulator of the physical environment, with facilities for fault injection). However, there is the issue of test data selection. To address this issue, we propose a strategy consisting of the stepwise construction of test scenarios. Each step explores possible continuations of the "dangerous" scenarios identified at the previous step, using a black-box sampling technique. The feasibility of the strategy is illustrated on the steam boiler case study [1], the target property being the "non explosion" of the boiler in presence of faults in the physical devices.

Section 2 introduces our testing problem and presents the proposed strategy. Section 3 describes the steam boiler case study that we used to illustrate the feasibility of our test strategy. Sections 4 and 5 present empirical results related to successive steps of the strategy, using random sampling as a simple example of black-box technique. Conclusions and future directions are outlined in Section 6.

SAC 2003, Melbourne, Florida, USA.

2. PRINCIPLE OF THE TEST STRATEGY

2.1 System Behavior wrt Property Violation

The ultimate objective is to find reachable system states where the target property is violated (if any). Due to the sequential (with memory) behavior exhibited by control systems, it is expected that property violation will occur after execution of particular trajectories in the input domain of the system, rather than just at specific points of the input domain: the system will progressively evolve towards property violation. During this progressive evolution, the system reaches intermediate states that correspond to *dangerous situations*, that is, states from which property violation may eventually occur if the system is not robust enough to recover. For example, a general notion of dangerousness for control systems may be related to the fact that the state of the controlled environment, as perceived by the control program, differs from the actual state. In practice, the dangerous situations specific to a given system domain can be elaborated based on the safety analyses that accompany system development. Their pertinence is a prerequisite to the effectiveness of the test strategy we propose.

Hence, the strategy should be conducted in close connection with domain experts.

There is no way to know *a priori* how long it could take to reach property violation. Using the dangerous situations as intermediate test objective gives us the opportunity to progressively explore solution spaces of larger test sequences. This is the rationale for the stepwise construction of scenarios: each step explores possible continuations of the dangerous situations reached at the previous step, trying to get close to a violation.

2.2 Stepwise Construction of Test Scenarios

The stepwise construction of test scenarios proceeds as follows. The *first step* concentrates on test sequences s of predefined maximum length L_1. The test objective includes both property violation during the execution of s, and the reaching of a dangerous situation at the end of the execution of s (without property violation). If property violation is reached, it must be decided whether further testing has to be done before fixing the identified problem, or not (i.e., the testing process is stopped, and it will be restarted on the next release of the control program). If the testing process may go on, the test sequences that lead to dangerous situations are analyzed. Some of them may be eliminated based on their very low probability of occurrence, or based on strong evidence that they cannot lead to property violation. The remaining ones require further tests to determine whether the property may be violated, or whether the system is robust enough to recover from the dangerous situation.

From each dangerous scenario s retained at the first step, the *second step* explores possible continuations, in the form of sequences s' of predefined maximum length L_2. During the experiments, each s' is prefixed by a sequence s that forces the system into the dangerous situation under investigation, yielding a complete sequence $s.s'$. As in step 1, sequences reaching the test objective are analyzed to determine whether or not some of them require further tests.

If a test sequence $s.s'$ is retained, the *third step* is performed using $s.s'$ as the new prefix of subsequent sequences s'' of maximum length L_3. And so on, until every dangerous situation retained at step i, leads — after further tests at step i+1 — either to a property violation, or to a safe situation.

It is worth noting that the first step is the corner stone of the strategy since it initiates the search process: if no sequences of maximum length L_1 can reach the test objective, then larger test sequences will not be investigated and no property violations will ever be observed. Thus, the underlying hypothesis is that L_1 is chosen large enough to initiate a dangerous situation. We think that this assumption is less strong than setting a priori the maximum length required for property violation, which involves both the time to reach a dangerous state and the time to progressively evolve from this state to property violation. In practice, the value of L_1 has to be tuned (preferably with the help of experts of the system under test) according to problem-specific features such as the reaction time of the system and the definition of the notion of dangerousness. It must be a tradeoff between potentially revealing dangerous scenarios versus exponentially increasing the search space. As regards revealing dangerous scenarios, the subsequent lengths L_i (i>1) are not so critical since they may only have an effect on the number of steps that will be required to eventually observe either safe situations or property violation.

2.3 Selection of Test Sequences at Step i

At step i, the process of test sequence selection should try to favor the choice of those scenarios that will be the most "stressing" with respect to the test objective. The definition of such a guiding process requires consideration of two main issues: (1) the intractability of a detailed analysis of the system behavior, and (2) the large input space to be sampled, even for a bounded length L_i. The first problem compels us to use a black-box approach to select the test sequences. The second one may be alleviated (to a certain extent) by decomposing the input space into smaller subspaces to be separately explored during testing.

Based on the system specification, the input space may be modeled in terms of high-level functional activities, and possibly fault hypotheses (physical faults in the devices controlled by the program). This information may be used to define subspaces to be explicitly considered during the test. As an example, the decomposition into subspaces can take the system operating modes into account (e.g., normal mode, degraded modes that correspond to partial operational modes after failure of some physical devices, and transitions between different modes). Finer analysis of system behavior may be considered, as long as it remains tractable.

Within each subspace, a black-box technique is used to select test sequences. A well-known one, which is easy to implement, is random sampling [3]. In this paper, we use this technique to study the feasibility of the proposed test strategy on the steam boiler case study. In the general case, more sophisticated techniques can be used. For example, crafted probabilistic profiles [10] or modern heuristic search techniques [7] should be potential solutions to the problem of test sequence selection.

3. THE STEAM BOILER CASE STUDY

We have carried out a case study based on a steam boiler problem for which high-level requirements, control program code, and a software simulator of physical devices, are publicly available [11]. A number of formal approaches have been used to model and verify this steam boiler problem [1]. Few of them have considered the whole problem, since most have focused on the control program. They built formal models based on its informal specification, paying little attention to the (implicit) assumptions this specification puts on the controlled environment. In contrast to previous work, our aim is to verify the absence of critical failure modes when the control program is connected to the physical environment. Most of the complexity of the control program specification comes from the definition of fault tolerance mechanisms. Hence, we put emphasis on a specific category of environmental inputs: the faults to be tolerated. The stepwise strategy is applied to the search for scenarios combining faults with the functional activity, and yielding a violation of safety requirements.

3.1 Steam Boiler Description

The physical environment comprises the boiler, 4 pumps to provide the boiler with water, 4 pump controllers to sense the state of the pumps (open/closed), a device to measure the quantity of water in the boiler, and a device to measure the quantity of steam, which comes out of the boiler. The function of the control program is to maintain the water level in the steam boiler between two predefined thresholds (denoted N_1 and N_2) by controlling pumps. Besides this main function, the control program must also maintain safety by shutting the steam boiler down if its water level is either too low ($<M_1<N_1$) or too high ($>M_2>N_2$) for more than five seconds, otherwise, the steam boiler can be seriously damaged (explosion). Finally, the control program must be able to withstand some physical failures by continuing to operate while part of the equipment is malfunctioning, until it is repaired. The corresponding degraded operational modes are defined in the requirements. If the control program cannot assure its function, it must shut the system down. Our aim is to study the capacity of the control program to maintain safety in the presence of faults affecting the physical devices. So, the target safety property is the "non explosion" of the steam boiler, i.e., "The water level must not be $< M_1$ or $> M_2$ for more than 5 s", where M_1 and M_2 are the water level safety limits. This is a high-level property depending on proper interaction between the control program and the physical environment. To test this property, we use a simulator that mimics the behavior of the physical devices, including some physical faults.

3.2 Test Input Domain and Dangerous Situations

As shown in Figure 1, our input domain includes the physical faults that can be injected via the simulator, and the cycle numbers during which the physical faults are injected. Hence, a test sequence is defined as a sequence of faults affecting physical devices (e.g., faults affecting pumps or pump controllers) with the cycle numbers during which the faults are to be injected. In this case study, we can inject up to 10 faults affecting the 10 physical devices: 4 pumps (actuators), 4 pump controllers (sensors), the water level sensor and the steam sensor. When a pump fault occurs, its state remains unchanged ("do nothing") until it is repaired: if the pump is open (respectively closed), it remains open (respectively closed) until it is repaired. When a sensor fault occurs, the sensor keeps sending the value sensed immediately prior to the fault occurring. Each of the ten possible faults is denoted by a unique identifier, yielding *Fault A .. Fault J*.

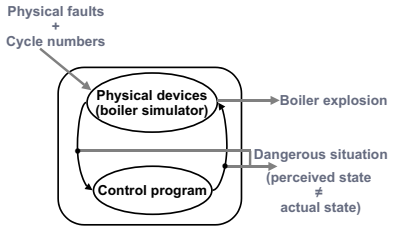


Figure 1. Overview of the system under test

According to Section 2, the test objective is to reach either a boiler explosion or a dangerous situation. Since no safety analysis is available for this case study, we adopt a general notion of dangerousness for control systems: a dangerous situation is reached when the state of the environment, as perceived by the control program, differs from the actual state. This difference may be expressed in three sub-objectives:

- *Fewer failure detections*, which is fulfilled when the control program does not identify the presence of at least one of the injected faults.
- *More failure detections*, which is fulfilled when the control program wrongly identifies the presence of at least one fault, which indeed was not injected.
- *Bad estimation of the water level*, which is reached when the water level in the simulator is not included within the plausibility limits (*qc1*, *qc2*)[1] that are calculated by the control program.

Let us note that the three sub-objectives are not exclusive: a test sequence can fulfill several sub-objectives at the same time (e.g., both one injected fault not identified and one not-injected fault wrongly identified).

A dangerous situation is identified from the messages exchanged between the simulator and the control program. An explosion is directly observed from the simulator (see Figure 1).

4. FIRST STEP OF THE TEST STRATEGY

The first step of our test strategy searches for test sequences of predefined maximum length L_1. This length is tuned according to the reaction time of the boiler and the definition of the dangerous situations: we make the hypothesis that a maximal length of 10 injection cycles is sufficient to initiate a dangerous situation[2]. The corresponding input space is decomposed into smaller subspaces to be separately explored (Section 4.1). The random sampling algorithm used to explore each subspace is given in Section 4.2. Then, experimental results are presented and analyzed (Sections 4.3 and 4.4).

4.1 Decomposition of the Search Space

To decompose the input space into smaller subspaces, two criteria are taken into account: (1) the operating mode of the boiler, which can be *transient* when the boiler heats or *permanent* when it produces steam, and (2) the number of injection cycles, which characterizes the temporal dispersion of faults. In the absence of faults, manual calculation indicates that the transition from transient to permanent mode should occur at cycle 3. This result is confirmed by preliminary test experiments coupling the boiler simulator with the control program. The boiler mode thus determines the first potential injection

[1] qc1 (respectively qc2) denotes the minimal (respectively maximal) water level limit that is calculated by the control program. The observation of qc1 and qc2 requires a slight instrumentation of the control program.

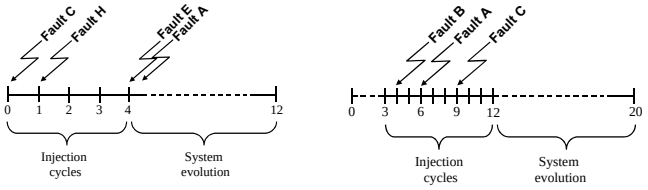
[2] As will be seen in Section 4.3, setting $L_1 = 10$ is indeed sufficient to initiate the search process.

cycle (0 and 3 for transient and permanent modes, respectively). In order to study the influence of the temporal dispersion of faults, we take three sizes of injection cycles: a small number, a medium number and a large number (respectively 3, 5 and L_I=10 cycles). By combining the mode and number of injection cycles, we get six classes of test sequences to be explored at step 1 (notation S1.k means class *k* at *Step 1*):

- Class S1.1: Sequences injecting during cycles 0..2 (transient, small);
- Class S1.2: Sequences injecting during cycles 0..4 (transient, medium);
- Class S1.3: Sequences injecting during cycles 0..9 (transient, large);
- Class S1.4: Sequences injecting during cycles 3..5 (permanent, small);
- Class S1.5: Sequences injecting during cycles 3..7 (permanent, medium);
- Class S1.6: Sequences injecting during cycles 3..12 (permanent, large).

Whatever the class, we add a fixed number of cycles (8 cycles) after the allowable injection cycles, to let the system evolve. After the system evolution time, we determine whether at least one of the sub-objectives is fulfilled or not.

Figure 2 shows two example test sequences, one of Class S1.2 (Figure 2.a) where faults may be injected during cycles 0..4, and one of Class S1.6 (Figure 2.b) where faults may be injected during cycles 3..12. For instance, in the first test sequence, *Fault C* is injected during cycle 0, *Fault H* during cycle 1 and *Faults E* and *A* during cycle 4. After the system evolution time (cycles 5 to 12), the test outcome is determined by accounting for the observations that have been collected during the 13 cycles.



(2.a) Class S1.2 **(2.b) Class S1.6**

Figure 2. Example test sequences

4.2 Random Sampling Algorithm

As a first investigation, we used a simple technique to explore each subspace, namely random sampling. Figure 3 shows the algorithm. It is worth noting that test sequences are generated in accordance with the considered class of sequences: the allowable injection cycles depend on the target class. For each class, we generate a fixed number of sequences. At the end of the executions, the collected observations are analyzed to identify the test sequences that led to either a boiler explosion or a dangerous situation.

```
Sequence_number = 0;
REPEAT
    Choose randomly the total number of faults Nb_Inject_Fault
    between 1 and 10
    FOR i = 1 to Nb_Inject_Fault
        Choose randomly a fault without replacement
        Choose randomly its injection cycle (according to
        the considered class)
    ENDFOR
    Apply the sequence to simulator + control program;
    Increment Sequence_number;
UNTIL (Sequence_number = MAX_SEQUENCE)
```

Figure 3. Random sampling algorithm

4.3 Experimental Results

First experiments with *MAX_SEQUENCE = 100* allowed us to identify three test scenarios leading to steam boiler explosion and two test scenarios leading to a dangerous situation. The three scenarios that lead to a boiler explosion are:

1. Faults affecting several pumps or pump controllers followed by the fault affecting the water level sensor, under the condition that the control program detects at least two failures of pump or pump controller before the sensor fails;
2. Fault affecting the steam sensor during cycles 0 or 1;
3. Faults affecting the water level sensor and the steam sensor, during cycles 0 or 1.

The two scenarios that lead to a dangerous situation are:

a. A pump fault occurs, while the associated pump controller is working correctly;
b. A steam sensor fault occurs, just when the mode turns to permanent (during cycle 3).

Concrete examples of such test scenarios are given in Table 1. For instance, the first test sequence is composed of the fault affecting pump controller 2 (“stuck-at-previous-value”, denoted *Pump_Ctr2-Fault*) injected during cycle 4 (put in brackets), and the fault affecting pump 3 (“do nothing”) injected during cycle 6, and the fault affecting the water level sensor (“stuck-at-previous-value”) injected during cycle 10. When we apply this test sequence to the simulator + control program, the control program detects the pump controller and pump failures before cycle 10, and consequently enlarges the plausibility interval [*qc1*, *qc2*] of water level for safety reasons. After injection of the third fault, the sensor’s value remains within [*qc1*, *qc2*]. The program cannot detect this failure. It continues to trust the sensor, i.e., it believes that the water level is stable until the boiler explodes.

Table 1. Examples of test sequences leading to boiler explosion or a dangerous situation

TYPE OF SCENARIO	CONCRETE EXAMPLES Fault (injection cycle)	CONTROL PROGRAM BEHAVIOR	
1	Pump_Ctr2-Fault (4), Pump3-Fault (6), Water-Level-Fault (10)	Detection of the pump and pump controller failures, and non-detection of the water level failure	Explosion
2	Steam-Fault (1)	Wrong detection of the water level failure, and non-detection of the steam failure	
3	Water-Level-Fault (0), Steam-Fault (0)	No failure detection	
a	Pump1-Fault (2)	Detection of the pump failure, and wrong detection of its pump controller failure	Dangerous situation
b	Steam-Fault (3)	Non-detection of the steam sensor failure	

Random sampling finds test sequences that fulfill the test objective very easily. For example, Table 2 shows the results obtained for Classes S1.2 and S1.6. In this table, for each type of scenario, we give the number of sequences found over a sample of size 100. For instance, scenario 1 is not found in Class S1.2, while it is found 25 times in Class S1.6. Let us note that Class S1.2 cannot produce scenario 1, because the allowable temporal dispersion of faults is not sufficient. Class S1.6 does not produce scenarios 2 and 3 since it starts fault injection from cycle 3.

Table 2. Experimental results for test sequences of Class S1.2 and Class S1.6

TYPE OF SCENARIO	NUMBER OF SEQUENCES FOUND (Sample of size 100)	
	CLASS S1.2	CLASS S1.6
1	0	25
2	24	0
3	5	0
a	13	18
b	3	1

As a general comment, it is surprising to fulfill the test objective so easily, in terms of both the number of different scenarios found, and the density of test sequences instantiating these scenarios within the classes. Indeed, random sampling is sufficient to reveal a number of serious problems. Scenarios *1* and *a* were already known in the literature. To the best of our knowledge, the others were not, in spite of the fact that the steam boiler has been extensively studied using formal methods. This is because the identified cases are not caused by the non-conformance of the control program to its requirements. The problems originate from the very definition of the degraded modes in the requirements (specification fault). This confirms the usefulness of taking the physical environment into account in the testing process.

4.4 Decisions Taken at the End of Step 1

According to the strategy, the outcomes of step 1 have to be analyzed to decide how to continue the testing process.

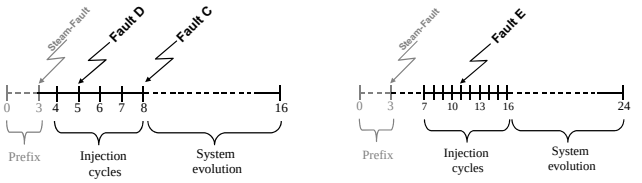
Given the three explosive scenarios found so far, it would be expected to fix the safety problems and restart step 1 of the test strategy. Since the problems exist in the requirements, the impact of the fix would be very important. In the framework of our experimental study, we did not correct the requirements and decided to continue experiments with the same version of the control program.

Two dangerous scenarios (scenarios *a* and *b*) are candidates for further testing at step 2. Based on previous work [2] and our analysis, scenario *a* is not retained because it cannot lead to explosion: although the control program does not clearly distinguish a pump fault from a pump controller fault, the water level interval is safely calculated. Let us consider scenario *b*, in which the control program is not able to detect a steam sensor failure at cycle 3 (transition from transient to permanent mode). This introduces a small error in the calculation of the water level interval. It is not easy to determine whether or not the system can evolve towards explosion upon occurrence of additional faults. Hence, scenario *b* is retained for step 2.

5. SECOND STEP OF THE TEST STRATEGY

5.1 Selection of Test Sequences at Step 2

The second step of the test strategy explores possible continuations of scenario *b*. Thus the test sequences are prefixed by scenario b (*Steam-Fault* at cycle 3), and new faults may be injected only from cycle 4, during permanent mode. To decompose the new input space into smaller subspaces to be separately explored, we now account for (i) the time spent in the dangerous situation before additional faults may occur — 0 or 3 cycles, and (ii) the number of injection cycles, which characterizes the temporal dispersion of faults — small, medium or large (3, 5, or 10 cycles). By combining these two criteria, we get six classes of sequences, labeled S2.1, …, S2.6. For example, Class S2.2 sequences inject additional faults during cycles 4..8, and Class S2.6 sequences during cycles 7..16 (Figure 4). As in step 1, the experiments involve 100 sequences randomly selected for each class.



(4.a) Class S2.2 **(4.b) Class S2.6**

Figure 4. Example test sequences

5.2 Experimental Results

Random sampling allowed us to identify a new explosion scenario (scenario 4) related to the dangerous situation. A concrete example of this scenario is:

Steam_Fault (3), Pump2-Fault (4), Water-Level-Fault (8)

Scenario 4 involves one pump or pump controller fault followed by water level fault. It corresponds to the triggering of an extremal/special case. When the above test sequence is applied without injection of *Steam-Fault* at cycle 3, the boiler safely shuts down. This is because the value delivered by the faulty sensor turns out to fall outside the plausibility interval [*qc1, qc2*] calculated by the control program. As already mentioned (Section 4.4), the injection of *Steam-Fault* at cycle 3 introduces a small error in the calculation of the water level limits; in fact, it causes a slight overestimation $qc2+\varepsilon$ of the upper plausibility bound. Now, the scenario is such that when the fault affecting the water level sensor occurs, the erroneous sensor value is greater than *qc2*, but still lower than $qc2+\varepsilon$. The water level sensor failure is not detected, and the system inexorably evolves toward explosion, because the program wrongly believes that the water level is stable.

Property violation due to scenario 4 is triggered by test sequences of all classes except for Class S2.1 since scenario 4 does not belong to the subspace explored for this class. For Classes S2.2 to S2.6, the numbers of sequences finding scenario 4 are respectively: 3, 6, 3, 11, and 12. To the best of our knowledge, the scenario was never reported in the literature.

No other dangerous scenarios are identified by the experiments. Hence, the test strategy applied to the steam boiler case study involves only two steps.

5.3 Effectiveness of the Stepwise Construction of Scenarios

It is worth noting that sequences of Classes S1.3 and S1.6 may produce scenario 4. Thus, it could have been identified in step 1, but it was not. To investigate whether this was due to the size of the sample drawn from each class, we have generated and executed 1000 sequences (instead of 100) of Classes S1.3 and S1.6. None of them identifies scenario 4.

Thus, the stepwise construction of test sequences, based on the notion of dangerous situation, allowed us to drastically increase the observed frequency of the scenario in step 2. Indeed, by identifying the prefix that puts the system in a dangerous state (outcome of step 1), we managed to focus the search on pertinent subspaces.

6. CONCLUSION AND FUTURE WORK

The results of the steam boiler case study tend to confirm the feasibility and usefulness of the stepwise construction of test scenarios. This principle allows a progressive exploration of trajectories in the input domain up to property violation, as exemplified by the new explosive scenario found at step 2. It is based on the assumption that the system will gradually evolve towards critical failure, traversing states of dangerous situations. This should be a reasonable assumption for cyclic control system characterized by (1) a sequential (with memory) behavior, and (2) a cycle duration that is small compared to the inertia of the physical environment.

At each step, the selection of scenarios proceeds according to empirical techniques that do not require a detailed model of the target control system, so as to remain tractable. For the steam boiler case study, it turns out that the simplest empirical technique, namely random sampling, is very efficient. But such a blind technique is not expected to be sufficient in most cases, when very few input sequences from the sampled subspace may reach the test objective. An idea might be to further exploit the key notion of dangerousness for structuring the subspaces and guiding their sampling: e.g., define an evaluation function measuring "distance" from the dangerous situations, and favor the generation of new input sequences in the "neighborhood" of the closest sequences found so far. This motivates the direction taken by our on-going work, studying the power of modern heuristic search techniques [7] as potential solutions to the problem.

Heuristic search techniques have already been used to automate the generation of test data for a number of testing problems, including instruction and branch testing [6], conformance testing [9], and robustness testing [8]. As regards our stepwise strategy, first experimental results (not presented in this paper) have been obtained by applying the *simulated annealing* heuristic [5] to the steam boiler case study. These results are encouraging, but we experienced that performance depends on the parameters chosen for the implementation of the generic heuristic, so that further investigation is required. In particular, we are currently comparing alternative evaluation functions (called *cost functions* in the framework of simulated annealing algorithms), in order to study sensitivity to reasonable variations in the search problem formulation.

7. ACKNOWLEDGEMENTS

This work is partially supported by the European Community (Project IST-1999-11585: DSoS – Dependable Systems of Systems).

8. REFERENCES

[1] J-R. Abrial, E. Börger and H. Langmaack (Eds), *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, Springer-Verlag, 1996.

[2] J.R. Cuéllar and I. Wildgruber, "The Real-Time Behavior of the Steam-Boiler", in [1], pp. 185-202.

[3] J.W. Duran and S.C. Ntafos, "An Evaluation of Random Testing", *IEEE Transactions on Software Engineering*, 10(4), pp. 438-444, July 1984.

[4] G. Fink and M. Bishop, "Property-Based Testing; A New Approach to Testing for Assurance", *ACM SIGSOFT on Software Engineering*, 22(4), pp. 74-80, July 1997.

[5] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, 220(4598), pp. 671-680, May 1983.

[6] C.C. Michael, G. McGraw and M.A. Schatz, "Generating Software Test Data by Evolution", *IEEE Transactions on Software Engineering*, 27(12), pp. 1085-110, December 2001.

[7] C.R. Reeves (Ed), *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill, 1995.

[8] A.C. Schultz, J.J. Grefenstette and K.A. De Jong, "Learning to Break Things: Adaptive Testing of Intelligent Controllers", in *Handbook on Evolutionary Computation*, Chapter G3.5, IOP Publishing Ltd. and Oxford Press, 1997.

[9] N. Tracey, J. Clark and K. Mander, "Automated Program Flaw Finding using Simulated Annealing", in *Proc. Int. Symp. on Software Testing and Analysis (ISSTA'98)*, Clearwater Beach, Florida, USA, ACM Press, pp. 73-81, 1998.

[10] J.A. Whittaker and M.G. Thomason, "A Markov Chain Model for Statistical Software Testing", *IEEE Transactions on Software Engineering*, 20(10), pp. 812-824, October1994.

[11] http://www.atelierb.societe.com/BOILER/UK/main.h

9. BIOGRAPHY

Olfa Abdellatif-Kaddour received the Engineer degree from the National School of Engineering of Tunis, Tunisia, in 1996. She is currently a PhD student in the Dependable Computing and Fault-Tolerance research group at the Laboratory for Analysis and Architecture of Systems (LAAS-CNRS), in Toulouse. Her current research interests focus on software testing and heuristic search technique. She is currently part of the DSoS IST project.

Pascale Thévenod-Fosse is currently a "Directeur de Recherche" at CNRS, the French National Organization for Scientific Research. From 1975 to 1986, she has been working on hardware testing at the Laboratory of Automation of Grenoble. In January 1987, she joined the Laboratory for Analysis and Architecture of Systems (LAAS-CNRS) in Toulouse to work within the research group on Dependable Computing and Fault Tolerance. Since 1987, her research has focused on software testing. She has written over 70 papers published in International Journals and Conferences, and is co-author of two books. She serves as reviewer in many Conferences and Journals, and as a program committee member in several International Conferences, such as IEEE FTCS, ACM ISSTA, IEEE DSN, EDCC (European Dependable Computing Conference). She recently served as general chair of the 4th issue of EDCC.
Pascale Thévenod-Fosse has managed research contracts and acted as a consultant for companies in France. She has also been involved in five European projects.
Dr. Thévenod-Fosse is a member of the IEEE Computer Society, and of the French SEE Society where she led the Working Group on Dependable Computing (400 memberships).

Hélène Waeselynck received the Engineer degree from the National Institute of Applied Sciences of Toulouse, in 1989, and the PhD in computer science from the National Polytechnic Institute of Toulouse, in January 1993. From 1993 to 1995, Dr. Waeselynck was working at the French National Institute for Transport and Safety Research (INRETS), where she was involved in the validation and certification of railway systems. In 1996, she joined the Dependable Computing and Fault-Tolerance research group at LAAS-CNRS. Her research interests concern software validation, including both testing and formal methods. Dr. Waeselynck is co-leader of a National Project on Robustness Testing launched by CNRS (AS STIC no 23).