STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LX**, Number 2, 2015

ENVIRONMENT MODEL-BASED TESTING OF REACTIVE SYSTEMS: A CASE STUDY ON A SCADE MODEL

ANNAMÁRIA SZENKOVITS

Abstract. Model-based testing can facilitate automatic test generation, thus, it can significantly decrease testing costs. This paper presents a case study where model-based testing was performed on a SCADE (Safety Critical Application Development Environment) system. Test inputs were generated automatically based on a non-deterministic, realistic environment model expressed in Lutin. The goal of the case study was to investigate whether such a realistic test environment can increase model and oracle coverage. The main contribution of the paper consists in filling in a gap in the existing literature, since there are no other works available discussing both the model and oracle coverage obtained with Lutin on a SCADE system.

1. Introduction

Model-based testing is a widely used technique in the field of verification of reactive systems. The technique consists of five main steps [16]:

(1) The system under test (SUT) and/or its environment is modelled.
(2) Abstract test cases are generated from the model — usually automatically or semi-automatically.
(3) The abstract tests are concretized in order to make them executable.
(4) The tests are executed on the SUT and verdicts are assigned (fail/pass) by the oracle.
(5) The results are analysed.

In our case study, we performed environment-model based testing — according to the above mentioned five steps — on a reactive system. The main characteristics of reactive systems is that they are in continuous interaction with their environment. During their lifecycle, they continuously read the sensor data coming from the environment, update their internal state, then write

Received by the editors: July 14, 2015.

2010 *Mathematics Subject Classification.* 68N30, 68T35.

1998 *CR Categories and Descriptors.* D.2.5 **[Software Engineering]**: Testing and Debugging – *Testing tools (e.g., data generators, coverage testing)*;

Key words and phrases. Model-based testing, Reactive systems, SCADE.

the outputs to their actuators, by which they act upon their environment. The behaviour of reactive systems can be best described using synchronous languages [7].

The model of the SUT upon which our case study was based, was designed and developed in SCADE [5]. SCADE is a synchronous, widely used industrial toolset, especially for designing avionic and railway systems. For modelling the environment of the SUT, the Lutin language [15] was chosen. Similar to SCADE, Lutin is also a synchronous language. In addition to SCADE, Lutin's syntax contains elements that enable us to describe non-deterministic behaviour, thus, to express the logic of the test environment in a more realistic way.

In order to automatically assign verdicts (fail/pass) to the execution of a test case, but also to calculate the oracle coverage rate, we used the Lustre language, another synchronous language, based on the data-flow notation [3, 7]. Lustre is also the kernel of SCADE and Lutin. To analyse the coverage of the SUT achieved during the testing, we used the SCADE Suite MTC (Model Test Coverage)[1] tool.

The paper is structured as follows. Section 2 presents the components of the testing framework. Part 2.1 describes some of the fundamental aspects of the language Lutin, focusing on how different properties of the language will be exploited in our case study. Parts 2.2 and 2.3 explain the MTC and oracle coverage criteria, respectively. Section 3 summarizes the experimental results, while section 5 presents the conclusions and future work. Finally, section 4 reviews some of the work relevant for this topic.

2. Environment-model based testing of reactive systems

The testing framework used in the current case study consisted of the following components, as illustrated by figure 1: the SUT, environment model, oracle and MTC analyser.

The SUT on which the testing was performed was the the SCADE model of an airplane's roll control system. More precisely, we used the C code generated from the SCADE model by the SCADE Suite KCG code generator[2] and executed the test cases on it.

To model the environment, but also to simulate the SUT and to generate test inputs, we used the Lutin language. This way, we were able to generate realistic test inputs and achieve a high model and oracle coverage.

[1]http://www.esterel-technologies.com/wp-content/uploads/2013/02/SCADE-Suite-MTC.pdf, downloaded on May 26, 2015

[2]http://www.esterel-technologies.com/products/scade-suite/automatic-code-generation/scade-suite-kcg-ada-code-generator/, downloaded on May 26, 2015

The test decision was performed automatically be the test oracle. Its role was to decide whether the SUT generated the right outputs for the given inputs. In addition, the oracle also contained coverage criteria based on which we analysed the oracle coverage rate. The language used for formalizing the oracle and oracle coverage criteria was Lustre. Finally, we used the SCADE Suite MTC tool to analyse how thoroughly the C code generated from the SCADE model was executed.

As illustrated in figure 1, the SUT and environment were in continuous interaction. The SUT read the inputs from the Lutin environment, updated its internal state, and generated some outputs. This cyclic behaviour continued for a specified number of iterations. Meanwhile, the oracle observed the behaviour of the SUT and decided whether the outputs generated by the SUT matched the expected outputs for the given inputs. The SUT, environment and oracles were connected by the Lurette tool[3], an automatic test generator for reactive systems. Lurette automated the test decision and stimulation of the SUT, by executing the Lutin code and feeding in the inputs generated to the SUT. In addition, Lurette also computed the oracle coverage [9].

Further on in this section, we present more in detail the components of the testing framework.

2.1. **Realistic test cases with Lutin.** By using Lutin for modelling the test environment, we were able to express non-deterministic behaviour, and to perform guided random exploration of the environment state space. Because in industrial, real-life problems the environment is often underspecified, the non-deterministic nature of Lutin makes it suitable to model realistic test scenarios.

The Lutin language is based on the use of constraints, which represent descriptions of the environment and the expected properties of the SUT. The constraints can be both boolean and numerical [14]. The constraint solver of Lutin solves the constraints and randomly selects some of the solutions [10]. This way, a random exploration of the environment is performed.

Furthermore, Lutin enables us to influence the random exploration, and thereby, to perform guided random exploration. There are two main elements in Lutin's syntax that enable us to realize this. On one hand, we have the non-deterministic *choice operator* |, as illustrated in the code example from Figure 2. Weights can be assigned to the branches of this choice operator, this way we can influence how the environment reacts. On the other hand, non-determinism can be expressed in Lutin with *random loops*, which are defined in terms of expected number of iterations. Based on Raymond et al. [15], there are two possibilities to express the expected number of iterations:

[3]http://www-verimag.imag.fr/Lurette,107.html?lang=, downloaded on May 26, 2015

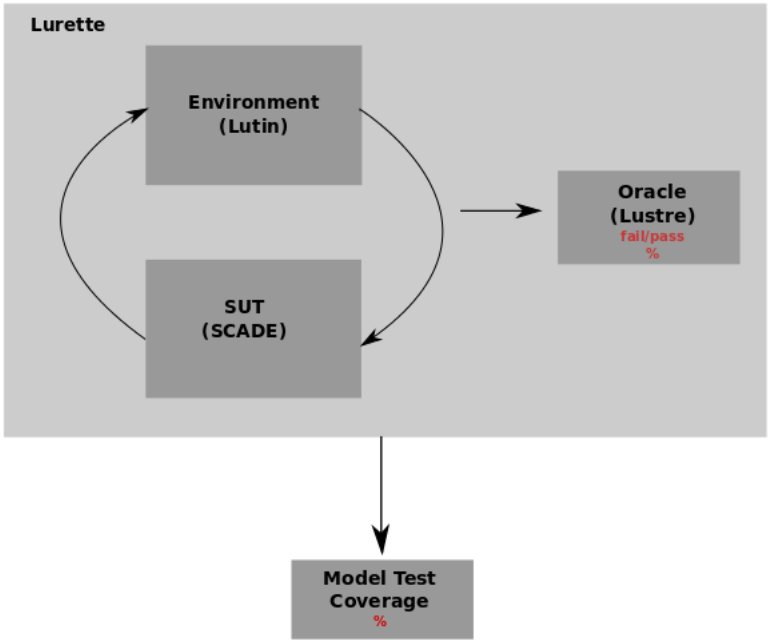

FIGURE 1. Components of the test framework: The **SUT** as a C code generated from a SCADE model with SCADE Suite KCG, the **environment** model expressed in Lutin, the **oracle** formulated in Lustre, and the **MTC analyser**. The SUT and environment are in continuous interaction with each other and have a cyclic behaviour. The execution of the SUT and environment, as well as the automation of the test decision and computation of the oracle coverage is done by the Lurette tool.

(1) `loop[min,max]`: the number of iterations should be between the constants min and max.
(2) `loop~av:sd`: the average number of iteration should be `av`, with a standard deviation `sd`.

The above mentioned control structures were used to create an environment model for the SCADE SUT. The environment model is presented in detail in section 3.

2.2. **Model test coverage.** In order to evaluate the performance of the automated test method, the SCADE Suite MTC tool was used. The MTC tool retrieves coverage data of the SUT model based on the MC/DC (modified condition/decision coverage) criterion [1]. The MC/DC is a widely-used coverage

```
node choice () returns( x :int) =
  loop {
    | 3 : x = 42
    | 1 : x = 1
}
```

FIGURE 2. Lutin code, featuring a choice operator and the weights in boldfaced font, associated with the different choice possibilities.

criterion in the model-based testing of SCADE models and of safety-critical systems in general.

One of the goals of this case study was to decide whether a higher model coverage can be obtained with a realistic environment model than with a random one. In order to investigate this question, the coverage rate obtained by the realistic environment was compared to the one obtained by some test inputs generated at random. A detailed analysis of our results can be found in section 3.

2.3. **Oracle and oracle coverage.** The SUT expected properties were formalized as Lustre predicates. We used the functional requirements specification of the SUT in order to extract the oracle information.

In addition to automating the test decision, *oracle coverage* can also be measured with Lurette. It is defined as a set of Boolean conditions, while the *oracle coverage rate* is the rate of coverage conditions that have been true at least once during the run of the simulation [9].

In our case study, we used the oracle coverage rate as feedback in order to manually refine the environment and create even more realistic test scenarios. Furthermore, we compared the oracle coverage rate with the MTC coverage rate obtained with the Lutin environment. The results of the comparison are presented in section 3.

3. EXPERIMENTAL RESULTS

3.1. **SUT model.** Our case study focused on the roll control SCADE model. The model, together with the system's specification, was provided by Ansys [4], as part of the SCADE installation package. As shown in figure 3, the system's main functionality is to calculate a plane's roll rate, based on the joystick command and the left and right adverse yaw rates. The model has three real inputs: the joystick command (generated by the pilot), and the left and right adverse yaw of the plane's wings. Based on these inputs, the SUT calculates

[4]http://www.ansys.com/, downloaded on May 26, 2015

the plane's roll rate. In addition, if this rate falls outside of a given interval, warning alarms are generated by the model, represented as boolean outputs.

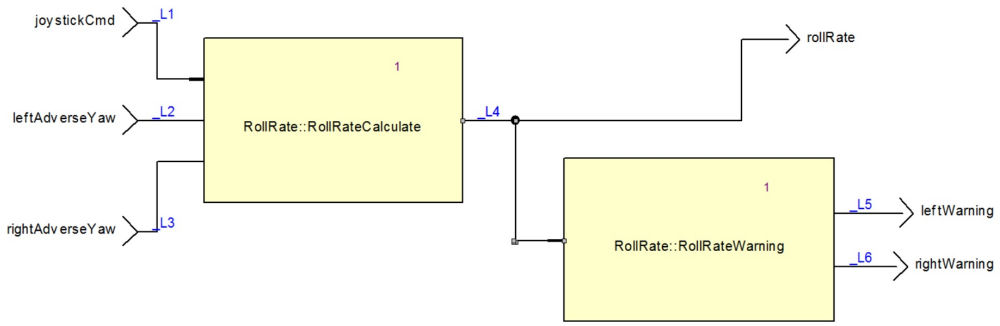

FIGURE 3. SCADE model of the roll control system. The system calculates the the plane roll rate (output **rollRate**) according to a joystick command (input **joystickCmd**) and the effect from the two plane wings (inputs **leftAdverseYaw** and **rightAdverseYaw**). In addition, the system computes the left and right warning alarms (outputs **leftWarning** and **rightWarning**) which are activated, respectively, if the plane roll rate is strictly less than -15.0° per second or strictly greater than 15.0° per second.

3.2. **Environment model.** For designing a realistic environment model in Lutin, the code from the Lutin manual[5] was used and adapted to the roll control SCADE model. The Lutin code simulates the change of the joystick command and left and right adverse yaws, respectively. Starting from an initial value chosen at random, the values start to increase or decrease by a random number, as shown in code snippets from figures 4 and 5, until they reach a certain minimum or maximum. Then, their values start to change in the opposite direction. The direction of the change is also chosen at random, using Lutin's choice operator. The number of iterations is influenced with Lutin's choice operator, as illustrated in the code snippet from figure 6. The parameters of the main node — shown in figure 7 were chosen empirically based on the oracle coverage. Figure 8 illustrates the inputs generated with Lutin after 50 steps.

3.3. **MTC analysis.** In order to analyse the effect of a realistic environment model on MTC, we generated two test suites. In the first case, we used the Lutin environment to generate input sequences of length 5, 10, 20, 50 and

[5]http://www-verimag.imag.fr/DIST-TOOLS/SYNCHRONE/lurette/doc/lutin-man.pdf, downloaded on April 24, 2015

```
let between(x, min, max : real) : bool =
        ((min < x) and (x < max))
```

FIGURE 4. Lutin combinator choosing a random number between **min** and **max**. Combinators are a kind of well-typed macros, which were introduced in the language to allow code reuse.

```
node up(init, delta:real) returns( x : real) =
   x = init fby loop
                { between(x, pre x, pre x + delta) }

node down(init, delta:real) returns( x : real) =
   x = init fby loop
                { between(x, pre x - delta, pre x) }
```

FIGURE 5. Lutin nodes increasing/decreasing the value of **x** with with a random number between 0 and **delta**. The **pre** operator accesses the value of **x** from the previous iteration.

```
node up_and_down(min, max, delta : real) returns
        (x : real) = between(x, min, max)
 fby
   loop {
        | run x := up(pre x, delta)
                        in loop { x < max }
        | run x := down(pre x, delta)
                        in loop { x > min }
       }
```

FIGURE 6. Lutin node with the choice operator |, choosing at random between the execution of the two nodes: up and down. Since there are no weights assigned to the two branches of the operator, the nodes are both chosen with probability 0.5. Once a branch is chosen, the execution of the node up/down is repeated until **x** reaches **max**/**min**.

100. In the second one, we randomly generated test sequences with the same length. The length of the input sequences was chosen empirically. The coverage obtained with the two methods was analysed with the MTC tool. The results are summarized in table 1.

A significant improvement in the coverage rate can be observed in the case of the Lutin environment compared to the random one, where the length of the input sequences was 5, 10 and 20. However, no improvements can be observed

```
node main(a:real; b:bool; c:bool) returns
        ( x:real; y:real; z:real; t:bool) =
  run x:= up_and_down(-30.0, 30.0, 10.0) in
  run y:= up_and_down(-30.0, 30.0, 10.0) in
  run z:= up_and_down(-20.0, 20.0, 10.0)
```

FIGURE 7. Main node of the Lutin code, describing the environment's behaviour. The weights were chosen empirically, based on the oracle coverage rate. Inputs: the plane's roll rate (**a**), the left (**b**) and right (**c**) warning signs; outputs: joystick command (**x**), left (**y**) and right (**z**) adverse yaws.

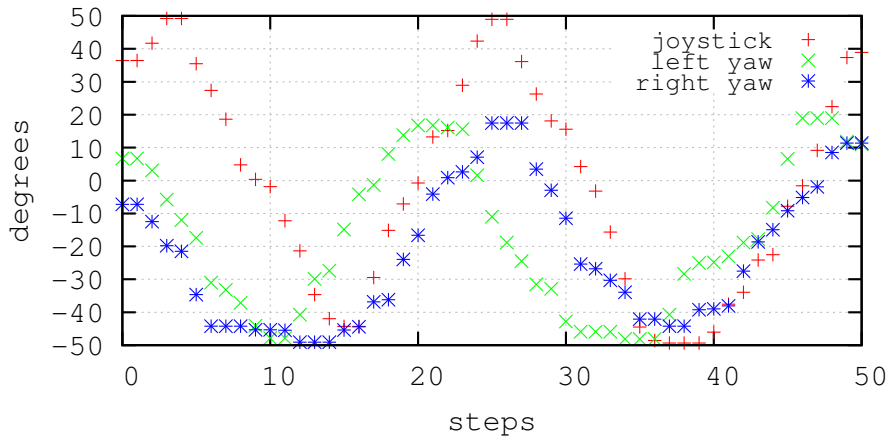

FIGURE 8. Test inputs generated with Lutin for the Roll Control SCADE system, through 50 iterations.

in the case of the Lutin environment for longer input sequences. This is due to the fact that the SUT chosen for the case study is a small SCADE model with few inputs and small inner state space. Thus, a high model coverage can be obtained even with random inputs, if the number of inputs is high enough.

3.4. **Oracle coverage analysis.** The oracle was implemented using Lustre observer nodes having access to both the input and output variables of the SUT. We focused on the requirements specifying the functioning of the roll control system's roll rate warning alarms. The complete description of the requirements can be seen in figure 9. We used two oracle nodes, as shown in code snippets from figure 10 and 11. The first oracle (figure 10) checks the correct functioning of the left warning alarm, while the second one (figure 11) supervises the right warning alarm.

Length of the input sequence	Random environment	Lutin environment
5	54.38%	68.42%
10	71.92%	75.43%
20	81.70%	84.21%
50	91.22%	91.22%
100	91.22%	91.22%

TABLE 1. MTC coverage rates obtained with the Lutin and the random environment.

	Roll Rate Warning Alarms
Short description	The roll rate warning alarms subsystem computes left and right warning alarms, which sound, respectively if the plane roll rate is strictly less than −15.0° per second or strictly greater than 15.0° per second.
Inputs	- Plane roll rate
Outputs	- Left warning alarm - Right warning alarm

FIGURE 9. Part of the roll control system's specification describing the functionality of the roll rate warning alarms. In our case study, we focused on this requirement when designing the oracle and oracle coverage. Source: official SCADE tutorial.

The oracle nodes fulfill two roles. On one hand, they check if the SUT generated outputs match the expected outputs for some given inputs according to the specification. On the other hand, the oracle nodes also contain coverage criteria. As explained in section 2.1, the coverage criteria are expressed as Lustre conditions. The values of these conditions are stored in boolean variables, which must also be added to the node's output list. Lurette calculates the oracle coverage rate based on the value of these boolean variables. A coverage criterion is considered covered if it has been true at least once during the test execution.

We used the oracle coverage rates computed by Lurette to manually refine the Lutin environment and generate more realistic test cases. In addition, we found that there is a correlation between the oracle coverage and the MTC coverage. Table 2 shows how the MTC coverage rates increased together with the oracle coverage rate.

```
node too_low(b,c,t:bool; a,x,y,z:real)
        returns (ok, c1, c2, c3:bool);
let
   ok = true -> a>=-15.0 or b;
   c1 = a<-15.0;
   c2 = a=-15.0;
   c3 = a>-15.0;

tel
```

FIGURE 10. Oracle in Lustre, checking requirement 9. If the roll rate of the plane is **smaller than -15° per second**, the left warning alarm should turn on. Boolean inputs **b** and **c** represent the state of left and right warning alarms, respectively. Real input **a** holds the value of the plane roll rate, while **x** the joystick command, **y** the left adverse yaw, and **z** the right adverse yaw. The oracle coverage is expressed as the constraints **c1**, **c2** and **c3**. It covers the cases where the roll rate of the plane lies around the lowest allowed limit (-15° per second).

```
node too_high(b,c,t:bool; a,x,y,z:real)
        returns (ok, c4, c5, c6:bool);
let
   ok = true -> a<=15.0 or c;
   c4 = a<15.0;
   c5 = a=15.0;
   c6 = a>15.0;

tel
```

FIGURE 11. Oracle in Lustre, checking requirement 9. If the roll rate of the plane is **greater than 15° per second**, the left warning alarm should turn on. Boolean inputs **b** and **c** represent the state of left and right warning alarms, respectively. Real input **a** holds the value of the plane roll rate, while **x** the joystick command, **y** the left adverse yaw, and **z** the right adverse yaw. The oracle coverage is expressed as the constraints **c4**, **c5** and **c6**. It covers the cases where the roll rate of the plane lies around the greatest allowed limit (15° per second).

4. Related Work

The research domain of model-based testing has been explored in a number of references. We mention a few of the related articles which emphasize the practical applicability of environment-model based testing methods to real-life problems. We focus especially on articles in which Lutin and Lurette were

Oracle coverage	MTC coverage
33%	**73.68%**
50%	**75.43%**
66%	**77.19%**

TABLE 2. Oracle and MTC coverage rates obtained with the Lutin environment. The length on the input sequence was 20.

used to perform the testing, but we also briefly review the work related to other environment-model based testing tools.

In [9] a case study is presented where Lutin and Lurette are used for checking the correctness of reactive systems developed incrementally. These tools are also used for elaborating and refining formal, consistent and accurate functional requirements. The authors use the oracle coverage rate for early validation of both the system model and the formal requirements, and also to improve the test scenarios. The case study presented is based on a collaboration with industrial developers of nuclear power plant control systems. One of the tested systems was developed using SCADE.

Two further case studies are presented in [8]. One of them presents a dynamic system which simulates the behaviour of the temperature and pressure of a fluid in a pipe, while the other one Lutin is used to automate the execution of timed test plans. The authors concluded that Lutin and Lustre allow the design of test plans that are more robust to software (or specification) evolutions.

The usage of Lurette for automatic generation of realistic input sequences is demonstrated also in [11]. Three industrial case studies in testing reactive embedded programs are presented here. The examples were extracted from an application developed in SCADE. The first example is a SCADE node that converts resistance to temperature, the second one is a component that computes the position of a propulsion nozzle according to the values of two sensors that measure electric tension, while the last one is a typical example of a fault-tolerant heat controller. In the presented examples, the Lucky language was used to model the environment. Working with similar modelling techniques, Lucky was basically the previous version of Lutin.

In all of the above mentioned articles — just like in our case study — oracle coverage is analysed and used to refine the environment model. However, unlike in our work, the authors of these articles don't investigate the structural coverage of the SUT model.

Beside Lutin and Lurette, there are several other languages and tools available for performing environment-model based testing on reactive systems. Bousquet et al. [2, 4] present a specification-based language called Lutess,

while Marre et al. [12, 13] describe Gatel, a test generation tool for Lustre programs. However, we could not find any case studies discussing how these tools can improve the structural coverage of the SUT model.

5. Conclusion and Future Work

We presented a case study where environment-model based testing was performed on a reactive system in form of a SCADE model. The system modelled the behaviour of a plane's roll control. To create an environment model, we used Lutin, a language that was originally designed to program stochastic reactive systems. Through the use of Lutin, we were able to design realistic environments. For automating the test decision, as well as to define the oracle coverage criteria, we used several Lustre nodes.

We analysed the SUT model coverage using the SCADE Suite MTC tool. In order to investigate whether the Lutin environment has improved the model coverage, we compared the MTC coverage rate achieved by the Lutin environment with the coverage rate obtained by a random environment. Beside the MTC coverage, we also computed the oracle coverage rate based on the Lustre oracle coverage criteria. We used this coverage rate as feedback in the process of refining the environment model and obtaining realistic test cases with Lutin.

Experimental results with the MTC tool showed that, in the case of the realistic Lutin environment, the model's coverage rate increased significantly for short input sequences of length 5, 10, and 20, compared to the coverage rate obtained with the completely random environment. However, no improvements could be observed for longer sequences with length 50 or 100. If the input sequence was long enough, a coverage rate close to 100% could be achieved. Further research however should test the method on a more complex SCADE model in the railway automation domain, coming from our industrial partner, Siemens. Here we expect more spectacular results.

Furthermore, to fully benefit of the possibilities offered by Lutin's syntax, we plan to add weights to the choice operators from our Lutin code. This way, we will have more control over the non-determinism of the environment. In addition, we plan to experiment with different learning techniques in order to improve the Lutin parameters automatically, and increase the model coverage, with input sequences as short as possible. One possibility would be the implementation of the Differential Evolution [6] technique in order to automatically fine-tune the environment model. We will use the parameters of Lutin's non-deterministic operators to build up the population of the evolutionary algorithm, and both the MTC and oracle coverage rates to measure the fitness of a population.

6. Acknowledgement

This material is based upon work supported by the Siemens international Railway Automation Graduate School (iRAGS) and the SCADE Academic Program. Thanks for Erwan Jahier from Verimag Research Center[6], for providing technical support with Lutin.

References

[1] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[2] L. du Bousquet and N. Zuanon. An overview of Lutess: A specification-based tool for testing synchronous software. In *Proceedings of the 14th IEEE International Conference on Automated Software Engineering*, ASE '99, pages 208–215, Washington, DC, USA, 1999. IEEE Computer Society.

[3] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for real-time programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 178–188, New York, NY, USA, 1987. ACM.

[4] L. du Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon. Lutess: A specification-driven testing environment for synchronous software. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 267–276, New York, NY, USA, 1999. ACM.

[5] Francois Xavier Dormoy. Scade 6 a model based solution for safety critical software development. ERTS 2008, 2013.

[6] S. Das and P.N. Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, Feb 2011.

[7] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, Sep 1991.

[8] Erwan Jahier, Simplice Djoko-Djoko, Chaouki Maiza, and Eric Lafont. Environment-model based testing of control systems: Case studies. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 636–650. Springer Berlin Heidelberg, 2014.

[9] Erwan Jahier, Nicolas Halbwachs, and Pascal Raymond. Engineering functional requirements of reactive systems using synchronous languages. In *International Symposium on Industrial Embedded Systems, 2013. SIES'13.*, Porto, Portugal, 06 2013.

[10] Erwan Jahier and Pascal Raymond. Generating random values using binary decision diagrams and convex polyhedra. In *Trends in Constraint Programming*, pages 349–356. ISTE, London, UK, may 2007. http://www.iste.co.uk/index.php?isbn=9781905209972.

[11] Erwan Jahier, Pascal Raymond, and Philippe Baufreton. Case studies with lurette v2. *Int. J. Softw. Tools Technol. Transf.*, 8(6):517–530, October 2006.

[6] http://www-verimag.imag.fr/?lang=en, downloaded on May 26, 2015

[12] Bruno Marre and Agnes Arnould. Test sequences generation from lustre descriptions: Gatel. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering*, ASE '00, pages 229–, Washington, DC, USA, 2000. IEEE Computer Society.

[13] Bruno Marre and Benjamin Blanc. Test selection strategies for lustre descriptions in gatel. *Electronic Notes in Theoretical Computer Science*, 111:93–111, Jan 2005.

[14] P. Raymond, X. Nicollin, N. Halbwachs, and D. Weber. Automatic testing of reactive systems. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 200–209, Dec 1998.

[15] Pascal Raymond, Yvan Roux, and Erwan Jahier. Lutin: A language for specifying and executing reactive scenarios. *EURASIP J. Emb. Sys.*, 2008, 2008.

[16] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

Department of Mathematics and Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania

E-mail address: szenkovitsa@cs.ubbcluj.ro