Synchronous Programs Testing Language (SPTL)

Mouna Tka Mnad, Christophe Deleuze, Ioannis Parissis

Synchronous Programs Testing Language (SPTL)

Mouna TKA Mnad [1], **Christophe Deleuze** [2], **Ioannis Parissis** [3]

1. *Grenoble INP-Esisar, Laboratoire de Conception et d'Intégration des Systèmes*
   50 rue Barthélémy de Laffemas BP 54, 26902 Valence Cedex 9
   France

   mouna.tka@lcis.grenoble-inp.fr

2.

   christophe.deleuze@lcis.grenoble-inp.fr

3.

   ioannis.parissis@lcis.grenoble-inp.fr

ABSTRACT. *SPTL is a testing language for synchronous software. The goal is to generate test input sequences in conformance with a specification of the software external behavior and of guiding directives such as profiles and scenarios. In this document we present an overview of the language with a simple example of a reactive system testing.*

RÉSUMÉ. *SPTL est un langage de test pour les logiciels synchrones. L'objectif est de générer des séquences d'entrée de test en conformité avec les spécifications du fonctionnement externe du logiciel et avec des directives pour guider ces tests telles que des profils et des scénarios. Dans ce document, nous présentons un aperçu du langage avec un exemple simple d'un test d'un système réactif.*

KEYWORDS: *software testing, synchronous approach, model-based testing, automatic test data generation.*

MOTS-CLÉS : *tests de logiciels, approche synchrone, Test basé sur les modèles, génération automatique de données de test.*

1. Introduction

Synchronous reactive systems are widely used in industry for the automation of various processes. Verification of these systems include testing the embedded software. Testing is a costly activity, so tools such as Lurette (Jahier *et al.*, 2006), Gatel (Marre, Blanc, 2005) and Lutess (Halbwachs *et al.*, 1991) have been designed to automatize it. These tools generate test sequences from some description of the system and/or test objectives. Our work is focused on a specific class of synchronous controllers produced by a big international company, leader of a large research project involving several companies and labs of Grenoble and Valence area. They can build complex applications by composing existing components through a user-friendly graphical programming tool. We aim at adding testing functionnalities to this tool. This first step is to design a testing language that must be usable by non technical users. This "Synchronous Programs Testing Language (SPTL)" will thus be close to the application design language, enriched with primitives to specify possibly non deterministic test models and scenarios.

2. Example of reactive system and overview of the language

Let's consider the following simple reactive program, an air-conditioner controller. The program inputs are: onOff (Boolean: true when the user pushes the On-Off button of the air-conditioner), Tamb (integer: value in Celsius degrees of the ambient temperature) and Tuser (integer: value in Celsius degrees of the user selected temperature). The program outputs are: IsOn (Boolean: indicates the state of the air-conditioner) and Tout (integer: indicates the temperature of the air that the air-conditioner blows).

Based on the idea that a system performance and functioning are significantly dependent on the environment in which it operates, we introduce the notion of "profile". A profile represents a possible use of the system and is related to the place and context in which it is used. More generally, the notion of "operational profile" has been introduced by John Musa (Musa, 1993).To every profile correspond some constraints of use that will necessarily affect the testing. Here comes the idea to help generating these profiles for every system. The programmer has to define some categories that contain groups of constraints and possibly sub-programs used for the calculation of values of certain variables. For example, we can consider here the categories: categoryCountrySeason (see the code below) or categoryPlace which can contain the groupHouse and the groupCompany. The constraints of each group define limits related to a particular environment or users. The test model is made of an optional set of "Declarations", followed by a set of categories definitions. A declaration must contain a declaration of variables (input/output of the system) and possibly a declaration of constants.

```
Var
    input bool onOff;
    input random int Tamb;
    input int Tuser;
    output bool IsOn;
    output int Tout;
```

```
categ CountrySeason
{  group FranceSummer
{           20<Tamb ; Tamb<44 ; Tuser = CalculTemps(Tamb); } ;
    group TunisiaWinter
{           6 <= Tamb ; Tamb <= 14 ; } ;
    sp CalculTemps (int Tamb) returns (int Tuser) – CalculTemps is a sub-program
{           Tuser = pre ( Tamb ) - 3 ; } } ;
```

A set of profiles is automatically generated by the testing environment by a combination of different groups from different categories. The tester can then choose a profile, modify it or add test cases. When testing reactive systems, a direct approach to explicitly provide typical and significant user activities is using "Scenarios".We can say that scenarios are stories. Here is a scenario within the profile "House/TunisiaWinter".

```
6 <= Tamb ; Tamb <= 14 ; onOff = prob ( true, 0.4 ) ; – "prob" means that the probability that onOff become true is 0.4
begin
{ Tuser = 8 } | [ Tuser = pre (Tuser) +1 ( Tuser = 10 ] | – "pre" allows to refer to the value in the previous cycle
{ Tuser = 12 } | [ Tuser = pre(Tuser) + 2 ( Tuser = 22 ) ] |
end
```

A scenario is composed of pointwise constraints and interval constraints. A pointwise constraint, enclosed between "{" and "}", is a constraint to be checked in one test generation time. It is an initialization of the output values and environment variables or certain constraints at specific times to test another mode of functionalities of the system for example. An interval constraint, enclosed between "[" and "]", is a constraint checked repeatedly until a condition is satisfied. The condition enclosed between "(" and ")" is a specific constraint in scenarios to allow switching from one situation to another.

3. Conclusion

We have proposed a language to write models to generate test sequences for reactive systems. It is based on describing both the constraints on the environment and the profile of the users of the program under test which will reduce the work needed to prepare realistic tests. Some standard profiles may be predefined, by the manufacturerâs engineers or by a third party, to further assist the user in its testing design. The possibility to express test scenarios allows to take test objectives into account during test generation. In our future work, we will focus on the implementation of a prototype of test data generator based on models written in the SPTL language. We will also design an oracle for comparing actual system outputs with expected ones. That will allow us to experiment with our design on a number of case studies, including complex applications developped for logic controllers.

References

Halbwachs N., Caspi P., Raymond P., Pilaud D. (1991, Sep). The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, Vol. 79, No. 9, pp. 1305-1320.

Jahier E., Raymond P., Baufreton P. (2006). Case studies with lurette v2. *STTT*, Vol. 8, No. 6, pp. 517-530.

Marre B., Blanc B. (2005). Test selection strategies for lustre descriptions in gatel. *Electr. Notes Theor. Comput. Sci.*, Vol. 111, pp. 93-111.

Musa J. D. (1993, March). Operational profiles in software-reliability engineering. *IEEE Softw.*, Vol. 10, No. 2, pp. 14–32.