GTSC: Automated Model-Based Test Case Generation from Statecharts and Finite State Machines

Valdivino Alexandre de Santiago Júnior[1], **Nandamudi L. Vijaykumar**[1], **Érica F. de Souza**[1], **Danielle S. Guimarães**[1], **Raffael C. da Costa**[1]

[1]Instituto Nacional de Pesquisas Espaciais (INPE)
São José dos Campos – SP – Brazil

valdivino@das.inpe.br, vijay@lac.inpe.br

{ericaferrso,dani.guimaraes,raffaelc}@gmail.com

Abstract. *In this paper we present GTSC, a tool that allows test designers to model software behavior using Statecharts or Finite State Machines (FSMs) in order to automatically generate test cases based on some test criteria for FSM and some for Statecharts. Three test criteria for FSM models, Distinguishing Sequence, Unique Input/Output, and H-switch cover are implemented in GTSC as well as there are four test criteria from the Statechart Coverage Criteria Family (SCCF) targeting Statechart models implemented within the tool. GTSC has been successful in generating test cases for research and development projects in the space domain, and the seven different choices of test criteria provide flexibility for the test designer to choose the one that best suits his/her application.*

1. Introduction

It is almost inconceivable to think about an industrial project that can proceed without tools. In accordance with several studies previously presented in the literature and also with a more recent survey undertook by [Woodcock et al. 2009], the lack of commercially supported tools is an impediment to take-up of formal methods for systems development. In their survey, [Woodcock et al. 2009] also remark that some comments indicate that tools are still not usable, in the words of one respondent, "by mere mortals". They identified some challenges for the development of tools to support formal methods such as support for automated deduction, and common formats for the interchange of models and analysis.

Compared to other formal methods, Finite State Machines (FSMs) [Lee and Yannakakis 1996] and Statecharts [Harel et al. 1987] are relatively easy to understand. However, even these formal methods require a basic knowledge in automata theory, and certainly demand for tools that can assist professionals in their processes related to software development. Thus, ease of use is a fundamental requirement for a tool that aims to help processes such as software testing which relates to software Verification and Validation (V&V).

In this paper we present version 2.0 of the *Geração Automática de Casos de Teste Baseada em Statecharts* (GTSC - Automated Test Case Generation based on Statecharts) environment[1], a tool that allows test designers to model software behavior by means of

[1]A very preliminary version (1.0) of GTSC was presented in [Santiago et al. 2008b]. Version 2.0 of GTSC, described in this work, has several significant improvements if compared to version 1.0.

Statecharts or Finite State Machines (FSMs) in order to automatically generate test cases based on some test criteria for FSM and some for Statecharts. Three test criteria for FSM models, Distinguishing Sequence (DS), Unique Input/Output (UIO), and H-switch cover [Souza 2010] are implemented in GTSC as well as there are four test criteria from the Statechart Coverage Criteria Family (SCCF) [Souza 2000] targeting Statechart models implemented within the tool. GTSC has been successful in generating test cases for research and development projects in the space domain, and the seven different choices of test criteria provide flexibility for the test designer to choose the one that best suits his/her application.

This paper is organized as follows. Section 2 describes GTSC's architecture and main functionalities. Section 3 presents aspects related to the usability of the GTSC tool. Section 4 presents conclusions and future directions to follow.

2. GTSC's architecture and main functionalities

Figure 1 shows the architecture of version 2.0 of GTSC. The *Graphical Editor GTSC* component has all the features that allow the test designer to model software behavior using Statecharts or FSM. It has been implemented using as a basis ArgoUML, an open source Unified Modeling Language (UML) modeling tool. One feature of ArgoUML is to support eXtensible Markup Language (XML) Metadata Interchange (XMI) allowing the exchange of data models in different languages and modeling tools. Thus, we did this graphical editor taking ArgoUML as a basis, focusing on the characteristics of ArgoUML for modeling in UML state machines (variant of Harel's Statecharts), and removing functionalities that were not relevant to the GTSC environment.

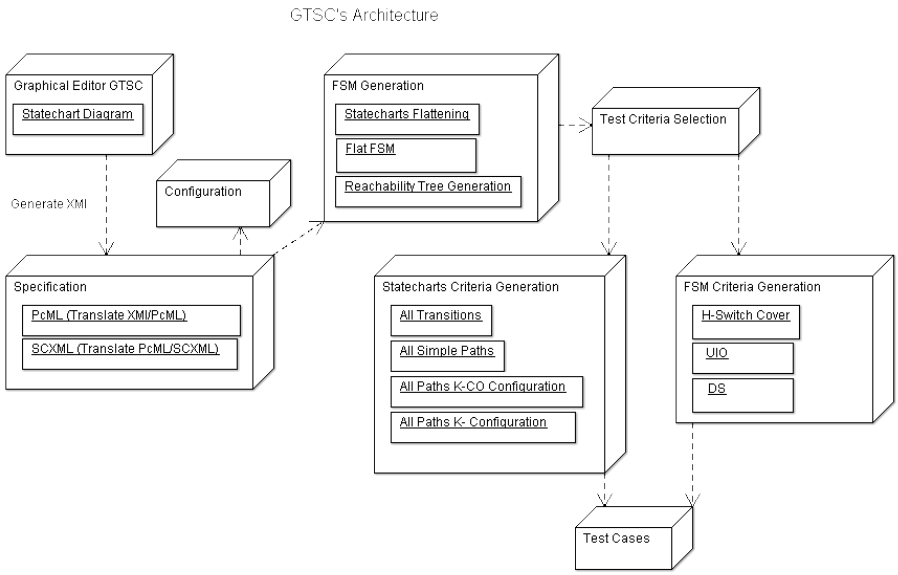

Figure 1. Architecture of version 2.0 of GTSC

The *Specification* component is responsible for converting the behavioral model from XMI into PerformCharts Markup Language (PcML) [Santiago et al. 2008b] and

State Chart eXtensible Markup Language (SCXML). The *Configuration* component handles input data necessary for generating a flat FSM, a model where all hierarchical and orthogonal features of a Statechart model were removed, from a PcML specification which represents the Statechart behavioral model. This component receives the names and locations of important files that are necessary for the operation of GTSC.

The *FSM Generation* component generates a flat FSM from a PcML specification. *PerformCharts* [Vijaykumar et al. 2006], a tool originally designed for performance evaluation and which it is currently incorporated into the *Statecharts Flattening* component, is responsible for such a translation. This flat FSM is indeed the basis for test case generation. The *Reachability Tree Generation* component of *FSM Generation* creates a reachability tree [Masiero et al. 1994] necessary for the generation of test cases based on SCCF's test criteria.

Test Criteria Selection is a component that allows the test designer to choose between the two options for generating test cases: based on test criteria from SCCF, in order to generate test cases via Statecharts, and based on test criteria for FSM models. Within the *Statecharts Criteria Generation* component four SCCF's [Souza 2000] test criteria, all-transitions, all-simple-paths, all-paths-k-C0-configuration, and all-paths-k-configurations, are implemented. This is the main component of GTSC's architecture that supports the generation of test cases via Statechart models.

Within the *FSM Criteria Generation* component, the traditional DS, UIO and a new test criterion that we designed, H-switch cover [Souza 2010], are implemented. The *Test Cases* component handles the test cases created in accordance with the model and the selected test criterion. In summary, after creating the flat FSM, there are two approaches to generate test cases. If an SCCF test criterion is selected to derive test cases, GTSC adapts the flat FSM to resemble a reachability tree [Masiero et al. 1994]. Thus, based on the selected SCCF's test criterion and on this tree, test cases are created. On the other hand, if an FSM test criterion is the option then GTSC simply generates test cases based on the flat FSM and on the selected test criterion.

The main functionalities of GTSC are: (i) it allows test designers to model software behavior using Statecharts or FSMs in order to automatically generate test cases based on three test criteria for FSM, DS, UIO, and H-switch cover, and four SCCF's test criteria, all-transitions, all-simple-paths, all-paths-k-C0-configuration, and all-paths-k-configurations, for Statechart models; (ii) it allows the user to make the models, Statecharts or FSM, according to their notations by means of its Graphical User Interface (GUI) that allows easy of use of the tool hiding the complexity related to the test criteria implemented; (iii) it allows test designers to generate test cases according to a certain SCCF's test criterion from n flat FSMs at once, avoiding the need to generate test cases considering one FSM at a time. This functionality is very interesting if we consider system and acceptance testing based on scenarios where each scenario can be represented by a Statechart model. In complex systems, the number of scenarios can be extremely large and this feature comes into picture to reduce the cost (time in this case) related to model-based test case generation for real systems.

3. Usability

Usability of version 2.0 of GTSC is briefly described in this section. Figure 2 shows the main GTSC's interface with a Statechart model. Note that this interface is basically the same that exists in version 0.28 of ArgoUML with two important differences. First, in the menu bar there is an additional menu, *GTSC*, which precisely incorporates the main features of the tool. Second, in addition to the *Diagram* tab there is the *GTSC* tab which also lets the test designer use the main functions of GTSC. Observe that it is perfectly possible, by using GTSC, to draw an FSM and use it as a basis for generating test cases instead of using Statecharts as a modeling technique.

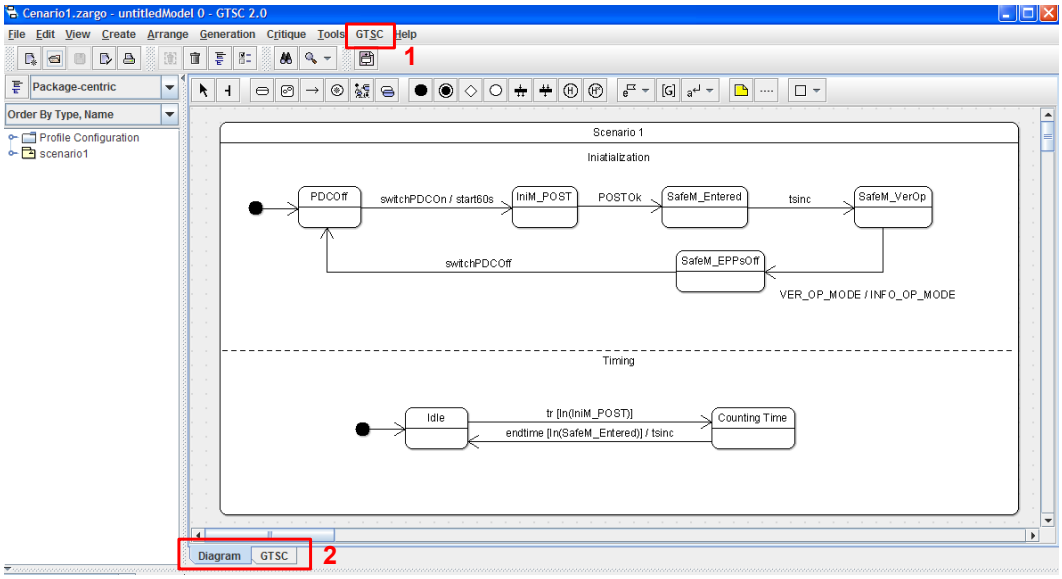

Figure 2. Main GTSC's interface

When changing from the *Diagram* tab to the *GTSC* tab, the XMI which represents the model drawn in the *Diagram* tab (Figure 2), the PcML specification, and the SCXML specification are automatically generated.

The steps to generate test cases via GTSC are basically four. In the first step (S1), the user draws the model (Statecharts or FSM) in the *Diagram* tab (Figure 2). In the second step, the user changes from the *Diagram* tab to the *GTSC* tab[2] and selects the generation of the flat FSM from the model (S2). In the third step, the user has two options. If the option is to generate test cases according to test criteria for FSM, then the user selects a specific option for this purpose in the *GTSC* tab (S3.1). If the option is to generate according to SCCF's test criteria for Statecharts, then the user selects a different option in the *GTSC* tab (S3.2). Once decided which type of test case generation (FSM or Statecharts), the fourth and final step (S4) is to simply select the test criterion and generate test cases. Figure 3 shows an example of a test suite generated for the model shown in Figure 2 and according to the all-transitions test criterion (SCCF family). The user can then save the test suite in a file for later use in the V&V process (test case execution).

[2]Due to space limitation, the interface related to this tab is not shown. The reader may refer to the GTSC User's Manual [INPE 2012] for more details about the interfaces of GTSC, installation instructions, and how to use the tool.

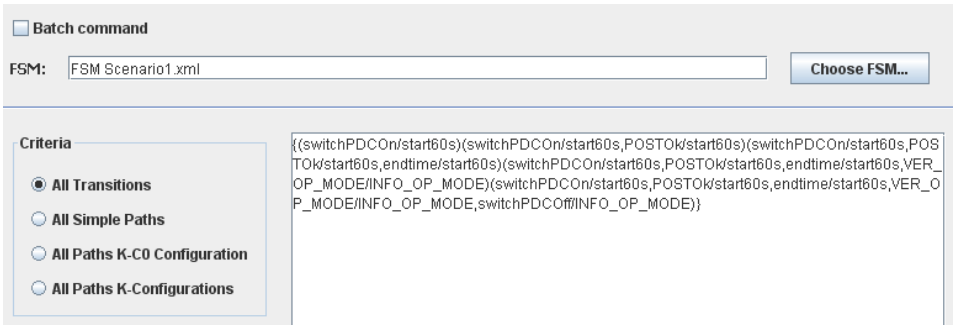

Figure 3. Piece of the interface to generate test cases via SCCF's test criteria: test suite for the model shown in Figure 2

However, the process of generation of test cases can even be more simplified. If, for example, there is a previously generated PcML specification available, it is not necessary to draw the model and the user should simply follow the second, third, and fourth steps described above. On the other hand if a flat FSM is available, only the third and fourth steps are required.

One last interesting feature is very attractive for complex software projects. In a strategy for system and acceptance testing, a common approach is to divide the interaction with the Implementation Under Test (IUT) based on scenarios. Then, for each scenario, a model (Statecharts or FSM in this context) is prepared and test cases are generated from this model. Hovewer, complex software projects usually have lots of different scenarios.

Thus, if there are n scenarios and if for each scenario a Statechart model is generated then the sequence of steps S1, S2, S3.2, and S4 must be repeated n times, and the user must manually save n files representing the n created test suites. With the *Batch command* option (Figure 3), it is possible to generate test cases according to a certain SCCF's test criterion from n flat FSMs at once and hence only the first two steps, S1 and S2, needs to be repeated n times. After generating the n flat FSMs related to the scenarios, the user can simply place them in a directory, select the *Batch command* option, inform the GTSC the directory where the FSMs are, and select the SCCF's test criterion to generate test cases. Thus, n files representing the n test suites will be created automatically and therefore steps 3.2 and 4 are repeated only once. This may significantly reduce the cost (time) required to generate test cases in complex projects with the support of GTSC.

4. Conclusions

This paper presented version 2.0 of GTSC, a tool that allows test designers to model software behavior using Statecharts or FSMs in order to automatically generate test cases based on some test criteria for FSM and some for Statecharts. In total, seven test criteria are implemented within GTSC: three for FSM models and four for Statechart models.

One of the strengths of GTSC is the ease of use. The test designer may generate test cases in a few steps (at most 4 steps), and the use of the tool is very simple by means of its intuitive GUI. Furthermore, the ability to generate several test suites at once is remarkable because this may reduce the effort related to test case generation in complex software projects. GTSC has been successful in generating test cases for research and development projects in the space domain, and the seven different choices of test criteria

provide flexibility for the test designer to choose the one that best suits his/her application. All these points are important so that there is an increased use of the theory proposed by the academic community in real projects in the industry and in institutes of research and development.

Future directions include improving GTSC to allow interoperability with other tool we have already developed [Santiago et al. 2008a], which allows both the automated execution of test cases and the automated generation of documentation related to the V&V process. Thus, it would be possible not only to automatically generate test cases via GTSC but also to automatically execute them immediately afterwards, without the need of manual steps. Moreover, it will be increased the compatibility with SCXML since the current version of GTSC can not still do the translation from the model drawn by the user to all syntactic aspects of SCXML.

References

Harel, D., Pnueli, A., Schmidt, J. P., and Sherman, R. (1987). On the formal semantics of Statecharts (extended abstract). In *Proceedings of the 2nd IEEE Symposium LICS*, pages 54–64.

INPE (2012). *GTSC-180000-SIS-003: GTSC – Manual do Usuário*. INPE.

Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines: a survey. *Proceedings of the IEEE*, 84(8):1090–1123.

Masiero, P. C., Maldonado, J. C., and Boaventura, I. G. (1994). A reachability tree for Statecharts and analysis of some properties. *Information and Software Technology*, 36(10):615–624.

Santiago, V., Silva, W. P., and Vijaykumar, N. L. (2008a). Shortening test case execution time for embedded software. In *Proceedings of the 2nd IEEE International Conference SSIRI*, pages 81–88.

Santiago, V., Vijaykumar, N. L., Guimaraes, D., Amaral, A. S., and Ferreira, E. (2008b). An environment for automated test case generation from Statechart-based and Finite State Machine-based behavioral models. In *Proceedings of the 1st ICST, 4th Workshop A-MOST*, pages 63–72.

Souza, É. F. (2010). Geração de casos de teste para sistemas da área espacial usando critérios de teste para máquinas de estados finitos. Dissertation (Master in Applied Computing), INPE, 133 p.

Souza, S. R. S. (2000). Validação de especificações de sistemas reativos: definição e análise de critérios de teste. Thesis (PhD in Applied Physics), USP–São Carlos, 264 p.

Vijaykumar, N. L., Carvalho, S. V., Francês, C. R. L., Abdurahiman, V., and Amaral, A. S. M. (2006). Performance evaluation from Statecharts representation of complex systems: Markov approach. In *Proceedings of the 26th CSBC, 5th WPerformance*, pages 183–202.

Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Computing Surveys*, 41(4):19:1–19:36.