A Model-Based Testing Methodology for the Systematic Validation of Highly Configurable Cyber-Physical Systems

Aitor Arrieta, Goiuria Sagardui, Leire Etxeberria

Computer and Electronics department
Mondragon Goi Eskola Politeknikoa
Goiru 2, Mondragón
Email: {aarrieta, gsagardui, letxeberria}@mondragon.edu

***Abstract*—Nowadays, the society is dependent on Cyber-Physical Systems (CPSs), which are complex systems that combine digital technologies and physical processes. The need for dealing with constant changes in products is leading these systems to handle variability in several aspects, which entails to a considerable increase in the complexity of the systems. Many of the research efforts are focused on the efficient development of these systems. Nevertheless, the infeasibility of testing all the possible configurations, the unclear notion of the achieved test coverage and the high amount of time required make testing processes non-systematic and challenging. This paper introduces the main problems for testing highly configurable CPSs and proposes a novel approach for testing systematically and efficiently while achieving high test coverage.**

Keywords–Model Based Testing; Test Methodology; Variability Modelling; Cyber-Physical Systems

I. INTRODUCTION

CPSs integrate digital cyber computations with, often complex physical processes, where embedded systems monitor and control physical processes with sensors and actuators [1]. The dependency of the society on CPSs that control many individual systems and complicated coordination of those systems is considerably increasing [2]. CPSs working in industries are highly complex systems [2], which make embedded systems to come with a set of configuration parameters [3]. As a consequence, the variability of CPSs increases, and the embedded systems have to deal with the changes that the physical environment requires.

Variability is the ability to change or customize a system [4], also understood as configurability (variability in the product space) or modifiability (variability in time) [5]. CPSs handling variability are commonly known as highly configurable CPSs, which are described as heterogeneous systems where hardware and software are integrated with the aim of controlling a physical process. These systems share the same embedded software code base, which has the ability of getting configured to work in systems with different features [6]. This configurability might be realized by parameters, where changes in the configuration of the product might lead to a complete different system's behaviour [7].

Testing highly configurable CPSs is a challenging task. These kind of systems can get configured into thousands or even millions of configurations, which makes it infeasible to test every single configuration and as a consequence the notion of the achieved test coverage is uncertain for quality engineers. Thus, ensuring that the CPS will meet all requirements in every possible configuration is not realistic.

Two main challenges are addressed when testing highly configurable CPSs. Each test can be executed many times, once for each possible configuration, and as a result, "the cost of running a test suite is proportional to the number of tests times configurations" [8]. Selecting concrete configurations as well as the right test cases for the selected configurations is one of the principal challenges when testing highly configurable CPSs.

On the other hand, the use of Model-Based Design (MBD) tools such as MATLAB/Simulink is increasingly growing when designing and testing CPSs. The high number of variants and the complexity of the CPSs make manual configuration error-prone and inefficient, which warrants the need for an automated solution for the configuration of CPSs [2], as well as for the test system, where a configurable test architecture handling variability becomes essential.

Section II of this paper introduces the related work. The proposed approach for the systematic validation of highly configurable CPSs using a model-based testing methodology is presented in Section III. Some discussions of the proposed approach and expected contributions are discussed in Section IV. Section V describes preliminary and expected results. Finally, Section VI presents conclusions and future work.

II. RELATED WORK

The work presented by Bauer et al. [3] proposes a systematic model-based test approach named REDUCE for the validation of highly configurable safety-critical systems. This approach uses combinatorial and model-based technologies with the main objective of reducing configurations and test cases until reliability estimations based on testing become feasible. The approach presented in [3] is focused on the reduction of configurations and possible test cases for the validation of highly configurable safety-critical systems using model-based statistical testing, while our approach will be more focused on the management and test architecture for the validation of CPSs. In addition, in the approach presented in [3], the test model is created for each system configuration, reusing similarities between different configuration-specific test models. In our approach, a test model handling

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-370-4

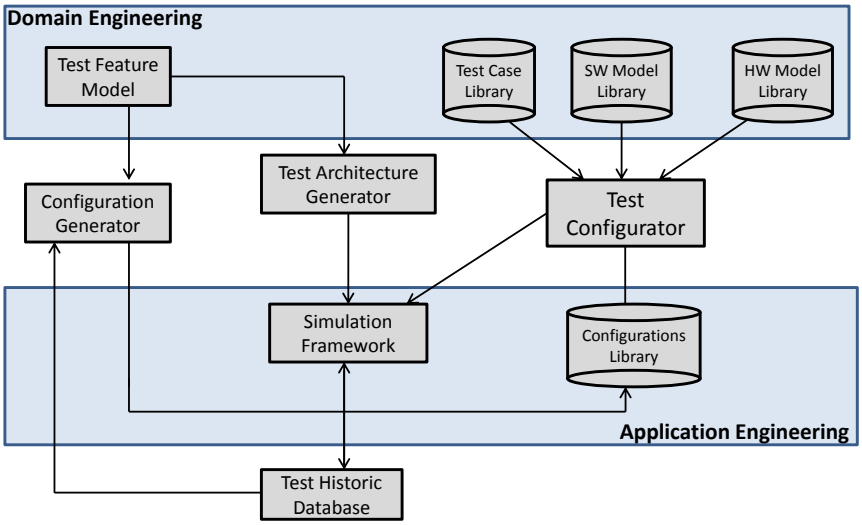

Figure 1. Overview of the proposed model-based methodology for the systematic validation of highly configurable CPSs.

variability in the test architecture as well as in the test cases is proposed, as the study presented by Weißleder and Lackner [9] demonstrates that it is more efficient binding variability after test case design.

A similar approach for highly configurable embedded systems in the automation domain is proposed by Streitferdt et al. [7]. In this case, a testing process is developed, where test cases are automatically generated with a parameter model in combination with a test model.

Combinatorial Interaction Testing (CIT) techniques are also widely used when testing configurable systems, where configurations are often selected using pairwise or t-wise techniques. According to Kuhn et al. [10], testing efficiency using pairwise testing method is 2.4 times higher and quality a 13 % better than manual testing method. CIT techniques are commonly combined with model-based testing as proposed by Oster et al. [11], where a tool chain is introduced named MoSo-PoLiTe. This tool selects pairwise configuration selection component on the basis of a feature model covering 100% pairwise interaction, and test cases are generated for each configuration.

Nevertheless, these combinatorial techniques select product configurations statically. In [8], a novel approach is presented named SPLat, where product configurations are determined dynamically during test execution by monitoring accesses to configuration variables. The technique consists in executing the test for one configuration, while observing the values of configuration variables used to prune other configurations.

Another approach to reduce validation costs of highly configurable systems is minimizing the test suite for testing a product, reducing redundant test cases. A set of test cases can be automatically obtained by selecting features of a feature model to test a new product, but there still can exist redundant test cases [12]. A fitness function based on three key factors (Test Minimization, Feature Pairwise Coverage and Fault Detection Capability) for three different weight-based genetic algorithms is defined in [12]. This approach allows reducing the test suite covering all testing functionalities achieving a high fault detection capability.

From the testing perspective, our previous work [13] presents an approach based on model-based testing and variability management integrated in Simulink, where a concrete configuration of the software is chosen by the test engineer, and the testing infrastructure is instantiated for the chosen configuration. In another previous work [14], a testing architecture with variability management in the test oracles is proposed to test distributed robotic systems in Simulink. A product line of validation environments with variability to test different applications in different domains and technologies is proposed by Magro et al. [15]. However, these works do not consider the automatic generation and configuration of the test architectures, there are not oriented for highly configurable CPSs, thus, they do not consider test case selection and test suite minimization.

Model-in-the-Loop for Embedded System Test (MiLEST) is a toolbox for MATLAB/Simulink developed by Zander-Nowicka in [16]. This test architecture is oriented for the validation of automotive real-time embedded systems in Model-in-the-Loop (MiL) phase. The main advantages of MiLEST is that it is oriented for the automotive domain. This industry is the one that most uses variability modelling according to [17]. Another advantage of MiLEST is the compatibility with Simulink, which is a simulation tool widely used to model embedded software and simulate CPSs. The test architecture used in this methodology will be based on MiLEST but it will address some changes: The developed architecture will handle variability issues and will be able of automatically getting configured. In addition, it will contain algorithms for the efficient validation of highly configurable CPSs. In addition, the test architecture will be communicated with a database with test historic data in order to prioritize test cases to be executed. Finally, the test architecture is expected to be used not only in MiL phase, but also in software, processor and hardware-in-the-loop phases.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-370-4

III. APPROACH

The methodology begins from a manual implementation of a Feature Model that manages the variability of both, the Cyber-Physical System Under Test (CPSUT) and the test architecture. The feature model is developed using the tool FeatureIDE [18], which automatically generates a .xml file. This .xml file will be used for the automatic generation of the test architecture and CPSUT Simulink model.

Once the model is automatically generated, the first configurations will be generated by the configuration generator. These configurations will be used by the test configurator to configure the simulation framework. Once configured the simulation framework, the test will be run and the results uploaded to the test historic database. Taking these results into account, the configuration generator will generate other configurations.

Figure 1 depicts the proposed methodology. Its components are classified following the theory of product lines: Domain Engineering and Application Engineering. The components in the domain engineering are for the whole product line, what means that variability is implemented. On the contrary, the components of the application engineering side correspond to a concrete product configuration, thus, variability is already bound.

A. Feature Model

According to Berger et al. [17], Feature Models are the most used notation in order to model variability of systems in industrial practice. As mentioned before, we propose using the tool FeatureIDE [18]. The main reason for using FeatureIDE is its simplicity, its completeness and the availability of its source code, which enables adapting the tool to our needs. In addition, this tool is constantly under construction, with new updates becoming available. It also uses CIT algorithms to generate correct configurations using pair-wise or t-wise techniques, which can be reused when implementing our dynamic configuration generator.

Figure 2 depicts a basic feature model from the mobile industry, where the basic functions of a feature model are shown. The features allow the following relationships: “mandatory”, “optional”, “alternative” and “or”. In addition, constraints between features can be specified by the relationships “requires” and “excludes”.

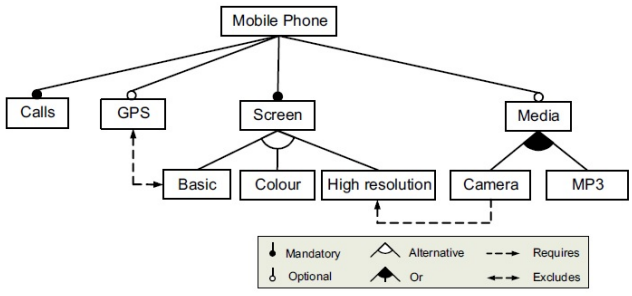

Figure 2. Example of a Feature Model from the Mobile Industry [19].

As shown in Figure 3, the feature model is composed of the CPS Feature Model, the Test Feature Model and the Integrator. The CPS Feature Model is the feature model related to the highly configurable CPS, which manages the variability that the CPS has. The Test Feature Model is related to the test architecture, and manages the variability of the test architecture. Lastly, the integrator is a file that enables the integration of all the components of the model among them, which is used to automatically generate the Simulink model of the CPSUT and the test architecture.

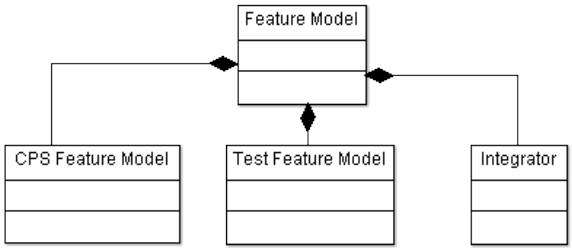

Figure 3. Meta-model of the Feature Model proposed for the systematic validation of highly configurable CPSs.

Figure 4 shows an example of how a feature model could be. The motivating example illustrates the control of the liquid of an industrial tank. The liquid can be a chemical product or water. In the case the liquid is a chemical product, the pH must be measured by a pH sensor. In addition, an optional temperature sensor can be used to measure the liquid’s temperature. The components of the CPSUT have been placed on the right branch of the feature model. On the left side, the variability of the test architecture is modelled. Traceability between both sides is modelled with constraints, as shown in Figure 4 (“requires”).

B. Test Architecture Generator

Once the feature model is built, FeatureIDE generates a .xml file which is read, together with the integrator file by the test architecture generator, which is implemented in MATLAB to automatically generate the Simulink model with the CPSUT and the test architecture. The main work of the integrator is to handle information about connection among the ports of the components’ models. The Simulink model is automatically generated using MATLAB scripts. In addition, the comonents’ models (sensors, actuators, etc.) will be designed by system and test engineers and saved into a Simulink library. Later, this library is going to be used when automatically configuring the Simulink model (as shown in Figure 5).

C. Configuration generator

One of the key points when testing highly configurable systems is to efficiently generate configurations. Current studies describe different static CIT techniques to generate configurations, e.g., [3][10][11]. FeatureIDE [18] also allows generating product configurations both automatically and manually, and these configurations are saved into a .config file.

When testing highly configurable systems, product configurations can be generated statically or dynamically. Statically

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-370-4

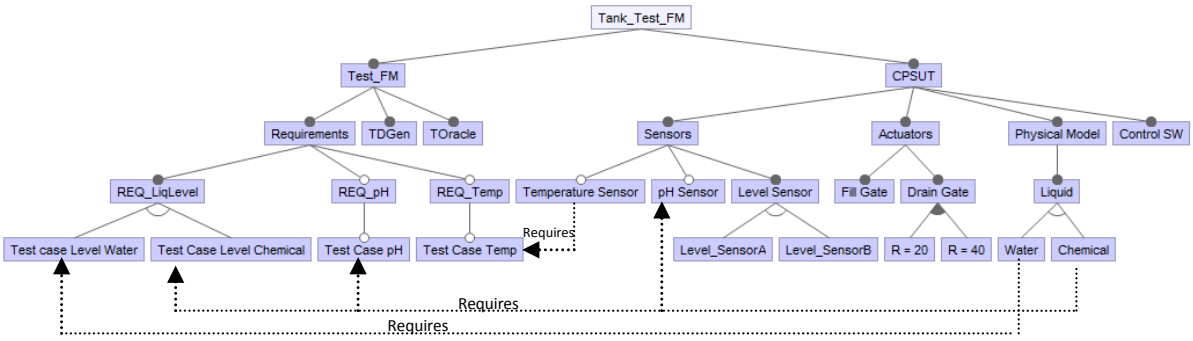

Figure 4. Example of how the Test Feature Model looks like.

generating configurations means generating a set of configurations and later testing those configurations. On the contrary, configurations are generated dynamically when the configuration generator algorithm generates a product configuration, the configuration is tested and depending on the test results and objectives the configuration generator generates another product configuration, e.g., [8].

Our hypothesis is that generating configurations dynamically is more efficient than generating them statically, as test results might have an influence when generating configurations. For instance, taking the before explained tank liquid controller example, the temperature sensor might be giving errors in the system. When generating configurations, the configuration generator should take into account this issue and generate configurations including the temperature sensor. Our configuration generator will be developed in C++, and it will communicate with both, the feature model (in order to obtain correct configurations) and the test historics database (to obtain information of the executed tests, test quality, etc.). The generated configurations are saved in a file with a .config extension and stored in the configurations library.

In order to generate configurations dynamically, it is necessary to study test quality metrics related to highly configurable CPSs. These metrics will be related to requirements coverage, features coverage, components coverage, etc.

D. Test Configurator

The configuration file is read by the test configurator, which will be implemented in MATLAB and whose main task is to automatically configure the CPSUT and the test system in a Simulink model.

To achieve this goal, the components corresponding to the CPS are allocated into a Simulink library. The previously explained test architecture generator automatically generates a first model, which consists of the principal components (Taking Figure 5 as example, Sensor1, Sensor2, Actuator1, Actuator2, REQ1 and REQ2). When the configuration is parsed by the test configurator, the principal features are replaced by the components corresponding to the CPS configuration, i.e., sensors, actuators, etc.

When modelling variability in Simulink, two kind of models can be used: 100 % models and 150 % models. The 100 % models are the models that allocate just the components corresponding to a concrete configuration. For example, first class variability modelling technique can be used to model variability this way, as proposed by Haber et al. [20]. On the contrary, the 150 % models allocate all the components in the models, the components related to a concrete configuration are selected with Simulink blocks such as switch or merged, e.g., [21] [22]. The main drawback of 150 % models is that as there are components not selected for a configuration allocated in the Simulink model, the simulation time is not optimal, which increases the overall validation time. Due to this reason, we do not rely on 150 % models, and our approach is designed to use 100 % models and optimize the simulation time.

E. Test Architecture

The test architecture is essential in order to carry out a systematic validation of any system and reuse the test cases along the whole verification and validation process. As mentioned before, the test architecture is going to be an adaptation of MiLEST [16], a test architecture developed to test real-time embedded systems of the automotive domain. The hierarchy of MiLEST is divided into four abstraction levels: Test Harness level, Test Requirement level, Test Case level and Feature level. The main components of the test architecture are shown in the meta-model depicted in Figure 6.

MiLEST will be adapted to be configurable and be able of testing any configuration of the CPSUT. To achieve this goal, its components will handle variability. Variability in the test data generator will be found in signals (number and characteristics of each signal), requirements (number and parameters of the requirements), test cases (test case duration and test case characteristics), etc. In the test oracle, variability might be found in signals (number of input signal to the oracle), requirements (number of requirements, number of validation function characteristics, parameters), etc.

In addition, a communication with a test historic database will be implemented to execute test cases according to the previous results. The test controller will encapsulate genetic algorithms with the objective of minimizing the test suite and prioritizing test cases.

Figure 6 shows the composition of the test architecture and the communication among its components. The test architecture is composed of the test data generator, the test controller

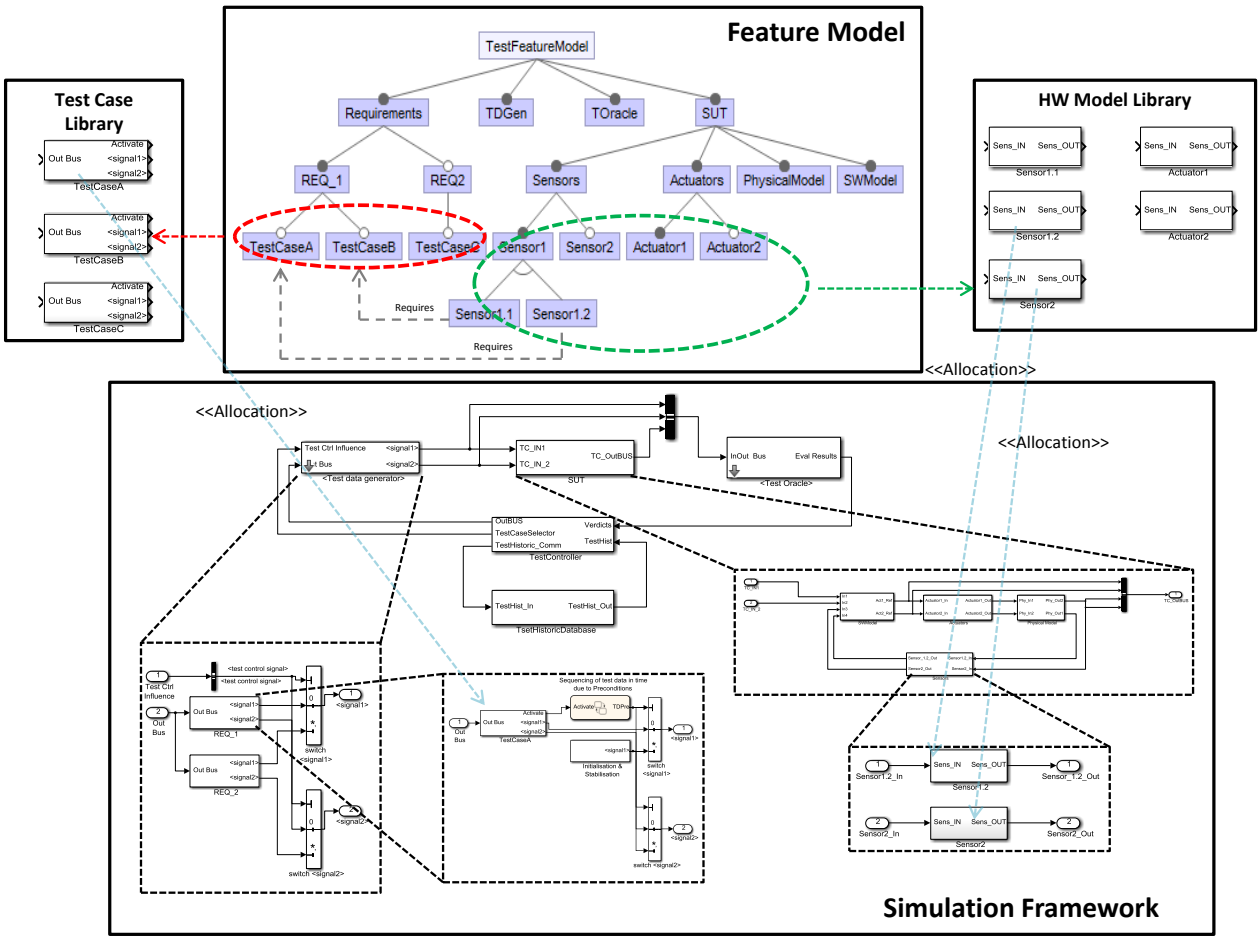

Figure 5. Overview of the relations among the features in Feature Model and Simulink Model.

and the test oracle. The test data generator is the source in charge of stimulating the CPSUT executing test cases. Test cases are implemented manually by test engineers, and each test case will test a single requirement of the system.

With regard to the test controller, it will select one test case or another depending on the test purpose; before executing any test, the test controller will obtain information about the previously executed test cases by communicating with the test historics database.

Finally, the test oracle will evaluate whether the expected behaviour of the CPSUT is correct or not. The test oracle will have one sub-oracle for each requirement, and each sub-oracle will be composed of one or more precondition and assertion, which are manually implemented by the test engineer, taking into account variability.

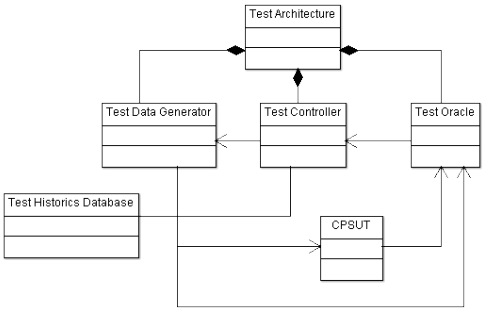

Figure 6. Meta-model of the test architecture.

F. *Cyber-Physical System Under Test*

Figure 7 shows the system to be tested, where the components are differentiated into Cyber and Physical. In our approach, the CPSUT is kept as black-box, where the test engineer does not need to be familiar with its internal behaviour [23]. Variability in a CPS can be found in the cyber as well as in the physical side. Variability in the physical side, commonly known as context variability, is related to the variability of the environment, i.e., the number and characteristics of sensors and actuators, variability of the mechanics, etc. The embedded system also has to handle variability in order to deal with the

variability of the physical side. Variability of the cyber side can be related to the software, where different configurations are achieved depending on the configuration of the physical side, or to the hardware (types of microprocessors to be used, etc.).

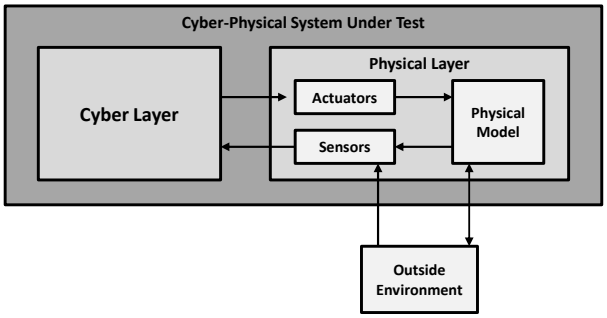

Figure 7. High level overview of the main components of a CPS.

Model-Based Design (MBD) tools are used for the development as well as for the testing of CPSs. Model-, Software-, Processor- and Hardware-in-the-Loop (MiL, SiL, PiL and HiL) tests, provide four testing phases [24], which are typically used to test CPSs in different stages and testing objectives. Our approach will study the possibility of testing the CPSUT in these four stages automatically.

Figure 7 also depicts a block named outside environment. This block refers to the environment in which the CPS resides, which often has a strong influence into the behaviour of the CPS. Considering the example of a mobile robot, the outside environment would include the obstacles to which it is exposed, e.g., the slopes, the surface or even the temperature or humidity which can lead to an inadequate performance of the on-board electronics.

IV. Discussion

The goal of the proposed methodology is to systematically validate highly configurable CPSs. To achieve that goal it is necessary to obtain the needed configurations and test cases to ensure the correctness of highly configurable CPSs in any possible product configuration. The main contributions of the study are foreseen to be the following:

- A methodology to systematically manage the validation of highly configurable CPSs. This methodology begins with the implementation of a feature model. A feature model is a notation that represents the features and relations among them of all possible products of a Product Line represented as a hierarchically arranged set of features [19]. This feature model handles the variability of both, the CPS and the test system. From this feature model, the optimal configurations are set in order to satisfy the maximum coverage. In addition, a Simulink model is generated automatically integrating the configured CPS and the test architecture.
- A configurable test architecture in Simulink that handles variability issues for the validation of configurable CPSs. This test architecture, based on [16], will include test data generators and test oracles handling variability as well as other components. The variability of the hardware and the software architecture requires variability in the test model at the verification and validation stages [7]. The test architecture together with the simulated CPS will get configured according to the chosen configuration.
- Algorithms for the efficient validation of configurable CPSs achieving high test coverage will be studied. The algorithms will combine dynamic CIT techniques, test suite minimization approaches and test case prioritization strategies. These algorithms will choose the most optimal configurations and will select and prioritize test cases to be executed for the chosen configuration.
- Test quality metrics for highly configurable CPSs. As mentioned before, it is infeasible to test all the possible configurations in highly configurable systems, and as a consequence, the notion of the achieved test coverage is uncertain. Different kinds of test quality metrics will be analysed to ensure the correctness and quality of the highly configurable CPS for any configuration.

V. Results

In our previous work [25], some experiments have been performed for the automatic generation of the CPSUT, where a novel variability modelling methodology is proposed for the plant models of highly configurable CPSs. In these experiments, we use FeatureIDE [18], a feature modelling tool for managing variability. With this tool we manage the variability of the physical side of the CPSUT and the .xml file is generated. With the .xml file, a first Simulink model is generated semi-automatically. Human intervention is needed to integrate the different components (sensors, actuators, mechanical components, embedded systems) of the model. Once generated this model, configurations are achieved either manually or automatically from Feature IDE, and each configuration returns a ".config" file. With this file, the configurator configures the model of the CPSUT.

The expected results include a reduction on the time needed for the validation of highly configurable system, at the same time as incrementing the obtained test coverage (requirements coverage, feature coverage, components coverage, etc.). This goal can be achieved by reducing simulation time through the analysis of the most effective variability modelling methodology. In addition, it is essential to select the optimal CPS configurations to be tested and to automate the simulation framework. In regard to test cases, it will be important to select the appropriate ones, prioritizing them and minimizing the test suite as much as possible.

VI. Conclusion and Future Work

This paper identified the main challenges to face when validating highly configurable systems. The main problems can be summarized into (1) the automatic configuration of the CPS and the test infrastructure, (2) the unclear notion of the achieved coverage and (3) the need for reducing test execution time. As a possible solution, we propose a model-based testing methodology to efficiently and systematically validate highly configurable CPSs.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-370-4

A highly configurable CPS can achieve different configurations, in the software as well as in the hardware, which leads to the need of automating the configuration of the model of the CPSUT as well as the configuration of the test system at verification and validation stages. The described methodology proposes the automatic generation and configuration of the test system and the CPSUT for Simulink models from Feature Models.

A highly configurable CPS can be configured into thousands or even millions of configurations. Testing each possible configuration is impracticable. Hence, the notion of the achieved test coverage is uncertain. Optimizing the execution time of test cases as well as configurations to be set up is essential. We propose dynamic multi-objective algorithms with the aim of achieving the highest possible test coverage (requirements, feature and component coverage), using the minimum CPS configurations and the minimal test execution time.

VII. ACKNOWLEDGEMENTS

This work has been developed by the embedded systems group supported by the Department of Education, Universities and Research of the Basque Government.

REFERENCES

[1] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling cyber-physical systems," Proceedings of the IEEE (special issue on CPS), vol. 100, no. 1, January 2011, pp. 13 – 28.

[2] T. Yue, S. Ali, and K. Nie, "Towards a search-based interactive configuration of cyber physical system product lines," in ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems, Poster, 2013, pp. 71–75.

[3] T. Bauer et al., "Combining combinatorial and model-based test approaches for highly configurable safety-critical systems," in 2nd Workshop on Model-based Testing in Practice, 2009.

[4] J. V. Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in Proceedings of the Working IEEE/IFIP Conference on Software Architecture, ser. WICSA '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 45–54.

[5] S. Thiel and A. Hein, "Modelling and using product line variability in automotive systems," IEEE Software, vol. 19, no. 4, 2002, pp. 66 – 72.

[6] R. Behjati, T. Yue, L. Briand, and B. Selic, "Simpl: A product-line modeling methodology for families of integrated control systems," Simula Research Laboratory, Tech. Rep., 2011.

[7] D. Streitferdt et al., "Model-based testing of highly configurable embedded systems in the automation domain," International Journal of Embedded and Real-Time Communication Systems, 2011, pp. 22–41.

[8] C. H. P. Kim et al., "Splat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 257–267.

[9] S. Weißleder and H. Lackner, "Top-down and bottom-up approach for model-based testing of product lines," in MBT, 2013, pp. 82–94.

[10] R. Kuhn, R. Kacker, Y. Lei, and J. Hunter, "Combinatorial software testing," IEEE Computer Society, vol. 42, 2009, pp. 94–96.

[11] S. Oster, I. Zorcic, F. Markert, and M. Lochau, "Moso-polite - tool support for pairwise and model-based software product line testing," in VaMoS, 2011, pp. 79–82.

[12] S. Wang, S. Ali, and A. Gotlieb, "Minimizing test suites in software product lines using weight-based genetic algorithms," in Proceedings of the 2013 Genetic and Evolutionary Computation Conference, Amsterdam, Netherlands, 2013, pp. 1493 – 1500.

[13] G. Sagardui, L. Etxeberria, and J. A. Agirre, "Variability management in testing architectures for embedded control systems," in VALID 2012 - 4th International Conference on Advances in System Testing and Validation Lifecycle, Lisbon, Portugal, 2012, pp. 73 – 78.

[14] A. Arrieta, I. Agirre, and A. Alberdi, "Testing architecture with variability management in embedded distributed systems," in IV Jornadas de Computación Empotrada, ser. JCE 2013, 2013, pp. 12–19.

[15] B. Magro, J. Garbajosa, and J. Perez, "A software product line definition for validation environments," in 12th International Software Product Line Conference (SPLC), Piscataway, NJ, USA, 2008, pp. 45 – 54.

[16] J. Zander-Nowicka, "Model-based testing of real-time embedded systems in the automotive domain," Ph.D. dissertation, Technical University Berlin, 2008.

[17] T. Berger et al., "A survey of variability modeling in industrial practice," in Variability Modelling of Software-intensive Systems (VaMoS), 2013, pp. 7:1–7:8.

[18] T. Thuem et al., "Featureide: An extensible framework for feature-oriented software development," Science of Computer Programming, vol. 79, 2014, pp. 70 – 85.

[19] D. Benavides, S. Segura, and A. Ruiz-Corts, "Automated analysis of feature models 20 years later: A literature review," Information Systems, vol. 35, no. 6, 2010, pp. 615 – 636.

[20] A. Haber et al., "First-class variability modeling in matlab/simulink," in ACM International Conference Proceeding Series, 2013, pp. 4:1–4:8.

[21] G. Botterweck, A. Polzer, and S. Kowalewski, "Using higher-order transformations to derive variability mechanism for embedded systems," in Models in Software Engineering. Workshops and Symposia at MODELS 2009, Berlin, Germany, 2009, pp. 68 – 82.

[22] C. Dziobek, J. Loew, W. Przystas, and J. Weiland, "Functional variants handling in simulink models," Mathworks, Tech. Rep., 2008.

[23] M. Z. Z. Iqbal, "Environment model-based system testing of real-time embedded systems," Ph.D. dissertation, University of Oslo, 2012.

[24] H. Shokry and M. Hinchey, "Model-based verification of embedded software," Computer, vol. 42, no. 4, 2009, pp. 53 – 59.

[25] A. Arrieta, G. Sagardui, and L. Etxeberria, "Towards the automatic generation and management of plant models for the validation of highly configurable cyber-physical systems," in Proceedings of 2014 IEEE 19th Conference on Emerging Technologies & Factory Automation, ser. ETFA 2014, no. (To be published), 2014.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-370-4