# Architecture Based on Keyword Driven Testing with Domain Specific Language for a Testing System

Ricardo B. Pereira, Miguel A. Brito, Ricardo J. Machado

# Architecture based on keyword driven testing with domain specific language for a testing system

Ricardo B. Pereira, Miguel A. Brito and Ricardo J. Machado

Centro ALGORITMI, Dep de Sistemas de Informação, University of Minho
Guimarães, Portugal
ricardo-97-pereira@hotmail.com
{mab, rmac}@dsi.uminho.pt

**Abstract.** For Cyber-physical systems (CPSs), whose task is to test industrial products, to carry out these tests, highly qualified engineers are always needed to design the tests, since the computational part of the tests is programmed in low-level languages. To optimize this process, it is necessary to create an abstraction of current methods so that tests can be created and executed more efficiently. Although this problem has arisen within the CPS, the architecture we propose will be generic enough to solve the problem in any testing system. We intend to do this by automating some of the current processes to minimize human error. In this paper, we present a novel architecture for a testing system that abstracts single low-level programming and coding of tests, based on two main concepts: the use of Keyword Driven Testing (KDT) that will abstract tests to the person responsible for the machine; the creation of a Domain Specific Language (DSL) to help configure and design new tests without requiring the experience of a highly qualified engineer.

**Keywords:** Cyber-Physical Systems · Test Automation · Keyword Driven Testing · Domain Specific Language · Architecture

## 1 Introduction

Cyber-physical systems (CPSs) are integrations of computing, network, and physical processes. Embedded computers and networks monitor and control physical processes. A CPS integrates the dynamics of physical processes with software and the network, providing abstractions and modeling, design, and analysis techniques for the integrated whole.[4] According to the state-of-the-art, the CPSs provide the necessary technology to improve the realization and automation corresponding to a complex system on a large scale. Currently, CPSs require solutions that support it at the system level. This is a challenge that includes an engineering approach and a fusion of information and automation technologies[1,2,3]. Traditional testing systems are adapted to each case, requiring a very expensive and time-consuming effort to develop, maintain, or reconfigure. The current challenge is to develop innovative, and reconfigurable

architectures for testing systems, using emerging technologies and paradigms that can provide the answer to these requirements[5]. The challenge is to automate the maximum number of tasks in this process and get the most out of the testing system, be it a CPS or just software.

What we intend with this research is to optimize the process of creating and executing the tests. Although this problem has arisen within the CPS, the architecture we propose will be generic enough to solve the problem in any testing system that supports the use of the software. We intend to do this by automating some of the current processes to minimize human error. In this paper, we present a novel architecture for a testing system that abstracts single low-level programming and coding of tests, based on two main concepts: the use of Keyword Driven Testing (KDT) that will abstract tests to the person responsible for the machine; the creation of a Domain Specific Language (DSL) to help configure and design new tests without requiring the experience of a highly qualified engineer.

Section 2 provides a background on KDT and DSL, as these are two important concepts to understand the architecture presented. Section 3 describes and presents the proposed architecture. Finally, Section 4 concludes and identifies future work.

## 2 Background

**Keyword-Driven Testing** is a type of functional automation testing framework which is also known as table-driven testing or action word based testing. In KDT, we use a table format, usually a spreadsheet, to define keywords or action words for each function that we would like to execute. It allows novice or non-technical users to write tests more abstractly and it has a high degree of reusability. The industrial control software has been having an enormous increase in complexity as technology has developed and requires a systematic testing approach to enable efficient and effective testing in the event of changes. KDT has been proving that it is a valuable test method to support these test requirements.[6] Recent results from other researchers have shown that the design of the KDT test is complex with several levels of abstraction and that this design favors reuse, which has the potential to reduce necessary changes during evolution. Besides, keywords change at a relatively low rate, indicating that after creating a keyword, only localized and refined changes are made. However, the same results also showed that KDT techniques require tools to support keyword selection, refactoring, and test repair.[7]

**Domain-Specific Language** is a language meant for use in the context of a particular domain. A domain could be a business context or an application context. A DSL does not attempt to please all. Instead, it is created for a limited sphere of applicability and use, but it's powerful enough to represent and address the problems and solutions in that sphere. A DSL can be used to generate source code from a keyword. However, code generation from a DSL is not

considered mandatory, as its primary purpose is knowledge. However, when it is used, code generation is a serious advantage in engineering. DSLs will never be a solution to all software engineering problems, but their application is currently unduly limited by the lack of knowledge available to DSL developers, so further exploration of this area is needed.[8] Other researchers used DSL in CPSs and left their testimony of how the specification language hides the details of the implementation. The specifications are automatically enriched with the implementation through reusable mapping rules. These rules are implemented by the developers and specify the order of execution of the modules and how the input/output variables are implemented.[9]

## 3 Architecture

The architecture that we propose in this paper aims to automate and facilitate the process of creating new tests. To have a complete understanding of the architecture and how its components interconnect, it will be explained first how to use KDT and DSL and only after how they work together.

### 3.1 KDT

We will use KDT to abstract low-level code scripts, associating each script with a keyword that will represent it most descriptively and explicitly possible. This way the user will not need to know the details of the script implementation, but only what it does. We will also associate each keyword with metadata related to the corresponding test, which will be stored in a database. Figure 1 shows the approach taken in using KDT. Note that the names given to the tests in the figure are only fictitious names to demonstrate that the names given to the keywords must be as descriptive as possible.

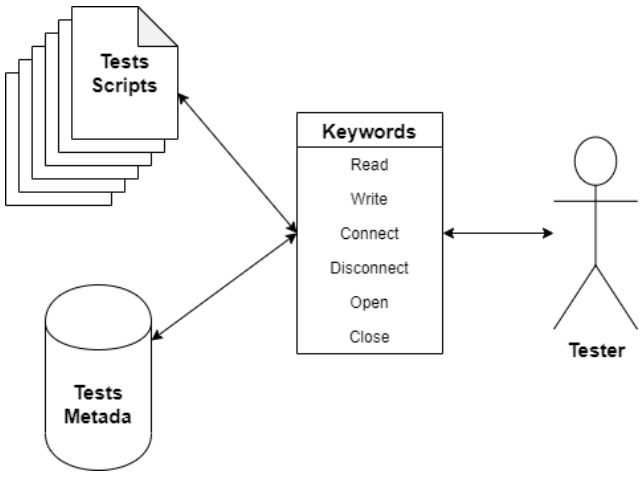


**Fig. 1.** Keyword Driven Testing Approach

In Figure 1 we see a stack of sheets that represent the test scripts already existing in the system, in this case, they should be primitive tests that focus only on testing a feature or a set of features as long as they can be well-identified only by a word that can serve to be a keyword. We also see the representation of a database that will be where all the information and metadata about the tests existing in the system will be stored, that is, the same ones that are represented in the stack of sheets previously explained. Connected to the database and the stack of sheets, we have a table with keywords in which each keyword represents all the information related to a test. This table is the most important element in the figure because it is where we can relate all the information in the stack of sheets and the database, and this is achieved with just one word which makes it possible for users/testers with little programming knowledge to be able to interpret what each test does or means. Finally, we have the connection between the Tester and the table of keywords that demonstrates that the Tester will only have access to the keywords without needing to know any details of implementation.

### 3.2 DSL

This use of KDT alone does not bring great advantages because we still need someone to design a test execution flow according to their purpose. This is where the importance of DSL comes in, as it allows us to define a friendly language for workers, without the need for very sophisticated programming knowledge. The proposed language is extremely simple, but it allows the creation of new scripts with new execution flows and logical rules applied, only with the use of keywords defined by the use of KDT and some symbols previously defined in the DSL. Table 1 shows the terminal symbols of the defined DSL and what they represent.

| **Symbol** | **Description** |
|---|---|
| keyword | Catches the keywords in the script. |
| -> | Catches the "next" symbol, which means that after that symbol the next block to be executed arrives. |
| ? | Catches the conditional expressions from the script. |
| ( | Catches the opening parenthesis. |
| ) | Catches the closing parenthesis. |
| : | Catches the next block of code to be executed when a condition is false. |
| ; | Catches the end of the script. |
| & | Catches the logical operator that means intersection. |
| \| | Catches the logical operator that means union. |

**Table 1.** DSL Symbols Description

### 3.3 Proposed Architecture

To achieve the full potential of this architecture, a final abstraction of all these processes is necessary. In Figure 2 we can see the diagram of the final architecture that ensures to abstract the whole complex process of creating new tests for the system, thus giving the possibility to users less endowed with programming knowledge to be able to build new tests.

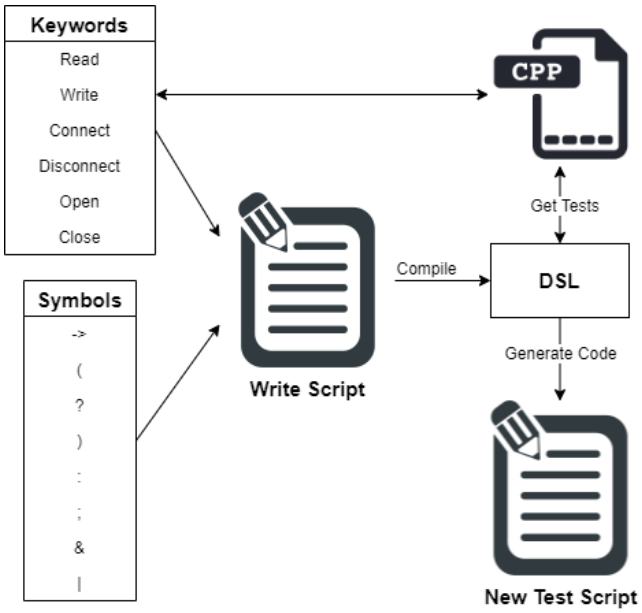


**Fig. 2.** Architecture

The two tables that are illustrated in Figure 2, "Keywords" and "Symbols" represent the elements that can be used to form scripts in a lexically correct manner. The syntactically correct way of writing scripts will be explained in the next section where we show an application example. The elements of the "Keywords" table contain the keywords that correspond to the tests that are defined and available in the system to be used as part of new tests. It is also possible to check the connection between this table and the existing tests programmed in lower-level languages, such as C++. The elements of the "Symbols" table contain the terminal symbols of the defined DSL and are what allow to give logic and organization to the new tests of the system. Therefore, the Tester will be able to write the script with the elements available in these two tables and that is exactly what is represented with the connections between the tables and the "Write Script" element of the figure.

Subsequently, the DSL will analyze the script written by Tester using a Lexer and a Parser and will check if it is syntactically and lexically well written. The

"Compile" connection represents this step of analyzing and verifying the script. If the script is correct according to the defined rules, the DSL will compile that script and generate the code for a new test. The DSL needs to access the code of the test scripts that were used through the keywords and it does so as can be seen in the "Get Tests" connection. The DSL will form a new test based on the tests that Tester specified using the keywords. This is only possible because the DSL can match the tests to the keywords that identify them. At the end of this process, we have the connection that shows us that the DSL generates the code for the new test and from that moment it is available for execution.

### 3.4 Example of Application

In this chapter, we will present a complete example of creating a new test with this architecture to demonstrate its simplicity and efficiency. In this example, we consider that our scope of tests will be the same shown in the "Keywords" table represented in Figure 2. The symbols that we can use will be those shown in Table 1, as they are the symbols that the developed DSL recognizes. The first step is to write the script with the available keywords and symbols. In this example we will use this script:

```
( Connect & Open ) ? ( Read -> Write -> Close ) : ( Disconnect ) ;
```

Here the scripts corresponding to "Connect" and "Open" will be executed and if both return a positive result (a positive or negative result will be defined by those who create these test primitives) the execution will follow to the block just after "?". If any of the scripts return a negative result, the next block of execution will be the one after the ":" symbol. The block after "?" will execute the three scripts corresponding to "Read", "Write" and "Close" sequentially in the order they are specified in the script. The block after ":" will execute only the script corresponding to "Disconnect".

Now that the script has been written, it will pass through the Lexer defined on the DSL, which will analyze whether all elements that are in the script are part of the language dictionary. In this case, all symbols will be recognized successfully and then it is the DSL Parser's turn to continue with its parsing and check that all the sentence formation rules are respected. Once verified, as this script is correct it will be compiled by the DSL and the source code of the new test script will be generated at that moment. The way the new script is created is by using the existing source code of the tests that are referenced in the script by the keywords and adding to it the logic applied by the DSL symbols used in the script.

## 4 Conclusions and Future Work

In this paper, we presented a novel architecture for a testing system that will allow cyber test components to have a much simpler and more efficient way to create new tests.

This architecture demonstrates how we can use KDT and DSL to achieve great abstractions and automate the process of creating new tests. This type of architecture gives a boost to the world of CPS research, but also to the general context of testing systems, since it is generic and can be applied both in the context of CPS and in any other context that allows the use of software.

The future work will be to continue the evolution of this new architecture, implementing a functional prototype and a system to be integrated into the CPS industry. And with that to be able to contribute even more to the investigation bringing real results of the application of the architecture.

## Acknowledgment

This paper is a result of the project POCI-01-0247-FEDER-040130, supported by Operational Program for Competitiveness and Internationalization (COMPETE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

## References

1. Leitão, P., Colombo, A. W., and Karnouskos, S. (2016). Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. Computers in Industry. https://doi.org/10.1016/j.compind.2015.08.004
2. Liu, Y., Peng, Y., Wang, B., Yao, S., and Liu, Z. (2017). Review on cyber-physical systems. IEEE/CAA Journal of Automatica Sinica. https://doi.org/10.1109/JAS.2017.7510349
3. Seshia, S. A., Hu, S., Li, W., and Zhu, Q. (2017). Design Automation of Cyber-Physical Systems: Challenges, Advances, and Opportunities. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. https://doi.org/10.1109/TCAD.2016.2633961
4. Lee, E. A. (2008). Cyber physical systems: Design challenges. Proceedings - 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2008. https://doi.org/10.1109/ISORC.2008.25
5. Leitão, P. (2009). Agent-based distributed manufacturing control: A state-of-the-art survey. Engineering Applications of Artificial Intelligence. https://doi.org/10.1016/j.engappai.2008.09.005
6. Hametner, R., Winkler, D., and Zoitl, A. (2012). Agile testing concepts based on keyword-driven testing for industrial automation systems. IECON Proceedings (Industrial Electronics Conference). https://doi.org/10.1109/IECON.2012.6389298
7. R. Rwemalika, M. Kintis, M. Papadakis, Y. Le Traon and P. Lorrach, "On the Evolution of Keyword-Driven Test Suites," 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), Xi'an, China, 2019, pp. 335-345.
8. Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM Computing Surveys. https://doi.org/10.1145/1118890.1118892
9. S. Ciraci, J. C. Fuller, J. Daily, A. Makhmalbaf and D. Callahan, "A Runtime Verification Framework for Control System Simulation," 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, 2014, pp. 75-84.