

A Framework for Automatic Testing of Industrial Controller Code

Dag Kristiansen and Karl-Petter Lindegaard

ABB Corporate Research Center
P.O. Box 90, N-1375, Billingstad, Norway
Phone: +47 03500, Fax: +47 22874750

E-mails: dag.kristiansen@no.abb.com, karl-petter.lindegaard@no.abb.com

Keywords: Automatic testing, industrial controller code, functionality testing, OPC.

Abstract—*The current trend in many industries is the introduction of more sophisticated optimization and control algorithms for enhancing production, regularity, and the operation of the plant. Some industries, like the automotive, have successfully used rapid prototyping tools like The Mathwork's Real-Time Workshop® in order to make development, testing, and implementation of new solutions more efficient. However, more traditional industries, like the process industry, have not yet fully adopted these new tools for various reasons. However, as the trend is towards more use of advanced optimization and control solutions also in these industries, the demand for testing will be more and more prominent. In this paper, we present a new framework for data-driven functionality testing of industrial controller code and applications. This new tool enables unit, integration, system, and regression testing for industrial controller code and is one of many steps towards bridging the gap between traditional desktop software development and industrial controller code development.*

1. Introduction

A control system is a computer system with a particular purpose; to control an external process in real-time. Examples of such processes are chemical plants, ships or airplanes, power plants, and facilities for oil and gas production and processing. Common for all control systems irrespective of the process they are supposed to handle is that they differ greatly from regular office applications. Typical requirements for a controller platform include:

- Hard real-time constraints. The system has limited time to respond to external events and to perform its tasks.

- Predictable scheduling of multiple tasks of different priorities.
- Cyclic execution of task. Read inputs – execute code – write outputs.
- A distributed system of CPUs requires a reliable and synchronized exchange of data.
- Safety issues and how to handle unexpected events.

More advanced features are redundancy and online editing of the executable code. The main objective for a redundant control system is of course robustness and safety. For instance, if a controller that handles a critical process for some reason fails, another unit should immediately take over its responsibilities and bumplessly resume execution.

Ability to update the controller code and parameters online is requested in cases where it is not tolerated to shut down a plant while doing maintenance on the control system. Rebooting a controller for smaller updates is often unacceptable for either economical, operational, environmental, or safety reasons.

Even though the context of industrial controller applications differs from office or enterprise applications, the typical processes for developing applications on top of a controller platform are very similar to more traditional software processes. Considering the vast amount of tools and practices available for regular software development, it is fair to say that the tools for control system development in general are somewhat limited [3]. In addition, the customers want applications that are modular, not proprietary, such that different applications from different vendors can easily be integrated. This makes integration testing very important. In this paper, we present a new *data-driven functionality* testing tool intended for industrial controller code and applications. The chosen approach is a stand-alone pc-application that is modular, using OPC [2] for data

exchange between the testing tool and the controller on which the controller code is running.

2. Terminology

We use the term *granule* throughout this paper to denote a separate piece of software with specific purposes. *Test granule* is the code that is to be tested. This term is preferred instead of component, unit, and system, all of which have less generic definitions.

Black-box testing is when the code to be tested is treated as a black box whose behavior can only be determined by inspecting its inputs and the related outputs. This means that the testing is done without reference to internal functions or variables. Another name for this is functional testing because the tester is only concerned with the functionality and not the implementation of the software.

White-box testing verifies the structure of the software itself and requires complete access to the granule's source code. This is also called structural testing. One example of white-box testing is so-called coverage analysis that monitors execution, records executed code segments and branches, keeps track of the frequency of execution of sections of code, percent of total execution time used by area of code, and whether a section of code is used or not.

Gray-box testing is functionality testing in accordance with black-box testing. However, what differs gray-box testing from black-box testing is that gray-box testing has access to the internal variables in the test granule (both read and write operations).

Tests are usually systematized and organized using so-called *test cases*. Test cases are designed to test certain features or functionalities of the test granule. This is done by first selecting a feature of the system or component that is to be tested. Then, one specifies a set of inputs that execute this particular feature along with predicted outputs and initial (default) variable values. The initial values are important for total control and verification of the test results (e.g. regression testing and documentation) since different initial values may lead to different outputs irrespectively of the specified inputs. Hence, for data-driven functionality testing, a test case is composed of three different signal specifications: 1) Specification of default/initial values, 2) specification of input signal values as functions of time, and 3) specification of expected results (output signals). The goal of the test case design is to create a set of test cases that covers the functionalities of the test granule and are effective in discovering program defects and showing that the system meets its requirements [5].

3. Industrial^{IT} 800xA

In this paper, as an example of an industrial

controller platform, we will use the Industrial^{IT} 800xA platform from ABB. The 800xA family of controllers consists of various controllers from which the customers can select the ones that fulfill their needs. One example is the AC 800M which is a modular, scalable, and energy-efficient controller well suited to a wide range of industrial control applications, whether centralized or distributed, and from basic logic to advanced control or any mix of these two. Redundancy is available at a number of critical levels, i.e. control networks, processor modules, power supplies, fieldbus media, and process inputs/outputs. The low power consumption eliminates cooling problems and facilitates compact installation. The AC 800M is configured and programmed using a Windows application called Control Builder M. Control Builder M supports all the five IEC 61131-3 standard languages [4], i.e. Instruction List, Structured Text, Function Block Diagram, Sequential Function Chart, and Ladder Diagram. In addition, ABB have extended the IEC 61131-3 POU (Program Organization Units) types with Control Modules. Control Modules can include program code, visual representation, interaction, data acquisition, communication etc, and is a powerful concept for turning new solutions into "black boxes" that can be re-used over and over again.

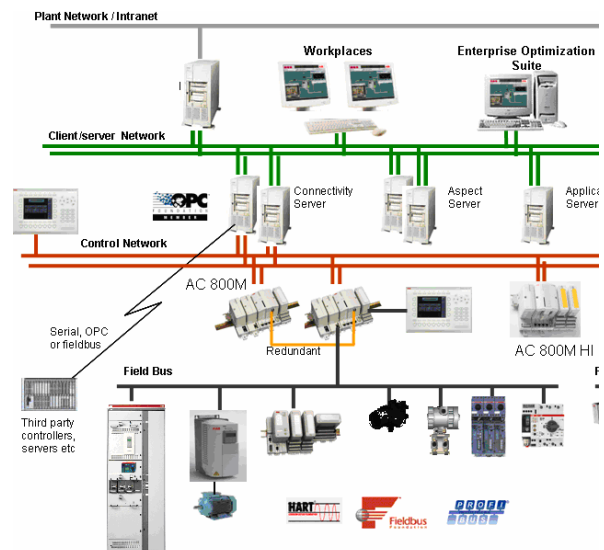


Fig. 1: Industrial^{IT} 800xA.

For simulation and test purposes, 800xA offers a pc-based application, called SoftController, which emulates the AC800M hardware controller. Thus, code can be downloaded to a SoftController, simulated and tested, before it is downloaded to the physical controller.

4. ConTest – A framework for data-driven functionality testing

Although testing of controller code is possible in the 800xA framework, the possibilities for systematic and automatic functionality testing are somewhat limited which makes this kind of testing not always very efficient. In order to increase engineering efficiency and reducing the cost of poor quality, ABB has developed a generic framework for automatic testing of controller code which is well-suited for the 800xA platform. The testing tool is called ConTest (CONtroller code TESTing), and it automates functionality testing of controller software and can be applied to smaller modules as well as larger applications. The underlying idea is very similar to unit testing: Specify inputs to a module, execute it, and verify that the actual outputs are in accordance with the expected results. ConTest supports the creation of input and output “scenarios” (test cases) in order to grasp the nature of dynamic systems. ConTest also automatically generates detailed test reports for formal test documentation and internal progress reporting which eliminates the need for the traditional manually written test descriptions and reports.

The initial focus for ConTest was to enable efficient and systematic testing of IEC 61131 controller code functionality developed in ABB’s Control Builder M. The basic requirements for the ConTest system were to support 1) Unit testing throughout the development process, and 2) Integration and system testing. When evaluating different design alternatives, it was realized that these requirements could be fulfilled by using a signal level testing approach, since - as a consequence of this methodology - there would be no conceptual difference between unit, integration, and system testing. The overall scope of the ConTest system is shown in Fig. 2.

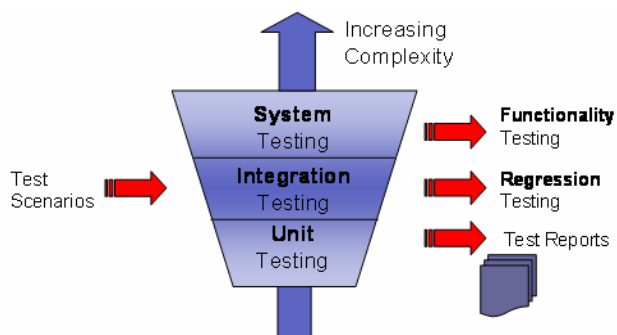


Fig. 2: Scope of the ConTest system.

In addition, it was decided to develop a stand-alone testing framework using .NET and relying on OPC [2] for signal transfer instead of developing a plug-in

for the Control Builder M environment. The main reason for the latter decision was that an OPC-based stand-alone testing tool could be applied to anything that was OPC DA compliant and hence not necessarily restricted to the IEC 61131 language, Control Builder M, and SoftController.

During the design, one of the main priorities was to keep the threshold of using ConTest as low as possible. This warranted a lean system with focus on usability with a small, but powerful set of testing functionalities, instead of a huge testing framework with complex functionality that would be difficult to use without the proper training or by studying extensive user manuals. Also, it would be beneficial to mimic tools like NUnit/JUnit, when appropriate, in an effort to bridge some of the gap between traditional software development (.NET, Java etc) and controller software development (e.g. IEC 61131). This relates both to the functionality and the graphical user interface. For instance, it was decided to adopt the concept of traffic lights used in NUnit/JUnit.

4.1. OPC

As mentioned above, OPC was chosen as an interface between ConTest and the platform on which the test granule is running. OPC, which is based on Microsoft DCOM, is a standard interface for access to Windows-based applications in automation technology. OPC has become one of the most popular industrial standards among users and developers over the last years. Most of the HMI (Human Machine Interface), SCADA (Supervisory Control and Data Acquisition), and DCS (Distributed Control System) manufacturers in the field of PC-based automation technology, as well as manufacturers of Soft PLCs, are offering OPC client and/or OPC server interfaces with their products. The same is true for suppliers of devices and interface cards [2].

Since most controllers (including soft-controllers) are OPC DA compliant, this means that data-driven functionality tests can be executed using OPC for data-exchange between the controller and the testing tool (Fig. 3).

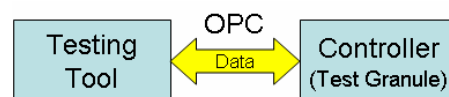


Fig. 3: OPC is very well suited for data-driven functionality testing on a signal-level basis.

When OPC is used for data exchange, it will be straight-forward to allow the tester to specify log signals, that is, signals whose values should be recorded (e.g. in a database). This is important in order to easily go back and inspect previously run test cases.

4.2. Data Synchronization

One significant challenge when doing rigorous data-driven functionality testing is data flow synchronization. To illustrate this, we look at the following example:

Assume the controller task to be tested consists of a pressure controller that adjusts the opening of a valve based on a pressure measurement. Suppose one important feature of the controller is that if the pressure measurement exceeds a certain limit, an alarm should be set and the controller should automatically switch to manual mode and freeze its output at its current value. The requirement specification document states that this should happen immediately without any delays whatsoever. This means that if we do not have 100 percent control over which output data values that correspond to which input data values (in time) and how these are related to the controller's task execution, we cannot test this functionality properly. The ideal setup for executing such a test would be as shown in Fig. 4.

The scheme shown in Fig. 4 is also called stepping (of the controller). By using controller stepping, the tests can in theory be performed much faster than real-time depending on how much time each step takes. ConTest is designed to support stepping of the ABB SoftController. However for cases where 1) freezing the controller task and execute one cycle at a time are not possible, or, 2) the test granule consists of more than one task, the only option is to perform the testing in real-time.

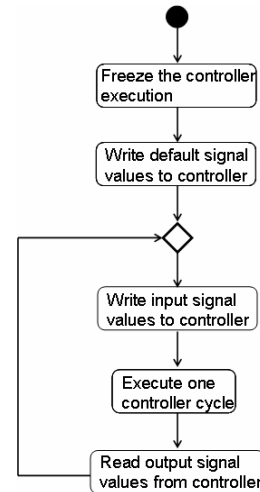


Fig. 4: Full control over the data flow synchronization between controller and testing tool.

4.3. Design

A conceptual drawing showing the ConTest building-blocks is shown in Fig. 5

At the core of the ConTest system there is an xml-file <test.xml> that contains all the information needed for executing the tests and the test results (configuration data, test case specifications, and passed/failed status of the tests). Hence, many components in ConTest only have to relate to an xml-file with a pre-defined schema definition making the design rather modular. Also, version control of test cases will be straight-forward since this basically boils down to version control of an xml-file.

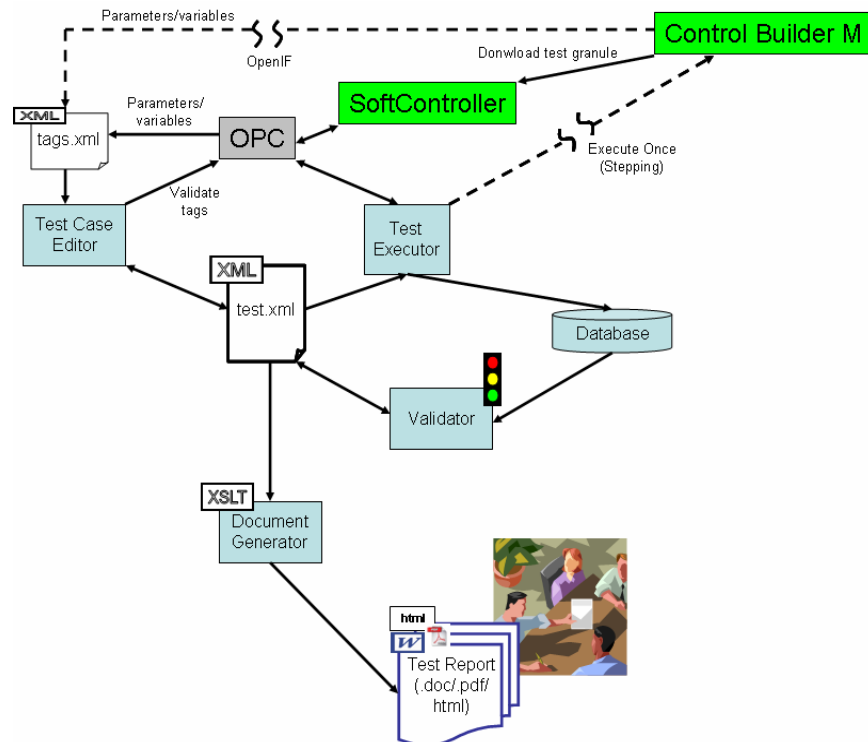


Fig. 5: ConTest design – executing tests via OPC DA when test granule is developed in ABB's Control Builder M and downloaded to a SoftController.

4.3.1 Test Case Editor

The user can define and configure the test(s) that are to be executed in the Test Case Editor. This includes

- Browsing and selecting OPC-tags that are to be written to and read from during the execution of the tests.
- Specifying default values of test granule parameters and variables in order to get unambiguous and repeatable test results.
- Specifying input signal behavior (Fig. 6), that is, what values the input signals should have as a function of time. Depending on what data-type the tag is, various pre-defined signal types are available including constants, sine-waves, and ramps. The user can also select to plot the trends of the specified input signals to make it easier to specify and verify the signal values.

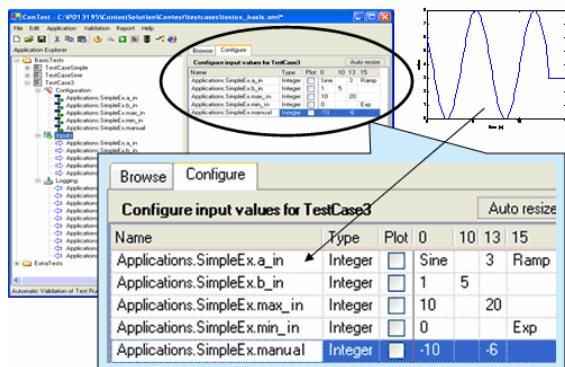


Fig. 6: ConTest Test Case Editor - Specification of input signals as function of time.

The nature of the input signals should be “rich” enough such that the functionality of the test granule is properly excited. ConTest offers a wide range of pre-defined signal types, e.g. sine waves, ramps, exponential functions, constants, or any combinations thereof, providing the users with a lot of flexibility when designing test cases. In addition, input signal values can be read from a pre-generated file.

An example of an input signal consisting of three different signal types is shown in Fig. 7.

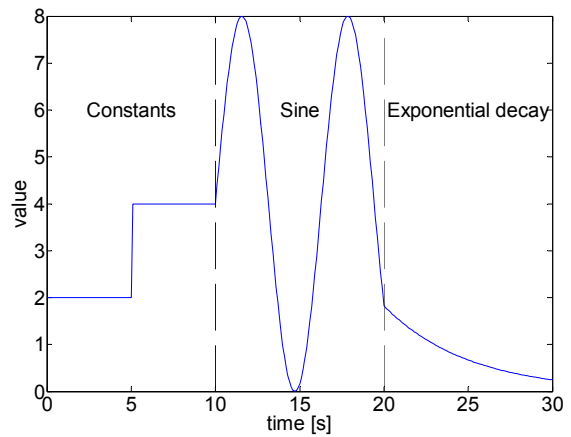


Fig. 7: Example of input signal consisting of constant values, a sine, and an exponential decaying function. The input signals are specified such that they properly excite certain functionalities of the test granule.

- Specifying expected results used for automatic validation of the results (Fig. 8), that is, how the output signals are expected to behave as a result of the behavior of the input signals.

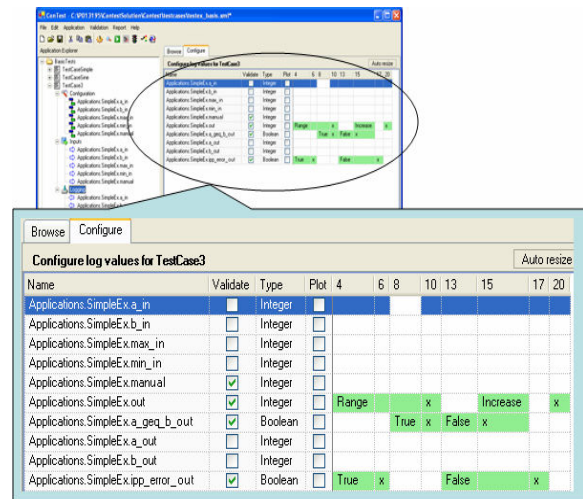


Fig. 8: ConTest Test Case Editor - Specification of which signals to write (log) to the database and the expected results.

The types of checks (expected results) that can be performed on the output signals depend on the data types of the output signals. Some of the checks that can be performed are listed in Table 1.

Table 1: Typical verification criteria for expected results.

Data Type	Types of automatic verification			
	Value	Range	Gradient	Equality
Real		X	X	
Integer	X	X	X	X
Boolean	X			X

The “Gradient” criteria simply checks whether the slope of the output signal is within a specified range, whereas the “Equality” criteria compares two or more

signals with each other.

An example of an output signal and its specified behavior is shown in Fig. 9.

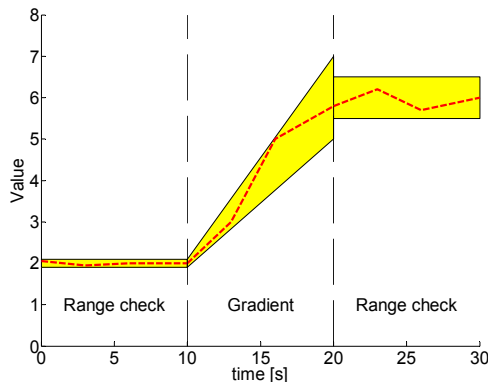


Fig. 9: Example of an output signal (dashed line) and its specified expected results.

4.3.2 Test Executor and Database

The Test Executor executes the tests and logs the output signals from the test granule. It reads all the configuration and execution data from the <test.xml> file and sends and receives data (via an OPC client) to and from an OPC server that is connected to the test granule. For the SoftController, the tests can be executed either in real-time or by stepping the SoftController. The logging is done by writing the tag

values into a database. The database contains all values for the tags that are logged.

4.3.3 Validator

The Validator compares the output signals with their expected results as specified by the user in the Test Case Editor and stored in the <test.xml> file and reports whether the tests failed or passed (Fig. 10). The reporting of the automatic validations are done using traffic lights and Gantt diagrams of passed/failed intervals (Fig. 10). ConTest can also generate plots of the output and input signal trends for manual inspection of the test results.

4.3.4 Document Generator

The Document Generator automatically generates a test report containing all information about the executed tests. The Document Generator simply uses Extensible Style Language Transformations (XSLT) to transform the <test.xml> file, which now also contains the test results, into a more readable format such as HTML, PDF, or MS Word. The transformation can be based on pre-defined templates resulting in an official test report.

It is important to note that even though the ConTest framework supports fully automated testing, in some cases, some of the output signals may be easier to verify manually by plotting the trends than

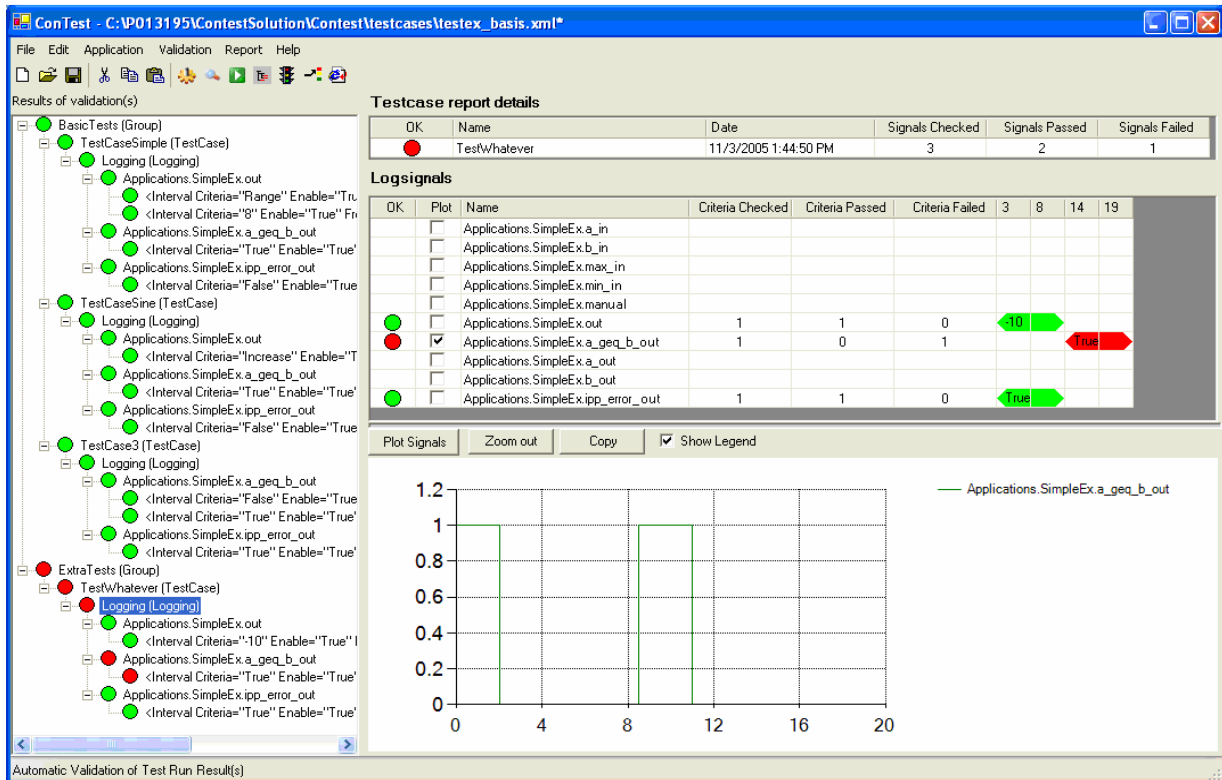


Fig. 10: ConTest Validator. The red lights on the left hand side - signaling a failed test - are aggregated to the top level resulting in a failed test case.

specifying the expected results beforehand. This is true when the expected results are non-trivial and therefore time-consuming to specify. Hence, ConTest allows any mix of automated and manual validation of the test results.

5. Future Work

There are several additional testing features that, as a consequence of the OPC-based approach of ConTest, are natural extensions of the existing framework. ConTest could easily be extended to offer efficient closed-loop simulations by bridging OPC-tags from the control systems to an OPC DA compliant process simulator as illustrated in Fig. 11.

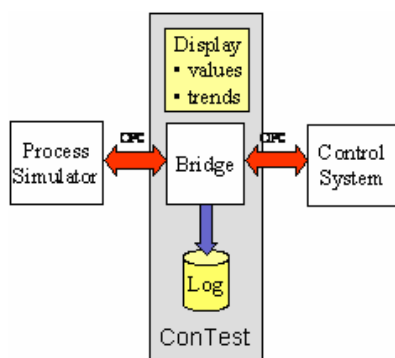


Fig. 11: ConTest as a simulator bridge.

In this case, ConTest serves as a signal bridge between the controller and the process simulator. Apart from serving primarily as a router, it will be possible to monitor and verify more realistic scenarios than the ones specified using normal test cases. Moreover, this opens the door for performance testing and parameter tuning because the process model emulates the actual system being controlled. Functional tests remain valid, and discrepancies can be reported. Simulator testing is feasible only for larger constructs or for applications. Normally, it is not very efficient to test units, with a limited set of functionality, with a simulator.

ConTest could also be extended to provide automatic validation of tests during e.g. Factory-Acceptance-Tests (FATs) and similar manually conducted tests resulting in an automatically generated official test report. This would significantly reduce the time spent on FATs.

6. Conclusion

In this paper we have presented a framework for data-driven functionality testing intended to be applied for industrial controller code and applications. ConTest offers systematic and automatic testing using a signal level testing methodology and the industry standard OPC as the signal interface. The

advantages of this approach are several: There are no conceptual differences between unit, integration, or system testing, it works for all kinds of applications irrespective of size, and it is applicable to all OPC DA compliant controllers.

ConTest is one of many steps towards bridging the gap between the tools available for testing of traditional software (e.g. NUnit/JUnit) and the tools available for testing of code developed on industrial controller platforms.

7. References

- [1] J. Gren, "Description of function – CB Open Interface", *ABB Automation Technologies AB*, Doc.no. 3BSE033313, Rev. B, 2003-10-30.
- [2] F. Iwanitz and J. Lange, *OPC – Fundamentals, Implementation, and Application*, 2nd rev. ed., Hüthig Verlag, 2002.
- [3] K.-P. Lindegaard and D. Kristiansen, "Signal Based Functionality Testing of Control Systems", submitted to *IEEE International Symposium on Computer-Aided Control Systems Design (CACSD '06)*, 2006, Munich, Germany.
- [4] PLCopen website, <http://www.plcopen.org>.
- [5] I. Sommerville, *Software Engineering*, 7th ed., Addison-Wesley Publishers Ltd, 2004.