

Proxy-based component testing with TTCN-3 ¹

**Andreas Hoffmann, Axel Rennoch,
Alain Vouffo-Feudjio and Svetlana Skljarski**

Fraunhofer Institute FOKUS,
CC TIP
Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany,
phone: +49 30 3463-7000, fax: +49 30 3463-8000,
email: a.hoffmann@fokus.fhg.de
<http://www.fokus.fraunhofer.de/tip>

Abstract: The continuously increasing use of innovative component-based software technology in the software systems developing process underscores the need for appropriate means for testing in that area. It has to address active unit and (sub)system testing during system development, as well as passive (online) monitoring of running applications, once deployed on their target environment. Test systems for component-based software have to consider specific circumstances of the System under Test (SUT) but do also benefit from component technology, as the test system could be partly or entirely designed using the same approach. In this article we concentrate on the application of the international standardized test technology TTCN-3 for the observation, analysis and arbitration of component-based software. Special emphasis has been given to the CORBA components standardized by the OMG.

Keywords: CORBA components, CCM, QEDO, Online testing, TTCN-3, Monitoring

¹ This work is partially supported by the European IST Project Coach (IST-2001-34445, Component Based Open Source Architecture for Distributed Telecom Applications, <http://www.ist-coach.org>).

1 INTRODUCTION

Today's system engineers are putting more emphasis to component-based software development. With the OMG's CORBA Component Model (CCM) software developers who benefit already from the advantages of CORBA programming can now migrate and enhance their applications with CCM. There is still a need to provide appropriate means to validate large component-based software systems.

The European project COACH [3] is concerned with the rapid and cost effective development of large scale and mission critical distributed applications by using CCM components. Within COACH a CCM test environment has been specified [11] and is currently under development that covers a component test framework for test specification, test implementation and test execution. In this paper we present basic ideas of a CCM-based approach for testing and monitoring CORBA component-based applications.

Our approach considers the Testing and Test Control Notation (TTCN-3) [5][8], a new test specification and test implementation language, that supports all kinds of black-box testing of distributed systems. The syntax looks similar to a typical programming language like C, C++ or Java and should therefore be easy to understand and apply by someone familiar with programming. TTCN-3 is an international standard which allows good readability and common understanding of test scenarios semantics.

In the following sections we present the details on a tool environment for the automatic generation of CCM-based proxies and the information data types issued by the proxy components. Furthermore the proposed solutions on the application and adaptation of TTCN-3 for CORBA components testing are given. An example in the context of online testing and a summary conclude this contribution.

2 PROXY-BASED TESTING

The proxy-based testing idea has been identified as a suitable concept to monitor both CORBA-based and also component-based distributed systems [1]. In our approach we focus on a System under Test (SUT), which consists of one or more components that are considered as "black boxes". The interfaces of the involved components are well defined with OMG's Interface Definition Language (IDL v3.0), i.e. the SUT components are CORBA components which comply to the CCM standard of the OMG. This allows accessing both the external client side interfaces of the SUT but also the interfaces within the SUT, i.e. between SUT components.

Testing a CCM based SUT (sub)-system also benefits from the standardized interface implementation between components under test. CCM allows the substitution of components which is subject to the compliance of the interface definitions. CCM components can be introduced in the SUT to control and observe the communication of SUT components. They will be called *proxies* in this context since they take over the direct interface of particular components of the SUT. Proxy components are part of the test environment, they do cooperate with other test system parts, e.g. they report on their interactions with the SUT components.

Due to the different communication types we have to distinguish operation-based and event-based communication between CCM components of a SUT. The IDL specification and the implemented *business logic* of the corresponding proxy components of the CCM test environment are different, too, but are both subject of an automatic generation process.

2.1 Proxy Information Events

The basic functionality of the proxy components is the observation of the communication between SUT components. They do not necessarily evaluate or interpret the observed operations. Therefore some other parts of the test infrastructure have to be informed on the observation. In some cases it could be that an evaluation component receives and combines the observation of a set of different proxies to assess the validity. In some other cases there may be several arbiter components that do justify the distributed observations even if they are from the same proxies. Both variants require basic information on the proxy observations.

If proxy components do not log and evaluate their observations locally but distribute the information to other test system parts, CCM event technology can be used for the transport of the information.

In case of operation-based interfaces we have to distinguish between an incoming request (from the calling component to the proxy) and the outgoing reaction (from the called component to the proxy). The latter one could be either a regular output or an exception that is due to the request. In the following we distinguish three kinds of information: *ProxyRequest*, *ProxyReply* and *ProxyException*. The signaling of the *ProxyRequest* information may be optional if the resulting reaction of the target component has to be considered only. If the

target component reacts usually the proxy issues a ProxyReply otherwise a ProxyException event. Furthermore a proxy that observes event-based communication will issue a so-called *ProxyEvent* information.

The minimal information about a facet operation observed by a proxy comprises the location (identifier) of the proxy, the operation (name) invoked between SUT components, and basic information about the result of the operation, i.e. if the operation have been succeeded or caused any exception. For advanced test cases that require all details on the observations a more refined event structure is necessary.

The full analysis and interpretation of ProxyRequest information requires knowledge on the location and time of the observed request as well as in particular on the facet operation itself, i.e. operation name and parameters. In case of multiple invocations of the same operation on the same proxy a request identifier is demanded.

The following figure provides a class diagram of the proxy information related eventtypes:

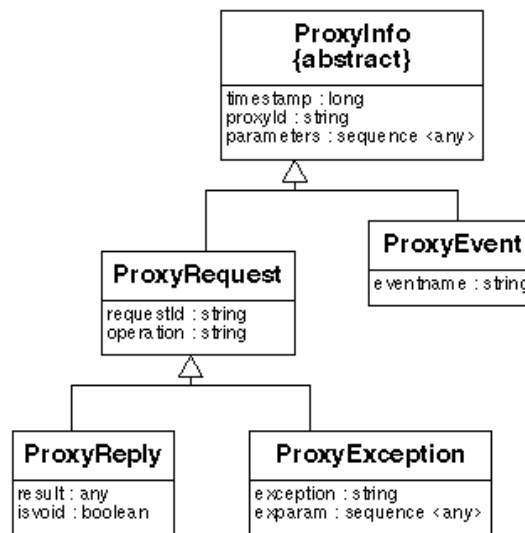


Figure 1: Proxy information class diagram

2.2 Automatic Generation of proxies

The IDL-3 specification and behaviour (business logic) implementation of the proxy can be generated automatically. The proxy components have

- a) to forward their invocations to their dedicated SUT components, and
- b) to provide the information on the observations to other test system components.

In our work the event publishing alternative (using the event types specified above) has been selected to transmit the observations.

Due to the component technology every programming language that is considered by CCM can be used for the implementation of the test system proxy components. Each existing CCM component development generator that is available with open source is a candidate for the proxy generation, too. The QoS Enabled Distributed Object (QEDO) [13] component generation environment has been selected to automatically produce CCM conformant proxy components.

According to the functionalities of the QEDO toolset the generator delivers both the C++ sources of the proxy component business logic (executor) and the corresponding container serving code (CCM servant). The generation of the required dll libraries is subject of the user's application of makefiles or software development

tools. Furthermore the generator produces the related software package and component descriptors (.csd and .ccd). An overview on the proxy generation and the generator usage is given in the following figure².

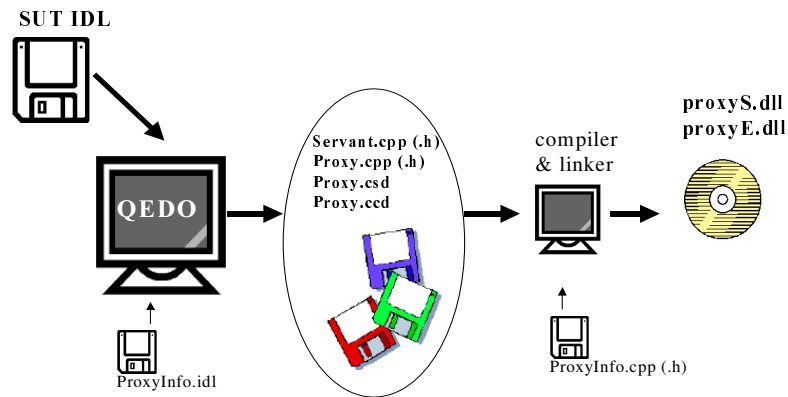


Figure 2: Proxy generation scenario

QEDO [13] technology has been selected due to its tool architecture that is based on a common repository for IDL specifications in combination with a front-end IDL-3 parser and an extensible set of backends for the generation of the CCM component servants and skeletons. An economic approach is to reuse and extend the QEDO infrastructure. Two steps have been identified to produce the proxy components:

1. The IDL definitions of the SUT components are used to derive the IDL definition of the proxy components. The resulting IDL definitions will be stored in the repository, too.
2. Based on the IDL definitions of the proxy components the component implementation, i.e. servants and business logic will be generated. This step is similar to the development of “normal” (user-defined) components produced by QEDO. The difference is the additional automatic inclusion of the “proxy” business logic in the generated skeletons.

As shown in the figure 3 for each step a separate QEDO backend is implemented.

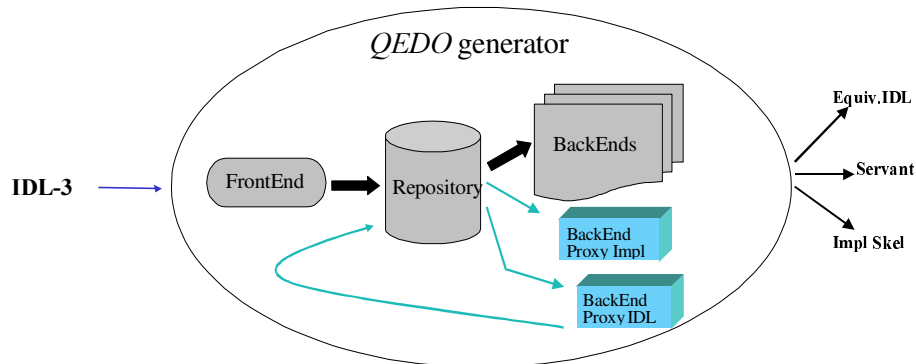


Figure 3: Backend Proxy generation

² Note: ProxyInfo.idl (.cpp, .h) files address the predefined implementation of proxy information events and do not have to be developed by the user of the proxy generator.

It is sufficient to parameterize the proxy generator with the filename of the IDL definitions of the SUT and a list of component names for which a proxy should be generated. Other parameters are due to the QEDO user specification.

3 ADAPTATION ISSUES FOR TTCN-3

According to our requirements for test descriptions there is a need to apply e.g. a programming language independent, i.e. abstract, test notation for the definition of the test cases. Standardized test notations like TTCN-3 or the testing profile for UML [12] appears as suitable candidates. TTCN-3 has been applied in the context of CORBA for testing both synchronous and asynchronous communication [15][18]. The UML testing profile is another alternative test notation but due to the delayed standardization process of UML2.0 within the OMG no tool support was available at the start of our work.

In our CCM testing environment the TTCN-3 technology will be used to develop and implement test scenarios running against CCM applications. The tests address the interface operations of the SUT components, which have been defined with IDL-3. Such earlier implementation approaches have been already reported [9]. We've continued the work w.r.t. the proxy-based testing approach and test specification issues.

Most of the required TTCN-3 type definitions arise from their corresponding SUT definitions that are provided in IDL-3. This holds for IDL data structures, exceptions, interfaces, and operation definitions. ETSI has specified the detailed mapping rules for IDL version 2.x to TTCN-3. Currently this ETSI standard has a draft status [4]. Beyond the scope of the mapping rules from IDL to TTCN-3 some new IDL-3 language elements have to be considered, too. Especially an event type definition has to be mapped. Since the IDL-3 event types are related to IDL value types a mapping to TTCN-3 records is proposed. Furthermore a TTCN-3 port type using message-based communication for exchange of the events of the particular types is required. Similar to IDL interface types all definitions related to an event type are stored in a TTCN-3 *group* definition, i.e. data templates related to the event type can be collected in the group, too. Please see Listing 1 for a simple example on the eventtype mapping,

```
//IDL3
enum PhilosopherStatus { DEAD, STARVING, HUNGRY, THINKING, EATING};
eventtype PhilosopherState {
    public PhilosopherStatus status;
};

// TTCN-3
group PhilosopherStateEvent
{ type enumerated PhilosopherStatus {DEAD, STARVING, HUNGRY, THINKING, EATING};
  type record PhilosopherState {PhilosopherStatus status};
  template PhilosopherState PhilosopherStateTemplate (PhilosopherStatus philstat) := {status := philstat};
  type port PhilosopherStatePort message {in PhilosopherState};
}
```

Listing 1: IDL3 eventtype use in TTCN-3

The execution of a TTCN-3 test case requires some knowledge on the structure of a TTCN-3 test system. A standardized TTCN-3 test specification has to be compiled to get a TTCN-3 executable. The test developer should add a test adapter (TA) and an Encoder/Decoder to the executables to run the tests. The TA implements the interface to the SUT, it has to conform to the standardized Test Control and Test Runtime Interfaces (TCI and TRI) [6] [7] to be used with different TTCN-3 runtime environments. The Encoder/Decoder supports the mapping between the implementation language and the TTCN-3 data types according to the standardized TCI.

The TA has to fill the technological gap between the test executable and the CCM based SUT components. In some test cases the TA has to fulfil the component roles of an interface (facets) provider or consumer and needs to exchange events asynchronously with the SUT. A facility to abridge this gap is the introduction of an *Adaptation* component that supports the TA to fit to the CCM communication requirements.

The Adaptation component is a regular (deployable) CCM component but also part of the test environment. It supports the access to/from the SUT and the test system. Furthermore it assists the TA in case of an event-based communication. For the latter case a CORBA-based communication (collector interface) is to be implemented to handle the events. The exchange of CCM events by the TA using the adaptation component is illustrated in the figure 4.

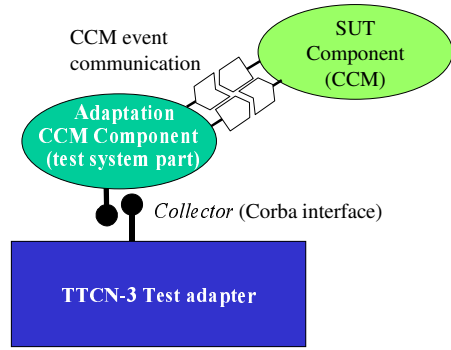


Figure 4: CCM Adaptation component for the TTCN3 Test Adapter

A TTCN-3 test manager and test runtime system (e.g. [16]) that is conforming to the TRI and TCI ETSI standards is needed for the execution of the tests.

We have to distinguish the different test configurations and test behavior scenarios in order to define the required facilities for the deployment process of the SUT and the TTCN-3 test system:

In the simplest case we consider a SUT providing a stand-alone service that is waiting for requests. Test drivers have the role of SUT clients and can get the SUT references using e.g. the CORBA naming service. This case allows the deployment of the SUT and the test system proxies independently from the active test system. The active TTCN-3 test system can be started later.

A combined deployment of the SUT and test system is needed if there are several relationships between the test system and SUT. This case holds if e.g. the test system emulates the SUT partly, e.g. the test system emulates some facets or event sources for the SUT that are needed for the SUT operation. Furthermore it is also needed if the components of the SUT that take part in the test scenario are “hidden” in the SUT i.e. if their references are available during the deployment procedure only.

If there is a need to exchange the interface references between the test system and the SUT in advance a CCM component (called “keeper”) is introduced that will be connected during the deployment procedure instead of the overall test system [10]. If the adaptation component is available the methods described for the keeper could be implemented in one component, i.e. the adaptation component takes over the functions of the test system keeper, too.

4 ONLINE TEST SCENARIOS

A particular application of the proxy-based testing approach is the so-called Online Testing of a running system. Figure 5 illustrates the principle configuration of such monitoring system that is connected to applications like a TTCN-3 tester.

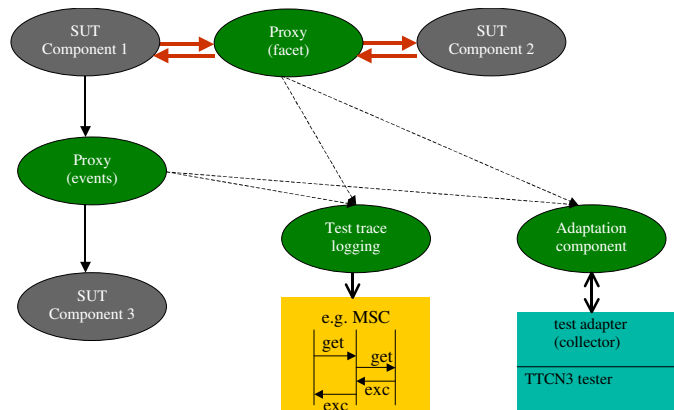


Figure 5: Sample Online Test Configuration

In the following listing 2 a section of a sample TTCN-3 behaviour specification of an online test case is given. The test configuration includes two proxy components which monitor the access of a *Philosopher* component at two *ForkManager* components³. The regular (*obtainfork*, *releasefork*) and exceptional (*inuse*) reactions of the target component will be observed and saved in the *fork1* and *fork2* variables. This knowledge will be used to validate state events (see listing 1) from the *Philosopher*, e.g. an *EATING* state should be observed if the *Philosopher* has got both forks in advance. If the state event does not conform to the observed receipt of forks the test stops with a “fail” verdict. The validation phase duration has been limited to 50 observed state events. After this period a “pass” verdict is assigned to the test case execution.

```

log("+++++++> START of Online test regular period.");
do
{alt
  [] ProxyInfoPCO[1].receive(inuse) {}
  [] ProxyInfoPCO[2].receive(inuse) {};
  [] ProxyInfoPCO[1].receive(obtainfork) {fork1:=true;}
  [] ProxyInfoPCO[2].receive(obtainfork) {fork2:=true;};
  [] ProxyInfoPCO[1].receive(releasefork) {fork1:=false;};
  [] ProxyInfoPCO[2].receive(releasefork) {fork2:=false;};

  [] PhilosopherStatePCO.receive(PhilosopherStateTemplate(EATING))
    {if(fork1 and fork2)
      {log("INFO: eating with 2 forks - test continues!"); test:= test+1 }
     else { verdict.set(fail); stop}}
  [] PhilosopherStatePCO.receive(PhilosopherState:?)
    {if(fork1 and fork2)
      {verdict.set(fail); stop}
     else {log("INFO: not eating & not 2 forks - test continues!"); test:= test+1}
    } }
} while(test < 50);
log("+++++++> END of Online test regular period.");
verdict.set(pass);

```

Listing 2: Sample TTCN-3 Online Test behaviour

In addition to the use of the TTCN-3 test manager we have applied a TraceServer and –viewer which have been implemented by Lucent within the COACH project. A snapshot of this application is presented in figure 6. Similar to Message Sequence Charts the SUT components are represented by vertical lines, their interactions observed by the test system proxies are corresponding to arrows between the component instances.

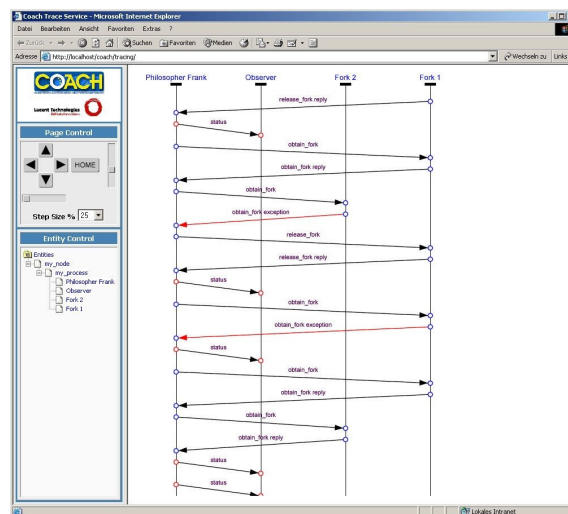


Figure 6: Graphical presentation of the Proxy observations

³ Note: The corresponding SUT has been proposed by the OMG in the ‘Dining Philosophers’ example.

5 CONCLUSIONS

From the software testing viewpoint the introduction of large CORBA component-based software systems is two-sided. First it demands for appropriate test concepts and techniques, second it allows the test developers to benefit from the component technology itself.

In our work we've introduced a proxy-based testing approach for a component-based system that provides a generation of test components and supports the application of various test systems for both passive online monitoring but also active (sub)system testing. Details on the proxy information types that are published by the proxy components and the automatic generation of the proxy components have been outlined. Furthermore the application of testing technologies like TTCN-3 for a particular SUT component technology has been investigated. Due to our practical experiments we found that TTCN-3 is suitable for proxy-based testing of components but needs extra considerations for adaptation.

In the next step we see the possibility to implement so-called *active* proxy components, i.e. proxies that do get instructions from other test system parts or a test operator in order to block or manipulate all or selected operations under test that are exchanged between components of the SUT (grey-box testing approach).

6 REFERENCES

- [1] M. Barnett et al.: Spying on Components: A Runtime Verification Technique. SAVCBS Workshop 2001, <<http://www.cs.iastate.edu/~leavens/SAVCBS/>>
- [2] A. Bohr: Software Component Testing Strategies, Dept. of Information and Computer Science, University of California, Irvine, Technical Report UCI-ICS-02-06, June 2001
- [3] M. Born et al.: The European CORBA Components Open Source Initiative. ERCIM News No. 55, October 2003. http://www.ercim.org/publication/Ercim_News/enw55/
- [4] ETSI DTS/MTS-00080: Methods for Testing and Specification (MTS): The IDL to TTCN-3 Mapping, V1.1.0, 2003-02.
- [5] ETSI: Testing and Test Control Notation (TTCN3). <<http://www.etsi.org/ptcc/ptcctcn3.htm>>
- [6] ETSI: TTCN-3 Runtime Interface (TRI). ES 201 873-5, Feb. 2003.
- [7] ETSI: TTCN-3 Control Interface (TCI). ES 201 873-6, May 2003.
- [8] J. Grabowski et al.: An Introduction into the Testing and Test Control Notation (TTCN-3). To appear in Computer Networks, 2003.
- [9] A. Hoffmann et al.: CCM testing environment. ICSSEA'2002, Dec. 2002.
- [10] A. Hoffmann et al.: The COACH approach to test and monitor CCM applications. STEP workshop, Amsterdam, Sep. 2003.
- [11] Object Management Group: CORBA Components (formal/02-12-06)
- [12] OMG ADTF. UML testing profile. <<http://www.fokus.fhg.de/tip/u2tp/>>
- [13] The QEDO project <<http://qedo.berlios.de/>>
- [14] W. Romijn (Ed.): Specification of the component test environment, Project deliverable D2.7, COACH consortium, June 2003.
- [15] M. Schünemann et al.: Improving test software using TTCN3, GMD Report No. 153, Dec. 2001. <<http://www.gmd.de/publications/report/0153/>>
- [16] Testing Technology: TTCN-3 products <http://www.testingtech.de>
- [17] E. Weyuker: Testing Component-Based Software – A Cautionary Tale: IEEE Software, pages 54-59, Sep./Oct. 1998
- [18] A. Yin et al.: Operation-based interface testing on different abstraction levels, ICSSEA'01, Paris, Dec.2001.