2008 International Conference on Software Testing, Verification, and Validation

Testing Grid Application Workflows Using TTCN-3

Thomas Rings, Helmut Neukirchen, Jens Grabowski
Software Engineering for Distributed Systems Group,
Institute for Computer Science, University of Göttingen,
Lotzestr. 16–18, 37083 Göttingen, Germany.
{rings,neukirchen,grabowski}@cs.uni-goettingen.de

Abstract

The collective and coordinated usage of distributed resources for problem solution within dynamic virtual organizations can be realized with the Grid computing technology. For distributing and solving a task, a Grid application involves a complex workflow of dividing a task into smaller sub-tasks, scheduling and submitting jobs for solving those sub-tasks, and eventually collecting and combining the results of the sub-tasks into a final result. The quality assurance of Grid applications is a challenge due to the highly distributed nature of the Grid environment in which the Grid application is deployed. This paper investigates the applicability of the Testing and Test Control Notation *(TTCN-3) for testing the workflows of distributed Grid applications. To this aim, a case study has been created that consists of a distributed Grid application which includes a typical Grid application workflow; as the main contribution, this case study contains a corresponding distributed TTCN-3 test suite that tests the correct execution of the Grid application workflow. To demonstrate the adaptation of the abstract TTCN-3 test suite to a specific Grid environment, corresponding reusable test adapters have been implemented for the Grid middleware* Globus Toolkit 4 *(GT4). The realized test system demonstrates that TTCN-3 is applicable for testing the workflow of distributed Grid applications.*

1. Introduction

In the last years, supercomputing experienced a change: expensive super computer hardware is replaced by cheaper distributed computer Grids. This trend has manifold reasons: on the one hand, today's demand for super computing resources grows faster than Moore's law and can thus only be satisfied using distributed computing; on the other hand processor cycles from idling computers can be cost-efficiently scavenged using Grid technology. Finally, Grid computing offers the promise to make computing, as well as, access to data and remote equipment as easy as the use of electricity from the electrical power grid. Like in many other domains, this trend is accompanied with the problem that the corresponding software becomes more complex and their development more error-prone. Surprisingly, quality assurance for Grid computing software is not very mature and only a few proprietary test approaches exist. This paper investigates the applicability of the standardized *Testing and Test Control Notation* (TTCN-3) for testing in Grid environments. While testing of Grid middleware is comparable to message-based protocol testing using the *Conformance Testing Methodology and Framework* (CTMF) standard, we focus on functional testing of distributed Grid applications that make use of higher-level communication mechanisms.

This paper is structured as follows: In Section 2, we give an overview about the foundations of Grid computing, conformance testing, and TTCN-3. Afterwards, in Section 3, we describe the Grid application used in this paper. In Section 4, we focus on the TTCN-3 case study that is used to test the Grid application introduced in Section 3. In Section 5, we provide a comparison with related work. Finally, we conclude with a summary and outlook in Section 6.

2. Foundations

In the following, we provide a brief overview on Grid computing, on the *Conformance Testing Methodology and Framework* (CTMF) that we adopt for distributed functional testing, and on the test language TTCN-3 that we apply for the specification and implementation of our tests.

2.1. Grid computing

For performing resource intensive and complex tasks, a Grid computing environment provides an efficient sharing and management of computing resources [2], which are not administered centrally, in order to achieve a non-trivial quality of service [6]. To overcome the high degree of heterogeneity of the underlying resources, a Grid

978-0-7695-3127-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ICST.2008.24

computing environment uses open, standard, general purpose protocols and interfaces as middleware. Particularly, a standardization of Grid computing allows a dynamic negotiation and management of shared resources between any interested parties of a virtual organization [6].

The architecture of a Grid system can be described in terms of layers as depicted in Figure 1. The lowest layer, the *Fabric* layer, provides a basis as a common interface for all kinds of physical devices or resources that can include computers, networks, and storage systems [5, 8].

The *Connectivity* layer is located above the Fabric layer and defines the core communication and authentication protocols, which are required by the Grid. The delegation of authorizations and methods for a single sign-on are essential for this layer [5, 8].

In the *Resource* layer, which is above the Connectivity layer, the common access to several resources is organized in order to enable secure initiation, monitoring, and control of resource-sharing operations, such as assignment or reservation [5, 8].

The role of the *Collective* layer that is above the Resource layer is the coordination between different resources. Responsibilities of the Collective layer include directory and brokering services for discovery, allocation, monitoring, and diagnostic of resources [5, 8].

At the top of any Grid system is the *Application* layer with the user applications. These make use of the services provided by the lower layers that allow to access resources transparently [8].

The algorithm of an application that is supposed to run in a Grid computing environment should be parallelizable to tap the full potential of the advantages of Grid computing, i.e., the main task is divided into smaller sub-tasks in order to allow parallel computation. Hence, a Grid application is partitioned into independently running sub-applications.

The usual approach for creating a Grid application is to solve each sub-task by a sub-application, i.e., an executable program. A node of the Grid, on which that sub-application is installed, may then execute that sub-application as a job. It is very common that an already existing command-line application shall be integrated into a Grid ("gridification"). As long as the already existing command-line application can be requested by the appropriate parameters to solve not the whole problem, but only a part of the problem, this command-line application can be integrated without any changes as sub-application. Since the existing command-line applications write in most cases their output into a file, this file can than be transferred via a network file transfer service between the different nodes of the Grid as required.

To control the workflow of dividing a task into smaller sub-tasks, scheduling and submitting jobs for solving those sub-tasks, monitoring the execution of the jobs, and eventually collecting and combining the results of the sub-tasks into a final result, a separate application management is required. This application management provides an interface to the Grid user and thus hides the complexity of the underlying Grid technology.

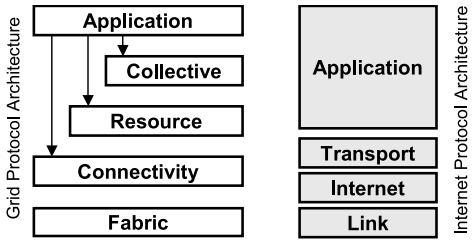

Figure 1. Layered Grid architecture mapped with the Internet protocol architecture [8]

It depends on the services that are actually provided by the layers of a Grid middleware implementation, which of the above steps actually need to be implemented by the application management or whether the application management may delegate them to a lower layer. If, for example, a Grid environment offers a full-fledged Collective layer, the application management may rely on workflow management services from that layer. If, for example, a Grid environment offers only minimal services in its layers and thus lacks scheduling, software discovery and brokering services, the application management may even be responsible for assigning and submitting its jobs to appropriate nodes of the Grid.

The application management and the related sub-applications form a massively distributed application. While the Grid middleware itself may use fine-grained message- or procedure-based communication mechanisms, a Grid application uses coarse-grained communication mechanisms, namely remote job submission and remote file transfer. Thus it is investigated in the remainder of this paper whether known distributed testing methods and technologies apply for testing Grid applications as well.

In the case that the sub-applications of a Grid application are existing command-line applications, it can be assumed that these applications are already adequately tested. However, the application management of a Grid application has usually been developed from scratch and thus requires thorough testing.

2.2. Conformance Testing Methodology and Framework

For the functional black-box testing of distributed Grid applications, we adopt concepts of the international ISO/IEC multipart standard 9646 *OSI Conformance Testing Methodology and Framework* (CTMF) [13]. CTMF defines a comprehensive procedure for the conformance testing of

Open Systems Interconnection (OSI) protocol implementations. The entire standard consists of seven parts and covers the following aspects: concepts (Part 1), test suite specification and test system architectures (Part 2), test notation (Part 3), test realization (Part 4), means of testing, and organizational aspects (parts 5–7).

Even though CTMF provides concepts and means that are specific to OSI protocol testing, CTMF has been successfully applied for testing other kinds of distributed systems like for example, ISDN- and GSM-based systems. Thus, we decided to adopt concepts from CTMF as well for functional testing of distributed Grid applications. However, only the test suite production procedures and test architecture from Part 2 of CTMF were applicable for our work, because the other parts were too OSI or conformance testing specific.

The CTMF procedure for producing test suites starts with the identification of the requirements to be tested. The requirements are used to define test groups necessary to achieve an appropriate coverage of the requirements. The objective of each test group is described by means of an informal test objective. For each test group, a set of test purposes is developed. A test purpose is an informal or semi-formal description of a test case. Afterwards, an appropriate test architecture is selected and for each test purpose, a test case is implemented by means of a test language.

The CTMF test methods are abstract, i.e., implementation independent, test architectures. They define *Implementation Under Test* (IUT), *System Under Test* (SUT), the *Points of Control and Observation* (PCOs) that are needed to control and observe the SUT, and the connections between these constituents.

Figure 2 presents the CTMF *Multi-party test method*, which we used to test the application management. The IUT is controlled and observed by one *Upper Tester* (UT) and several *Lower Tester*s (LTs). The underlying service and the IUT constitute the SUT. CTMF assumes a layered architecture: the LTs communicate with the IUT by using an underlying communication service. It is assumed that this service provider has already been adequately tested. Conceptually, the UT plays the role of a service user of the IUT and LTs play the role of peer entities that realize together with the IUT the service used by the UT. The PCOs are the standardized interfaces used by UT and LTs to communicate with the SUT. UT and LTs communicate by means of *Test Coordination Procedure*s (TCPs). To avoid probe effects, the underlying communication service should not be used for the implementation of the TCPs.

For the functional testing of Grid applications, we followed Part 2 of CTMF to produce test suites and adapted the standardized test methods to the Grid requirements. As part of this adaption, we do not distinguish between UT and LT, but use the more general term *test component* instead.

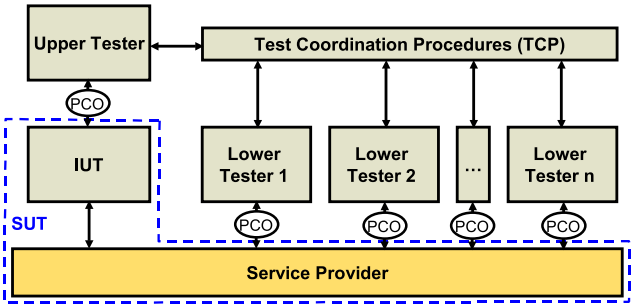

Figure 2. CTMF Multi-party test method

2.3. TTCN-3

The *Testing and Test Control Notation* (TTCN-3) [4, 11] is a test specification and test implementation language standardized by the *European Telecommunications Standards Institute* (ETSI) and the *International Telecommunication Union* (ITU). While TTCN-3 has its roots in functional black-box testing of telecommunication systems, it is nowadays also used in other domains such as Internet protocols, automotive, aerospace, or service-oriented architectures. TTCN-3 can be used not only for specifying and implementing functional tests, but also for scalability, robustness, or stress tests.

The TTCN-3 language has the look and feel of a typical general purpose programming language, i.e., it is based on a textual syntax. Most of the concepts of general purpose programming languages can be found in TTCN-3 as well, e.g. data types, variables, functions, parameters, loops, conditional statements, and import mechanisms. In addition, test related concepts are available to ease the specification of test suites.

TTCN-3 supports distributed testing through the notion of test components: in addition to the *Main Test Component* (MTC), *Parallel Test Component*s (PTCs) can be created dynamically. Each test component runs concurrently and may therefore execute test behavior in parallel to other test components. Test components can be connected to each other or to the SUT via *ports*. These concepts allow to create powerful distributed test architectures, for example instantiations of the CTMF test methods.

For the communication between test components and with the SUT, operation such as **send** and **receive** (TTCN-3 keywords are printed in bold typeface) can be used to transfer messages via ports. The values of these message are specified using *templates*. TTCN-3 templates may involve wildcards and thus provide a powerful matching mechanism to check whether expected test data has been received or not.

Further concepts that ease test specification are: test verdict handling, logging, timers, and *defaults*. Defaults are typically used for specifying alternative behavior that deals

with unexpected events. Since a **receive** operation blocks until it observes a message that matches the specified template, defaults can be activated to catch, e.g. the expiration of a timer or any unexpected message.

To allow the automated execution of TTCN-3 test suites, TTCN-3 tools can be used to compile TTCN-3 test specifications into executable tests. However, TTCN-3 test specifications use abstract communication mechanisms. Thus, to make TTCN-3 test specifications executable, an adaptation layer is required. Hence, a *System Adapter* (SA) entity that implements operations of the *TTCN-3 Runtime Interface* (TRI) [4] and a *Coding/Decoding* (CD) entity that implements operations of the *TTCN-3 Control Interface* (TCI) [4, 18] must be additionally realized.

For those ports that are mapped to PCOs, the SA realizes **send** and **receive** operations by using the communication mechanisms of the SUT, e.g. sockets. The CD is responsible for the translation between the abstract TTCN-3 values and the concrete bit-level data encoding used by the SUT.

Using TTCN-3 has several advantages in comparison to proprietary test languages or low-level test implementation. The high abstraction level speeds up test development. Furthermore, the re-usability is higher, because both, the abstract TTCN-3 test specifications and the adapters, can be re-used independently from each other. Finally, due to the fact that TTCN-3 is standardized and various TTCN-3 tools are available, a vendor dependence is avoided.

3. Grid application

For investigating the applicability of TTCN-3 for testing the workflow of Grid applications, we developed a Grid application that consists of an application management and corresponding calculation sub-applications.

The application that is transferred into a Grid application solves the task of calculating the Mandelbrot set [15] or a subset, respectively. This task is parallelizable, because each element of the Mandelbrot set can be calculated independently from each other. A corresponding non-interactive Mandelbrot set *Calculation Application* has been implemented: it uses command-line parameters that specify as input the subset of the Mandelbrot set that shall be calculated and provides as output a file that contains a color graphic of the calculated Mandelbrot subset.

For the gridification of this application, an *Application Management* and the actual Calculation Application have to be integrated into the Grid environment. To ensure that we are able to investigate also the case, where a Grid environment does not provide services of the higher Grid layers, the Application Management realizes on its own services of the Collective layer such as job scheduling and assigning jobs to the nodes of the Grid.

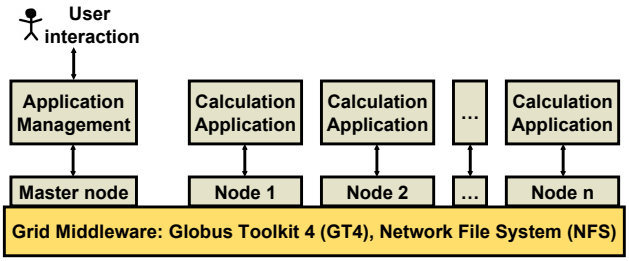

Figure 3. Application integration into the Grid environment

3.1. The Grid environment

The Grid environment that has been applied for hosting the implemented Grid application is the Instant-Grid [12] environment that is a project funded by the e-Science Initiative from the German Federal Ministry of Education and Research. Instant-Grid provides a complete computer Grid environment based on a Knoppix [14] Linux live CD-ROM, which includes all required services for establishing a Grid with Intel x86 computers. It includes an automated configuration mechanism that deploys without any user interaction Instant-Grid on computers that share a *Local Area Network* (LAN).

Instant-Grid uses the software services and libraries offered by the *Globus Toolkit 4* (GT4) Grid middleware [7, 10] that is developed by the Globus Alliance. GT4 implements the Resource and Connectivity layers of the Grid protocol architecture. The higher Collective and Application layers are not provided by GT4, but by Instant-Grid and its user applications.

Additional to these services, Instant-Grid offers a *Network File System* (NFS) for collective data access by all nodes of the Grid. Alternatively, it is possible to use the GT4 GridFTP service for data transfer.

3.2. Integration into the Grid

The integration of the Mandelbrot set Calculation Application and its Application Management into Instant-Grid is depicted in Figure 3. It has been integrated as a Grid application into the Instant-Grid CD-ROM in order to make it available on the file system of each Grid node.

After the user has sent a request, the Application Management divides this request into sub-tasks and submits them as jobs to GT4 in order to execute the Calculation Application on an idle node. For the submission of a job, the GT4 service globusrun-ws is used. This call requires as parameters the name of a node, the name of the executable to be started, its file system location, and command-line parameters.

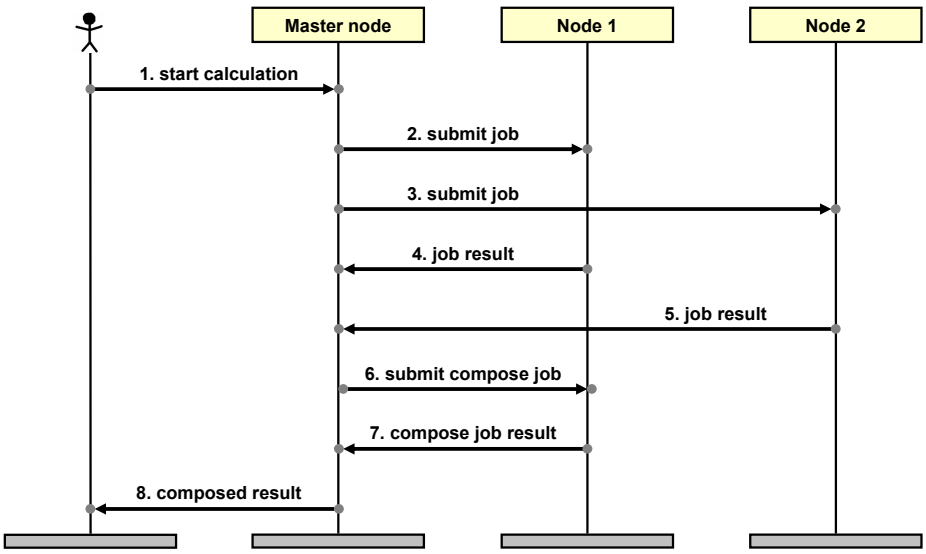

Figure 4. Grid application scenario

When a Calculation Application sub-task finishes, it creates a result file in the NFS. Once all Calculation Application sub-tasks are completed, the Application Management initiates on one node of the Grid the submission of a final job for the composition of the other sub-task's results. This final composition sub-task uses the NFS to read in the graphic files created by the other sub-tasks and to write back the composed result graphic files. The composed result is read by the Application Management via NFS and presented to the user through a web interface.

The above interactions are visualized by the sequence diagram in Figure 4. In this Grid application scenario, the Grid computing environment consists of a Master node, on which the Application Management is running, and two further nodes, on which the sub-tasks are executed. In this scenario, the user starts the calculation through the web interface of the Application Management (Message 1). The Application Management splits this request into two calculation sub-tasks: the first task is submitted to Node 1 and the second to Node 2 (messages 2 and 3). After their completion (messages 4 and 5), the final job that composes both results is assigned to Node 1 (Message 6) and its output (Message 7) is presented to the user via a web interface by the Application Management on the Master node (Message 8).

The Application Management contains its own scheduler, which ensures that on each node only one job is running at the same time. A sophisticated synchronization of the tasks is not required in that case, since every task is independent from each other. The only restriction is that the final compose job has to be executed after all the other subtasks have been finished, because it is dependent on the result of all previous jobs.

4. TTCN-3 Grid application testing case study

To investigate the applicability of TTCN-3 for testing Grid applications, the previously described Mandelbrot Grid application is used. Our testing approach follows the CTMF, i.e., we start with creating test purposes, we use the Local and Multi-party test methods as test architectures, we assume that the underlying layers (such as the GT4 Grid middleware) have already been tested and can thus be regarded as correct, and we perform only functional black-box tests, but no non-functional tests.

The Mandelbrot Grid application consists of two different types of items that need to be tested: the sub-applications, such as the non-interactive command line Calculation Application, and the Application Management that is responsible for the correct workflow. As discussed in Section 2.1, it can be assumed that the Calculation Application is already adequately tested. Therefore, we focus on testing the Application Management and refer the reader to further work [17] where we describe how we test the Calculation Application using an adaptation of the Local test method of CTMF.

The descriptions of a related test purpose, test architecture and distributed test behavior, TTCN-3 test specification, test adapters, and test execution are provided in the following subsections.

4.1. Test purpose

The test purpose that is pursued in the following is to assess the correctness of the workflow that is created by the SUT, i.e., the Application Management. We refer to this test purpose as *Testing the Application Management* (TAM).

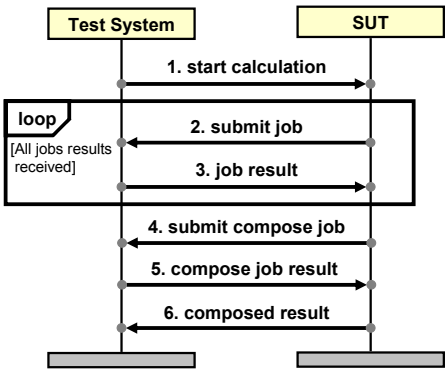

Figure 5. Test purpose TAM illustrated as a sequence diagram

For testing the correctness of the workflow, the job submission order and the parameters that are passed to the jobs by the SUT have to be checked.

The message exchange of the TAM test purpose is based on the Grid application scenario in Figure 4. In addition to messages that relate to the actual job submission, the test purpose includes also the start of the calculation via the Web-based user interface as an initial stimulus and the presentation of the final composed result to the user.

The corresponding graphical representation of the TAM test purpose is depicted in the sequence diagram in Figure 5. Initially, the test system sends a message for stimulating the SUT (Message 1). This stimulus includes the data that is supposed to be entered by the user. The SUT uses this data in order to determine the parameters for the sub-tasks. After this determination, the SUT assigns the tasks to the nodes of the Grid by submitting jobs via the Grid middleware. The test system has to observe these job submissions (including their parameters) to decide whether the SUT divided the overall task correctly into sub-tasks (Message 2). Since the SUT expects that each job returns a file as result, the test system has to provide them to the SUT (Message 3). The number of jobs that are submitted by the SUT depends on the parameters that are passed in the stimulating start calculation message. Hence, the sequence diagram for the test purpose TAM contains a loop that iterates until the expected number of job submissions has been observed by the test system. After the correct number of calculation sub-tasks has been performed, it has to be tested if the final composition job has been submitted with the correct parameters (Message 4). Afterwards, the test system provides the overall result as file to the SUT (Message 5). Finally, it is tested that the SUT forwards this composed result correctly to the user interface (Message 6).

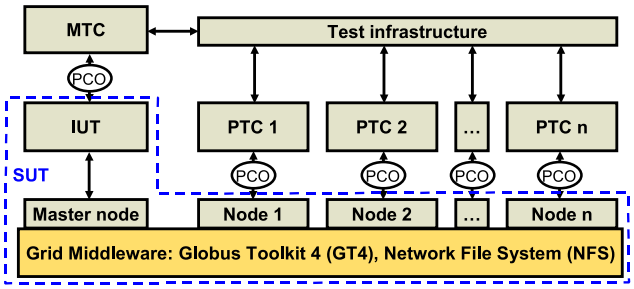

Figure 6. Test architecture for test purpose TAM

4.2. Test architecture and distribution of test behavior

To turn the TAM test purpose into a test case, a suitable test architecture needs to be selected and the test behavior needs to be assigned to the different test components of the test architecture. Because the job submissions that are performed by the SUT may result in the execution of processes on nodes that are distributed throughout the Grid, the Multiparty test method of CTMF is a suitable test architecture. Thus, the SUT comprises not only the Application Management that is actually the IUT but also the lower layers of the Grid middleware.

Figure 6 shows the test architecture that is used for testing the TAM test purpose. This figure is related to Figure 3 and shows the integration of the test system into the Grid environment: the human user is substituted by the *Main Test Component* (MTC), i.e., the MTC stimulates the IUT via an HTTP-based interface and observes the final composed result. For observing that the IUT submits jobs correctly to the nodes of the Grid environment, a *Parallel Test Component* (PTC) is required on each node. These PTCs serve as stubs for the Calculation Application and are thus able to intercept the parameters of the job submission. Since these PTCs replace the Calculation Application, they must also be able to provide a file as job result. The same considerations apply for the final compose job sub-application that is replaced as well by a stub.

It is not possible to predict to which node the scheduler inside the IUT submits a job. This may depend on the current load in the Grid and on differing computing power of the nodes. Hence, in every test run, different nodes may be selected by the IUT. Therefore, a PTC cannot decide using its local knowledge only, whether observed job submissions are part of a correct workflow or not. Instead, global knowledge is required to be able to decide whether for each subtask a corresponding job has been submitted. To this aim, each PTC forwards the observed job submission parameters to the MTC, thus providing the MTC with global knowledge that is required to set a test verdict. To support these test co-

Listing 1. Excerpt of the TTCN-3 test behavior for test purpose TAM

```
1  ...
2  // send the stimulus
3  pt_http .send(a_httpStimulus(HORIZONTAL, VERTICAL, v_workingpath));
4
5  // check received job parameters messages
6  while(i<HORIZONTAL*VERTICAL) {
7    pt_PTCParameters.receive(
8        a_Reaction( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?))
9        -> value v_Parameters {
10     v_check := false ;
11     for (j:=0; j<HORIZONTAL*VERTICAL; j:=j+1) {
12       // check if the received message is in the array
13       if (v_Parameters == JobArray[j]) {
14         log("matched ReactionType");
15         checkArray[j] := true;
16         v_check := true;
17         i:=i+1;
18       }
19     }
20     // if the message does not match, testcase failed
21     if (v_check == false) {
22       log("mismatch");
23       setverdict ( fail );
24     }
25   }
26 }
27 ...
```

ordination procedures between the PTCs and the MTC, the test architecture provides also the corresponding communication channels ("test infrastructure" in Figure 6).

4.3. TTCN-3 test specification

The actual test suite for testing the Grid application is specified using TTCN-3. It contains definitions of types, test data templates, several test components as well as test behavior. The behavior of a TTCN-3 test case for the test purpose TAM is described in the following. The test case itself is parameterized to support a stimulation of the IUT with different job sizes and to be able to specify the number of nodes in the Grid environment, i.e. how many test components are to be created in order to observe all relevant nodes of the Grid.

The MTC is responsible for dynamically creating the PTCs of the distributed test architecture, connecting the test components to each other, and starting the behavior on the several PTCs. Afterwards, the actual behavior that relates to the test purpose TAM is performed. Listing 1 shows the corresponding excerpt of the test case that runs on the MTC.

First, the MTC sends the stimulus to the SUT (Line 3 in Listing 1). This corresponds to Message 1 in Figure 5. The data used for the stimulation is specified by the parameterized TTCN-3 template a_httpStimulus (Line 3 in Listing 1). It includes the desired resolution of the final output graphic and the file system working path, where the IUT places the output graphic. According to the application's specification, the desired resolution of the final output graphic influences directly the number of jobs that are submitted by the IUT. The actual parameter values for the parameterized template are based on test case parameters and can thus be easily changed to obtain test cases with different test data.

The **while** loop in lines 6 to 26 is used collect the observed job parameter values that are forwarded by the PTCs to the MTC. Since the order, in which the forwarded jobs submission messages are received, cannot be predicted, a sort of tally sheet is used. All the expected job parameters are predefined and stored in the array JobArray – the predefinition of this array is not shown in the excerpt. Once an arbitrary job submission message is received (the ? wildcards in the template used in lines 7–8 matches any job submission message), the contained jobs parameters are retrieved and stored into the variable v_Parameters (Line 9). The **for** loop in lines 11–19 checks whether the received job parameters correspond to one of the predefined job parameters from JobArray. If true, this is marked in the Boolean array checkArray (Line 15). If false, the **for** loop will eventually terminate unsuccessfully and the test verdict will be set to **fail** (Line 23). Subsequently (not shown in the excerpt anymore), it is checked that no further job submissions are received and that all jobs in the checkArray have been marked, i.e., all of the expected job submissions have been observed. All of the above described test behavior relates to how the MTC implements the test coordination procedures that are associated to Message 2 in Figure 5.

Since a distributed test architecture is used, the PTCs execute behavior as well: once a Calculation Application job submission is observed at a PTC (Message 2 in Figure 5), the job parameters are forwarded to the MTC as part of the associated test coordination procedures. After a configurable amount of time, the PTC creates a graphic file as result (Message 3 in Figure 5).

The remainder of the TTCN-3 test case contains behavior that relates to messages 4–6 of the test purpose depicted in Figure 5. Furthermore, to prevent blocking in the case that no message arrive and to be able to detect when unexpected messages are observed, the TTCN-3 concept of *defaults* is used: such a default is activated at the beginning of the test case and a timer is started. In case that the default catches a timeout or a wrong message, the **fail** test verdict is automatically set by the behavior of the default.

4.4. Adaptation layer

The TTCN-3 test suite that has been described in the previous subsection, abstracts from implementation details of the Grid middleware. Hence, to allow an automated execution of the abstract TTCN-3 test suite, an adaptation layer is required. This adaptation layer consists of the *Coding/Decoding* (CD) and the *System Adapter* (SA) entities. The CD

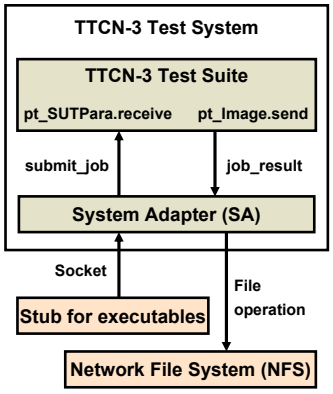

Figure 7. System Adapter

translates between the concrete data representation of the Grid environment and the abstract TTCN-3 data types and values. The SA maps the TTCN-3 **send** and **receive** operations to concrete operations of the Grid environment. In our case study, the adaptation has been realized for the InstantGrid environment, e.g. by mapping **send** and **receive** to the creation of result files in the NFS or the observation of job submissions via GT4 respectively.

Figure 7 illustrates how the PTCs, their SAs, and the Grid environment interact: to be able to intercept a GT4 job submission (i.e., a call to a command-line executable by the SUT), the command-line executable is replaced by an executable stub that forwards its command-line parameters via a socket to the SA. The PTC is then able to retrieve this message from the SA via a **receive** operation. For creating a file with the resulting graphic, the PTC sends a message to a certain port. Inside the SA, this port has been implemented to write a file to the NFS.

All other adapters, e.g. those for the communication between MTC and PTCs are provided by the TTCN-3 test infrastructure. In our case study, we use the TTCN-3 development environment TTworkbench and the distributed test execution management TTmex from Testing Technologies [19]. TTmex uses CORBA [16] for the communication between distributed test components; hence the test infrastructure does not influence the Grid middleware.

4.5. Test execution

With the help of a TTCN-3 runtime environment, the TTCN-3 test suite can be executed. An example trace of the execution of the TAM test case is depicted in Figure 8. The test case has been parameterized to start two PTCs in addition to the MTC. The MTC runs on the Master node to observe and control the IUT, whereas each of the PTCs that observes the submission of jobs is running on a different node in the Grid. The assignment of MTC and PTCs to Grid nodes can be configured via TTmex that we used in our case study. In addition to the three instance axes for MTC, PTC 1, and PTC 2, the test trace in Figure 8 contains one instance axis that represents the whole SUT, i.e., the IUT and the underlying Grid environment.

In the example trace, the initial stimulus that is sent by the MTC to the SUT via the pt_http port (Message 1 in Figure 8) leads to the submissions of two jobs. From the trace, it can be seen that the SUT submits one job to the node, on which PTC 1 is running (Message 2), and one job to the node, on which PTC 2 is running (Message 3). The parameters that are observed by the PTCs are then forwarded to the MTC (messages 4 and 6). After a configurable time, each PTC that serves as stub for the Calculation Application, creates via its pt_Image port a graphic file as a result that is consumed by the SUT (messages 5 and 7). Afterwards, the SUT submits the final job to a node in order to compose a final result from all sub-results. This is observed by a PTC (Message 8) and subsequently forwarded to the MTC (Message 9). After a predefined time, this PTC creates the composed result file via its pt_Image port (Message 10). This file is consumed by the SUT. Finally, the MTC observes the presentation of the composed result image (Message 11). Since all the observations that were made by the MTC where correct, the MTC sets the verdict **pass**.

5. Comparison with related work

Our case study has demonstrated that TTCN-3 is very well applicable for testing distributed Grid applications. In particular the concepts that TTCN-3 provides for distributed testing facilitate the testing of the workflow created by an Application Management. We are aware of two works that can be related to application testing in Grid environments:

Duarte et al. describe an approach for *Multi-Environment Software Testing on the Grid* [3]. They developed and applied the tool *GridUnit* for controlling and monitoring execution of tests on several nodes of a Grid. The tests that are executed on each node are implemented using the *JUnit* [9] Java-based test framework.

The *Centre for Development of Advanced Computing* (C-DAC) provides *C-DAC Grid Computing Test Suites and Grid Probes* [1]. These test suites have the objective to check basic capabilities of the Grid middleware itself, e.g. remote job submission, validation of proxy or mutual authentication in a Grid environment that is based on GT4. These test suites are implemented in various general purpose programming languages such as C or Java.

In comparison to our TTCN-3 approach, the JUnit-based approach of Duarte et al. does not support a communication between the test components that run on the different nodes of a Grid. Therefore, a synchronization between different test components is not possible or requires a proprietary communication mechanism.

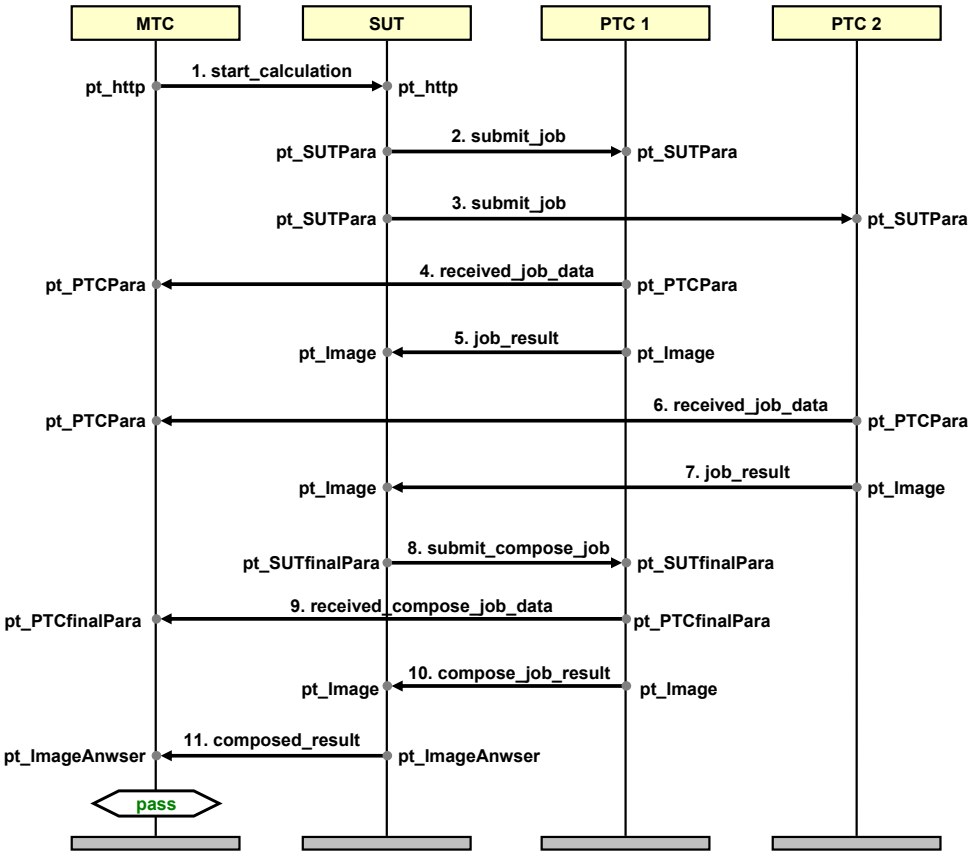

Figure 8. Test trace

The approach from C-DAC is intended only for testing the Grid middleware itself and involves to some extent distributed testing. Parts of the test suite implementations may thus also serve as example how to test Grid applications in a distributed manner.

Both related works have in common that they directly implement test suites using general purpose programming languages. Therefore, they lack the advantages that are provided by the standardized test specification and test implementation language TTCN-3. Due to the abstraction of TTCN-3, the re-usability is increased: the same TTCN-3 test suite (or parts of it) can be re-used in different Grid middleware, by simply exchanging the underlying adapters. On the other hand, the adapters can be re-used for different test suites that are executed on the same Grid middleware. Furthermore, the high-level concepts that are part of TTCN-3 increase the test development efficiency, since the existing TTCN-3 tools take charge of low-level details, e.g. the infrastructure for the communication between the distributed test components.

Standardization also started to study testing needs for Grid infrastructure. For this, *European Community/European Free Trade Association* funding was given to the ETSI *Technical Committee* (TC) *Grid* to establish a *Specialists Task Force* (STF) to study interoperability of Grid solutions. The STF started its work in October 2007. In a first step, the state of the art for Grid interoperability is studied and interoperability gaps are identified. In subsequent steps, proposals to close the identified interoperability gaps will be developed. This includes a roadmap for planning and coordinating standardization work on interoperability testing at ETSI, the study of methods and tools for Grid interoperability testing and the organization of interoperability events to test the interoperability of Grid solutions.

Our research group participates in this STF. The case study presented in this paper shows the applicability of the standardized test language TTCN-3 for Grid testing. It therefore can be regarded as a first step towards testing standards for Grid applications.

6. Summary and outlook

We presented a case study for the application of the standardized test specification and test implementation language TTCN-3 for the specification and automated execution of test cases for Grid applications. We described in de-

tail a test case for assessing the functional correctness of the workflow of a Grid application that makes use communication mechanisms like remote job submission and file transfer. Grid applications are highly distributed; hence a corresponding test requires also a distributed test system. Our case study demonstrates that the distributed test concepts of TTCN-3 and the remote communication mechanisms that are provided by a TTCN-3 runtime environment facilitate very well the distributed testing in Grid environments.

Related Grid testing approaches are not based on TTCN-3, but are implemented in a low-level general purpose programming language, such as the C language. A TTCN-3-based approach has the advantage of a higher abstraction. This abstraction leverages the re-usability, e.g. the adaptation layer can be re-used for different test suites and by simply changing the adaptation layer, the same test suite can be executed on different Grid middlewares. Additionally, the abstract level of TTCN-3 increases the test development efficiency, because the TTCN-3 tooling handles most of the low-level details, such as the communication between distributed test components.

In our case study, the test suite was executed in a Grid environment that spans a LAN. Future work will investigate the scalability of our approach and the available TTCN-3 run-time environments to Grid environments that are globally distributed with full security in place. Furthermore, we investigated only functional testing of Grid applications. Additional scopes to cover include testing the Grid middleware itself and non-functional tests (e.g. performance or stress tests) using TTCN-3. As part of our involvement in the ETSI TC *Grid*, we will also study interoperability tests. Finally, we intend to investigate the automatic generation of test cases from formalized workflow descriptions that are specified, e.g. with UML activity diagrams, Petri nets, or the *Business Process Execution Language* (BPEL).

Acknowledgments The authors like to thank Testing Technologies for providing the TTworkbench and TTmex TTCN-3 tools.

References

[1] Centre for Development of Advanced Computing. C-DAC Grid Computing Test Suites and Grid Probes. [Online; `http://www.cdac.in/HTML/npsf/gridcomputing/npsfgrid.asp` fetched on 01/15/2008].

[2] M. Di Stefano. *Distributed Data Management for Grid Computing*. John Wiley and Sons, Inc., Hoboken, NJ, 2005.

[3] A. Duarte, G. Wagner, F. Brasileiro, and W. Cirne. Multi-Environment Software Testing on the Grid. In *PADTAD-IV: Proceedings of the 2006 workshop on Parallel and Distributed Systems: Testing and Debugging*, pages 61–68. ACM Press, 2006.

[4] ETSI. ETSI Standard (ES) 201 873 V3.2.1: The Testing and Test Control Notation version 3; Parts 1-8. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation series Z.140, 2007.

[5] I. Foster. The Grid: A New Infrastructure for 21st Century Science. *Physics Today*, 55(2):42–47, 2002.

[6] I. Foster. What is the Grid? A Three Point Checklist. *Grid Today*, 1(6):22, 2002.

[7] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC05)*, volume 3779 of *LNCS*. Springer, 2005.

[8] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[9] E. Gamma and K. Beck. JUnit. [Online; `http://junit.sourceforge.net` fetched on 01/15/2008].

[10] Globus Alliance. [Online; `http://www.globus.org` fetched on 01/15/2008].

[11] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles, and C. Willcock. An introduction to the testing and test control notation (TTCN-3). *Computer Networks*, 42(3):375–403, June 2003. DOI: 10.1016/S1389-1286(03)00249-4.

[12] Instant-Grid. [Online; `http://www.instant-grid.de` fetched on 01/15/2008].

[13] ISO/IEC. Information Technology – Open Systems Interconnection – Conformance testing methodology and framework. International ISO/IEC multipart standard No. 9646, 1994-1997.

[14] KNOPPER.NET. *Knoppix*. [Online; `http://www.knoppix.org` fetched on 01/15/2008].

[15] B. B. Mandelbrot. *Fractals and Chaos: The Mandelbrot Set and Beyond*. Springer, New York, 2004.

[16] Common Object Request Broker Architecture: Core Specification, Version 3.0.3 (formal/04-03-12). Object Management Group (OMG), 2004.

[17] T. Rings. Testing Grid Applications Using TTCN-3. Master's thesis, Institute for Informatics, University of Göttingen, Germany, ZFI-BM-2007-27, 2007.

[18] I. Schieferdecker and T. Vassiliou-Gioles. Realizing distributed TTCN-3 test systems with TCI. In *Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems (TestCom2003)*, volume 2644 of *LNCS*. Springer, 2003.

[19] Testing Technologies IST GmbH TTworkbench. [Online; `http://www.testingtech.de/products_services/ttwb_intro.php` fetched on 01/15/2008].