Paper

Ensuring interoperability of command and control information systems – new ways to test conformance to the MIP solution

Nico Bau, Michael Gerz, and Michael Glauer

Abstract—In the Multilateral Interoperability Programme (MIP), 25 nations and NATO develop consensus-based, system-independent specifications to achieve semantic interoperability among distributed and heterogeneous command and control information systems (C2ISs). Implementing a distributed system is a complex and error-prone task. Therefore, extensive and efficient testing of the national MIP implementations is critical to ensure interoperability. For MIP baseline 3, Research Institute for Communication, Information Processing, and Ergonomics (FKIE) develops a test system that checks the conformance of national C2ISs with regard to the MIP specifications. It aims at reducing the testing effort and increasing the quality of MIP-compliant C2IS by automating the testing process. For that purpose, formal and executable test cases are specified. The test system is used as the MIP Test Reference System (MTRS) for the official MIP system level tests. In this paper, we motivate the development of the MTRS and describe the underlying testing approach. The client-server architecture and the test language are described in detail. Finally, the status quo and an outlook on future enhancements are given.

Keywords— Multilateral Interoperability Programme, conformance testing, MTRS.

1. Introduction

In an age, in which information superiority decides on the outcome of military missions, the interoperability of command and control information systems (C2ISs) is of paramount importance. However, the C2 systems currently fielded do not speak a common language, yet. Over the years, each nation has developed and maintained their own C2 system(s) based on their national doctrine and information requirements. This has led to dozens of systems with different, incompatible interfaces.

To support information exchange across national domains in combined and joint operations, 25 nations and NATO collaborate in the Multilateral Interoperability Programme (MIP) [5]. MIP is a voluntary forum that develops consensus-based, system-independent specifications. It aims at achieving "*international interoperability of command and control information systems (C2IS) at all levels from corps to battalion, or lowest appropriate level, in order to support multinational (including NATO), combined and joint operations and the advancement of digitization in the international arena*" [4]. MIP defines a common interface for distributed, heterogeneous C2ISs and covers operational, procedural, and technical aspects of C2 information exchange [3].

A core feature of the MIP solution is the joint command, control, and consultation information exchange data model (JC3IEDM) [6]. The JC3IEDM provides the basis for information exchange and specifies the semantics of militarily relevant objects, actions, etc., as well as their relationships in an unambiguous way. In addition, MIP defines information exchange protocols and procedures. The MIP data exchange mechanism (DEM) follows the publish-subscribe paradigm and supports partial replication of operational data.

The MIP interoperability tests. Implementing the MIP solution – like any distributed system – is a complex and error-prone task. In particular, this holds for its integration into legacy systems, which use proprietary data models and information exchange mechanisms internally. For such systems, not only the network protocols have to be implemented properly but also syntactic and semantic transformations must be applied to operational data/information. At the same time, the national MIP gateways build the backbone of the multinational network. A failure, in software or hardware, may have the most serious consequences!

Therefore, extensive and efficient testing of the national MIP implementations is critical to ensure interoperability even in special (error) situations and under heavy load.

In order to improve and evaluate the degree of interoperability of national C2ISs, MIP provides standardized test specifications with test cases of varying technical detail and abstraction. For instance, there are high-level test cases for verifying operating procedures as well as test cases for specific technical issues in the protocols of the DEM.

The MIP differs between three types of tests:

- Implementation level tests (ILT) are conducted under national responsibility.
- System level tests (SLT) demonstrate the timely end-to-end transfer of operational data between national C2ISs.
- Operational level tests (OLT) evaluate the MIP solution, when deployed in the context of an operational scenario, and validate that the MIP solution meets the operational objective.

The MIP system level tests are divided into three subcategories [7]:

- SLT 1 focuses on data transmission and communication protocols (technical level testing).
- SLT 2 focuses on the correct information exchange between JC3IEDM databases (data and procedural level testing).
- SLT 3 validates the information exchanges between C2ISs (C2IS level testing).

The MIP nations test their implementations in bi- and multilateral interoperability test sessions. These sessions are performed via the Internet or during fixed periods in Greding, Germany. According to [10], the MIP test activities can be classified as *active interoperability testing*, since some test specifications require fault injection (in order to test the error handling of the peer C2IS) or detailed information on the internal state of the MIP gateway.

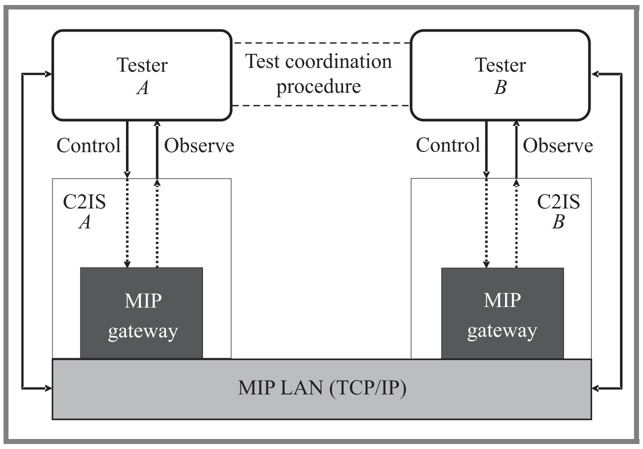


Fig. 1. Test configuration for bilateral MIP interoperability tests.

A generic test configuration for bilateral MIP tests is shown in Fig. 1. With this test configuration, the interoperability of two C2ISs *A* and *B* is tested. Both systems are stimulated by inputs and their behavior is compared with the expected results specified in the MIP test cases. Moreover, tester *A* and *B* are able to disrupt the underlying MIP LAN in order to test their implementations under adverse circumstances. During test execution, the test operators must coordinate with each other offline (e.g., by online chat) to stimulate their C2ISs in the right order and to determine the final test verdict. The test cases are executed twice with the national systems alternately taking the role of both C2IS *A* and *B*.

Restrictions of the current approach. Unfortunately, this way of testing has several limitations:

- Since the MIP test cases are described informally or semi-formally only, they easily become subject to interpretation. Moreover, they have to be performed manually. Therefore, the test results often depend on the judgment of the test operators involved (which may also be the C2IS implementers).
- Due to the needed coordination with test partners and the lack of automation, testing has proven to be very time-consuming.
- Interoperability does not necessarily imply that the systems conform to the MIP specifications. If all interoperating systems are implemented in the same erroneous way, errors remain undetected. The MIP community tries to address this problem by testing their C2ISs against as many other C2ISs as possible (3 to 5 systems). However, the resources for test sessions are limited, especially when the MIP community continues growing.
- Fieldable C2IS are not designed for testing. We cannot expect C2ISs to have (standardized) test interfaces for their internal components, as this would strongly limit implementation options.
- A C2IS does not provide dedicated support for the different stages of the testing process, i.e., test development, test preparation, test execution, and test evaluation. When testing with another C2IS, a lot of time is spent on setting up the test configuration (including resetting the operational database). Thus, it is practically impossible to run thorough regression tests after software changes.

The MIP Test Reference System. In order to support the correct implementation in national C2ISs, the Research Institute for Communication, Information Processing, and Ergonomics (FKIE) develops a dedicated test system, incorporating ideas and feedback of the MIP community[1]. Its purpose is to check the conformance of a national C2IS with regard to the MIP specifications rather than its interoperability with other C2ISs. The test system makes use of formal test cases and thus supports automated execution and evaluation of test cases. The test system is supposed to support the full range of MIP system level tests with the exception of tests for the message exchange mechanism.

Initially, the MIP Test Reference System (MTRS) was planned as a national project in order to decrease resources needed for testing several MIP implementations while increasing the test quality at the same time. Nevertheless, the plans and concepts for a conformance test system have been presented to the MIP Programme Management Group and various working parties. Due to the positive feedback, the test system is used as the MTRS for the official MIP system level tests, which have started in September 2007. The MTRS is offered free of charge to all systems participating in the MIP SLT. Furthermore, we try to make the MTRS as "transparent" as possible by unveiling its architecture, the test scripts, and the internal data flow during test execution.

[1]The project is funded by the Federal Office of the Bundeswehr for Information Management and Information Technology.

2. Conformance testing

In contrast to interoperability tests, which check whether two or more systems are able to communicate and exchange data with each other, conformance tests aim at checking the functional behavior of a single system under test (SUT) with regard to a specification. In doing so, the SUT is considered as a *black box*. The task of a conformance test system is to control the SUT by sending some input (stimuli) and to compare the observed output (responses) with the expected results.

The MIP conformance testing is challenging for two reasons.

First, the MIP solution requires the implementation of several different software components. These include the DEM protocol stack, a replication transaction component that assembles outgoing and processes incoming operational data, and, typically, a JC3IEDM-compliant database. Other software components may validate data against JC3IEDM business rules and map JC3IEDM data onto APP6a symbols on screen and vice versa. When testing for MIP conformance, these software elements cannot be tested in isolation but have to be considered as embedded components in a complex C2IS. Moreover, no clear line can be drawn between the C2IS core and the MIP-specific parts. Thus, testing MIP compliance does not stop at the gateway/interface of a C2IS but involves many different, possibly deeply hidden, C2IS components.

Second, the only standardized test interface of the C2IS under test is the MIP interface. Therefore, the control and observation of the SUT by means of an automatic test tool is restricted. If the protocol data units (PDUs) sent and received via MIP do not allow for the complete execution and evaluation of a test case, the test operator has to be involved. For instance, the test system may ask the operator to establish a contract manually via the user interface of the C2IS. In terms of the open systems interconnection (OSI) conformance testing methodology and framework (CTMF) [2], only the *remote test method* is applicable. A generic test configuration for MIP conformance testing with a single MIP tester gateway is shown in Fig. 2.

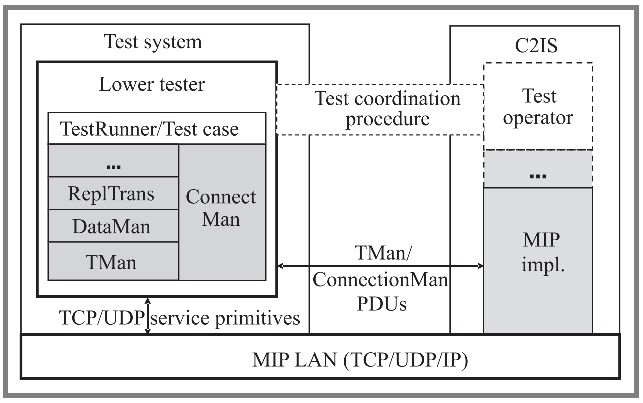


Fig. 2. Conformance testing – remote test method.

Testing of the MIP solution must happen on different technical and logical layers (protocol layers, database layers). Accordingly, test cases should be specified on different levels of abstraction. In fact, it is unacceptable and virtually impossible to specify test events on the lowest level, i.e., as TCP/UDP service primitives, when testing operational data. Therefore, parts of a MIP implementation have to be integrated into the test system itself.

For the MTRS, the MIP-specific modules have been designed as fine-grained components, each mapping to a particular aspect of the MIP specifications. They have been modeled as closely to the MIP specifications as possible. In particular, the names and the structure of the DEM message classes in the MTRS implementation match with the service primitives defined in the MIP DEM specification.

Moreover, we developed a lightweight component framework that adopts concepts from popular Java frameworks. The component framework allows to set up different test configurations and provides the test operator with information on the data flow between the components. The latter enables efficient error diagnosis and test evaluation. The concrete test configuration is specified as part of each formal test case.

"Quis custodiet ipsos custodes"[2]. The inclusion of MIP components raises the question whether the test system itself is implemented correctly[3]. There are several approaches to minimize the risk of an incorrect implementation. For instance, we use FindBugs [9], a static analysis tool that scans the source code of the MIP components for *anti-patterns*. Where possible, the test system components are derived directly from the (platform-independent) MIP specifications, which are correct by definition. Model-driven software development is applied for the MIP information resource dictionary that does not only comprise the meta model of the relational JC3IEDM but also formal representations of many JC3IEDM business rules. This way, database transactions can be handled generically.

Most importantly, a *bootstrapping* approach can be applied. The conformance test suite is not only applied to the national C2ISs but also to the MIP components of the MTRS. This is achieved in an incremental manner: starting with a test system that does not include any MIP gateway components, the test components of the lowest level (those that use TCP or UDP at their lower interfaces) are tested. Once these test components have passed all tests, they can be used in the test system to test components of the next layer. In other words: in order to test layer $n+1$, it only takes MIP components of layer $[1..n]$ and the executable test script for layer $n+1$.

This iterative process continues, until all test components have passed the conformance test suite successfully. Note that since the test cases for layer $n+1$ and the MIP components of the same layer are developed independently, the MIP implementation does not automatically pass all tests. Furthermore, the correctness of a test case can be

[2]"Who watches the watchmen?" Decimus Junius Juvenalis.

[3]Of course, the test framework, which is responsible for interpreting and executing the test cases, may also be erroneous.

verified by the MIP community by reviewing the test scripts and analyzing the data flow of test runs.

3. System architecture

The MTRS is designed as a client-server application. The high-level architecture of the MTRS is shown in Fig. 3.

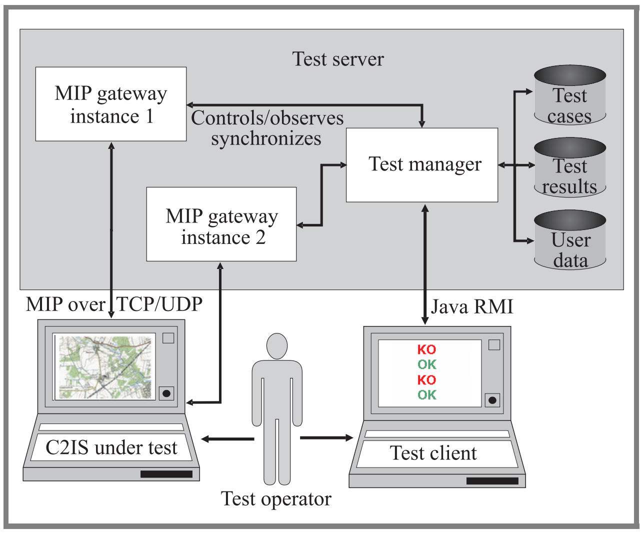


Fig. 3. Test system architecture.

The national test operator interacts with the MIP test system via the *test client* (see also below). In particular, the national test operator is able to run test cases on the server and to analyze the way the data is processed in the test server.

Conceptionally, the *test server* consists of a *test manager* that is responsible for test execution and evaluation. The test framework supports concurrent execution of test cases for different C2ISs. The MTRS test suite, the test results of each individual C2IS, and the user data are made persistent in the server database.

Depending on the test configuration required by a given test case, the test manager sets up, controls, observes, and synchronizes specific test components. In Fig. 3, two MIP gateway instances are set up. This is useful for, e.g., testing the data forwarding capabilities of a C2IS, where the test manager sends operational data via MIP gateway instance 1 and checks for the reception of the same data at MIP gateway instance 2. Exchange between the test server and the national C2IS takes place via TCP/IP and UDP/IP.

As mentioned above, the MIP interface is the only standardized interface provided by all national C2ISs. Therefore, at certain points, the test server sends action requests to the test client, which asks the test operator to perform some action or to provide feedback on what information is displayed at the C2IS's user interface.

Using a common test server shared by multiple nations has some advantages and disadvantages.

On the one hand, the integrity of test cases and test results is ensured by a central repository – the test operators are not able to change them (neither accidentally nor intentionally). Since all test results are available, cross-national test reports can be produced for the MIP test controllers at run time. For that purpose, we have implemented a PDF export functionality.

Moreover, the upgrade procedure for the test system and for test cases is simplified, as no distribution among the nations is needed. As described below, test scripts can be updated on the fly without having to restart the server. Similarly, the test suite shown in the test client can be synchronized with the server database during a user session.

Finally, server configuration and database backups are at the responsibility of a single organization, freeing the C2IS developers and test operators from administration.

On the other hand, a server-based test system is a single point of failure such that reliability and availability become crucial quality factors. In particular, we must ensure that:

– parallel test runs do not interfere;

– faulty test components do not tear down the complete server;

– no "zombie" threads remain if the test operator/client disconnects from the server without previously stopping a test run.

These issues have been addressed by encapsulating error-prone server components in a sandbox. All exceptions thrown within the sandbox are caught. Moreover, watchdog timers trigger the termination of long-running test cases and of user sessions for which no activity was noticed for a long period.

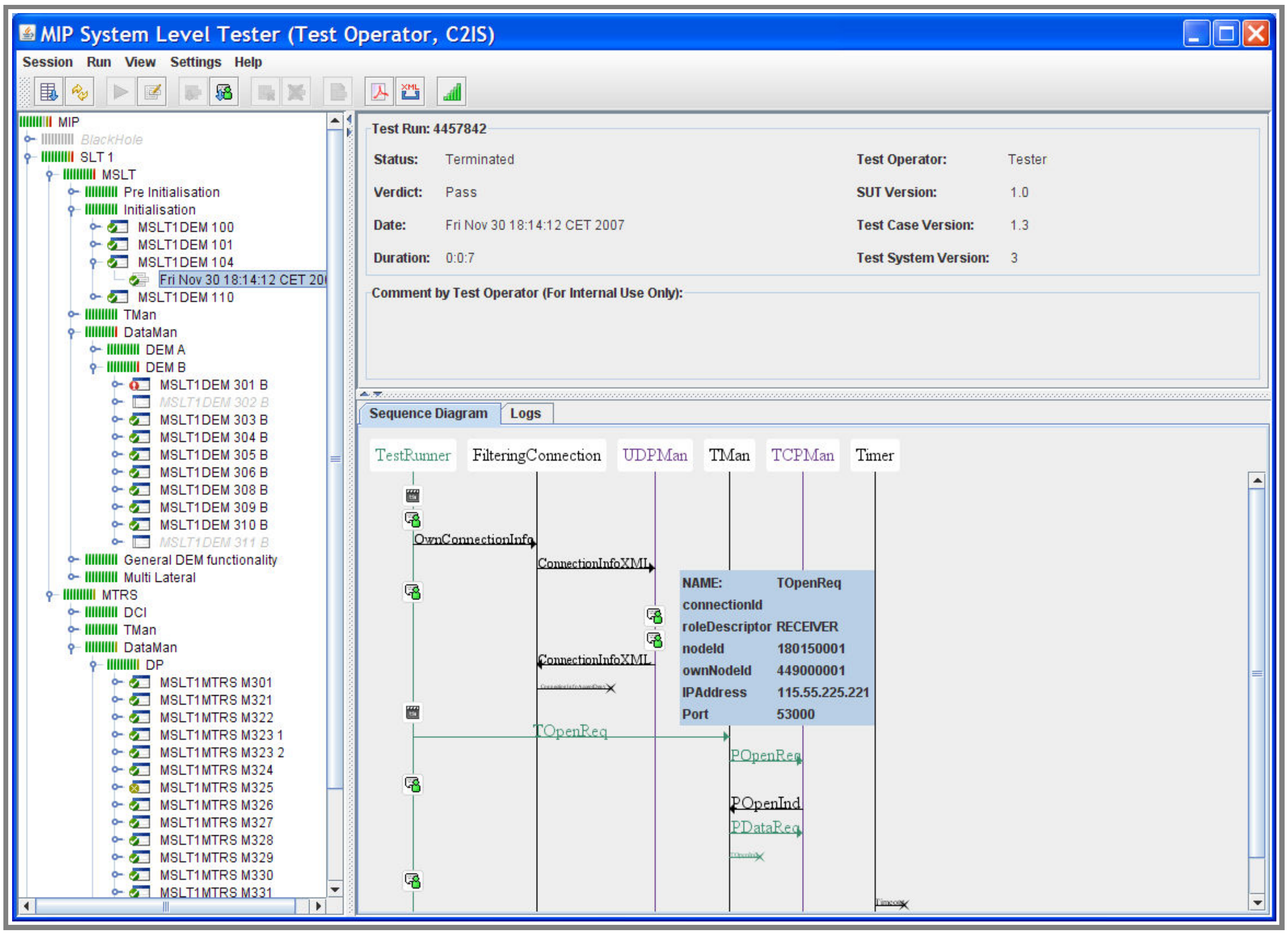
When running tests over the Internet, test operators must assure that their C2ISs are able to access the server. Typically, the companies and organizations involved in MIP have very strict firewall policies. Therefore, the MTRS was designed in a way that it uses a minimal number of fixed ports. For the MTRS client-server communication, access must be granted to only two server ports (1098/1099). For communication of the national C2IS with the MTRS server, the same TCP gateway ports are re-used for each test run.

Of course, all test data transmitted over the Internet must be unclassified. Since all test scripts – which describe the test data to be exchanged – are publicly available anyway, we do not consider this as a major problem.

Test client user interface. The MTRS client is a Java application. It can be run on any system, on which Java Runtime Environment 5 or higher is installed. Figure 4 shows a screenshot of the MTRS client.

The graphical user interface is split into three main areas. On the left, the test suite with its test groups, test cases, and corresponding test runs is represented as a tree.

On the top right, meta information is provided for the currently selected tree node. For instance, test cases are characterized by their name, version, and purpose, a reference to the relevant MIP documentation, a selection criteria, general comments, and keywords. The latter can be used for



Test Run: 4457842

Status:	Terminated	Test Operator:	Tester
Verdict:	Pass	SUT Version:	1.0
Date:	Fri Nov 30 18:14:12 CET 2007	Test Case Version:	1.3
Duration:	0:0:7	Test System Version:	3

Comment by Test Operator (For Internal Use Only):

NAME:	TOpenReq
connectionId	
roleDescriptor	RECEIVER
nodeId	180150001
ownNodeId	449000001
IPAddress	115.55.225.221
Port	53000

Fig. 4. The MTRS client.

filtering test cases in the test client. Moreover, the operator may add some C2IS-specific comments to a test case. For a test run, the MTRS keeps track of the status (running, terminated, aborted, etc.), test verdict, start date and time, and the duration, as well as the test operator and the versions of the SUT, MTRS, and test case at the time when the test was executed. In alignment with the OSI CTMF, the MTRS supports three possible test verdicts: *pass*, *fail*, and *inconclusive*. Test verdict *inconclusive* is assigned if the C2IS does not meet the test objective but behaves correctly according to the MIP specifications. Other causes for the verdict *inconclusive* include firewall and network problems.

Finally, on the bottom right, the internal data flow between the MTRS MIP gateway components and various status messages are shown. Besides displaying the log information in a tabular view, the MTRS is able to create sequence diagrams. The data flow shown in Fig. 4 corresponds to a successful execution of the test case given in Fig. 5.

4. Test specification

In order to run test cases in an automated way, they have to be written in a formal test language. Among others, such a language must fulfill a couple of test-specific requirements.

First, the test language has to allow for the definition of dynamic test configurations. It must be possible to set up and link individual test components on a per test case basis. The configuration concepts of the language must match with the component model used for the MIP implementation. As mentioned above, the MTRS is written in Java. It employs a lightweight component model based on asynchronous message exchange and uses the Java *interface* concept and the *dependency injection* pattern.

Second, the test language must provide control structures for handling non-deterministic, partially ordered, and unexpected test events. Whenever the test system expects some response from the SUT, it must be able to cope with various possible inputs, including valid, inopportune, and erroneous responses. Moreover, the order of (valid) responses may not be fixed. For instance, if a C2IS is expected to send three database records, the order of the records may be irrelevant. In other cases, alternative messages may be received (e.g., a connection confirmation or a disconnection). For clarity, the main body of a test case should describe the expected behavior. Nevertheless, the test system must be able to catch erroneous and inopportune behavior as well. For that purpose, some kind of exception handling should be available.

Third, the test language must support time and timers. Occasionally, response times of an SUT have to be constrained to check functional requirements and to make sure that, eventually, the test case terminates. For instance, timers are needed in scenarios, in which the MTRS checks whether a response does *not* occur within a given period. Moreover, it may be desirable to measure the time it takes for a C2IS to forward data from one system to another and to check whether this duration falls within a certain range of tolerance.

Finally, it must be possible to specify test verdicts based on the test events.

Ideally, the test language should be standardized. Rather than reinventing the wheel, the test language should adhere to some *de facto* or *de jure* standard. For the MIP test system, two test languages/frameworks have been investigated with regard to the requirements above: JUnit/Java and the testing and test control notation (TTCN-3).

JUnit is an open source framework for writing and running repeatable tests[4]. It provides assertions for testing expected results, test fixtures for sharing common test data, and test runners for running tests. JUnit test cases are actually Java classes that follow certain naming conventions or have special annotations.

Our assessment has shown that the control structures provided by JUnit/Java are not sufficient to cope with alternative system behavior in an elegant way. In addition, proper timer handling leads to convoluted code. This is not surprising, as JUnit was designed for unit testing rather than for testing distributed systems.

The TTCN-3 is widely used in the telecommunication and automotive area and has been standardized by the European Telecommunications Standards Institute (ETSI). In contrast to JUnit, it has very sophisticated testing features for distributed systems. However, we concluded that the "semantic gap" between TTCN-3 and Java, and the effort to integrate a TTCN-3 interpreter into the MTRS outweighs the benefits of using this standardized language. For instance, TTCN-3 provides its own data model, which does not support object-orientation. A lot of development would have been necessary for the mapping of Java classes and methods onto suitable TTCN-3 constructs and vice versa.

4.1. The MTRS test language

For the MTRS, we have defined a test language that is mainly based on Java but borrows some concepts from JUnit and TTCN-3 (Java with "syntactic sugar"). A sample MTRS test case is shown in Fig. 5.

In order to facilitate the instantiation of a component, the **new** operator of Java was complemented by a **create** operator. It does not only instantiate the respective component, but also creates proxy objects that automatically intercept all method calls of that component. Furthermore, the life cycle of a component that was instantiated using the **create** operator is tightly coupled with the execution of the test case. Thus, when the test case finishes, the component is stopped and disposed.

[4]See http://www.junit.org

Components can be linked via dependency injection. Whenever two components are linked, a proxy object is used to intercept the communication between them. For each intercepted invocation, the method's signature, the parameters provided, and information on the caller of that method are added to the respective test case's event queue. To stimulate a component, the **[!]** operator was added. Its syntax is:

[!] <*method call*> **to** <*component*>;

While the **[!]** operator only introduces a minor improvement concerning the ability to write and comprehend a test case, the addition of the operator **[?]** significantly simplifies test case specification in comparison to plain Java. The **[?]** operator checks for whether a specific method is called within a given time. The syntax of the **[?]** operator is:

[?] [<*called component*>**.**] <*method signature*>
[**from** <*calling component*>]
[**in** <*duration*>] <block>

It is used to express the expectation that a method with the provided signature is invoked. Each **[?]** operator within a test case is translated into Java code, which checks the event queue of the respective test case for communication between the called and calling component. If a proxy object has logged communication between the components, the test runner checks whether the logged method signature matches the one of the **[?]** operator. If it does, all parameters are assigned to variables and the preceding code block is executed; otherwise, an exception is thrown. If communication between the two components has not yet taken place, the test execution waits until this event is intercepted or a timeout occurs.

Additionally, the **alt** and **interleave** statements can be used to group several **[?]** operators. The **alt** block is left, if one of the **[?]** operators was triggered, whereas the **interleave** operator waits until *all* **[?]** operators were processed. This allows for waiting for multiple events that may occur in an arbitrary order.

Within the code block of the **[?]** operator, the new **repeat** statement can be used to leave the current block and to continue execution at the outer **alt** or **interleave** statement, thus effectively ignoring the event that has occurred. A single **[?]** operator without a surrounding **alt** or **interleave** is treated as being the only **[?]** operation inside an **alt** statement.

Finally, a **test** .. **handle** statement was modeled similarly to Java's **try** .. **catch**. The **handle** part lists an arbitrary number of **[?]** operators that are implicitly added to all **alt** statements within the **test** body. **test .. handle** allows to specify the reaction to unexpected or exceptional events separately from the expected test events.

The sample test case shown in Fig. 5 makes use of some of the new operators. It tests whether the SUT is able to receive and process a DEM connection information sent by the test system, and is able to send its own connection information back to the test system. Afterwards,

```
testgroup MIP.SLT_1.MSLT.Initialisation {

    /**
     * @Id 170
     * @Version 1.3
     * @Purpose Verify the establishment of a TMAN connection as a result of UDP based DEM connection information.
     * The test shall pass with a TMAN connection between the two DEMs to prove the connection information
     * was handled correctly.
     * @SelectionCriteria Low
     * @Keywords TMan DCI
     */
    testcase MSLT1DEM_104 {
        request("Please start your C2IS and expect to receive a DEM Connection Information via UDP.");

        // set up test configuration:
        Component udpMan = create UDPMan(getSut().getIpAddress(), getSut().getUdpPort());
        Component connection = create FilteringConnection(getSut().getNodeId());
        Component tcpMan = create TCPMan(getSut().getTcpPortGateway1());
        Component tMan = create TMan();

        link(udpMan, connection, "ConnectionInfoXML");
        link(connection, udpMan, "ConnectionInfoXML");
        link(connection, tMan, "TOpenIndAcceptDeny");
        link(tMan, connection, "TOpenInd");
        link(tcpMan, tMan, "PMessageInd");
        link(tMan, tcpMan, "PMessageReq");

        cm.start();

        OwnConnectionInfo ownDCI = getOwnConnectionInformation("449000001", "BLK3 SLT1 REP ORG A",
                                                               getSut().getTcpPortGateway1());
        ownDCI.setScope(Scope.ANNOUNCE);
        ownDCI.setReceiverIp(getSut().getIpAddress());
        [!] send(ownDCI) to connection;

        // now we want to receive a reply:
        [?] receive(ConnectionInfoAcceptDeny ind) from connection {
            ConnectionInfo info = ind.getConnectionInfo();
            assertEquals("nodeId", getSut().getNodeId(), info.getNodeId());
            assertEquals("ipAddress", getSut().getIpAddress(), info.getIpAddress());
            assertEquals("tcpPort", getSut().getTcpPort(), info.getTcpPort());
            assertEquals("scope", Scope.REPLY, info.getScope());
        }

        request("The MTRS successfully received your DEM Connection Information. It will open a TMan connection now.");

        TOpenReq openRequest = new TOpenReq(this, RoleDescriptor.RECEIVER, getSut().getNodeId(),
                                            ownDCI.getNodeId(), getSut().getIpAddress(), getSut().getTcpPort());
        [!] send(openRequest) to tMan;

        [?] receive(TOpenInd ind) from tMan;

        // check that the connection is open for at least 5 sec
        Timer timer = create Timer(5000);
        alt {
            [?] tMan.receive(PCloseInd ind) from tcpMan {
                return Verdict.Fail;
            }
            [?] tMan.receive(PDataInd ind) from tcpMan {
                repeat;
            }
            [?] receive(Timeout t) from timer;
        }

        return Verdict.Pass;
    }
}
```

Fig. 5. The MTRS test script.

the MTRS opens a connection to the C2IS in order to check whether the C2IS accepts connections from the node ID and TCP port provided in the DEM connection information.

4.2. Test script processing

Test scripts can be updated on the MTRS server at run time without having to shut down and restart the server. For that purpose, the server administrator connects to the MTRS server via a special administration tool and uploads the scripts. On the server, the test scripts are parsed for syntactical correctness. Then, the test suite, test group, and test case meta information (given in JavaDoc format) is extracted and stored in the server database. Next, the test scripts are transformed into pure Java classes by rewriting all test language-specific extensions and shortcuts. Finally, the Java compiler produces the corresponding Java byte code. By exploiting the class loader features of Java, it is possible to reload a Java class definition at run time or even to keep different implementations of the same class. Thus, whenever a new test run is started, the latest byte code based on the latest test script is loaded into the Java virtual machine without affecting other test runs.

5. Summary and outlook

The MIP Test Reference System introduces new ways to test the conformance of national MIP implementations to the MIP baseline 3 specifications. The MTRS aims at improving the quality of national MIP implementations (in terms of reliability, availability, and robustness), while reducing the overall testing effort (in terms of cost and time). The MTRS performs functional black box tests, i.e., it sends some stimuli to the C2IS under test and compares its responses with the expected results. In doing so, it does not rely on any specific C2IS interfaces other than those required by the MIP specifications.

The test system allows for the execution of tests on different layers and with varying test configurations. The architecture of the test server permits its simultaneous use by multiple nations without interference. Each nation performs its tests against one or two MIP gateways, exclusively set up at run time for a single test case or a group of consecutive test cases. Detailed protocol logs allow for simplified test evaluation. In particular, error diagnosis is supported by unveiling the information flow inside the test server gateway(s). Moreover, the MTRS provides powerful export features to generate MIP test reports for national use and for the MIP test controllers.

The MIP system level tests have started in September 2007. As stated in the MIP test and evaluation master plan (MTEMP) [7], “each system will test with the MTRS before testing with another system”. Many new SLT test cases have been standardized within MIP that can only be run with the help of a test system, as they cover scenarios not reproducible by a regular C2IS.

All official MIP SLT 1 test cases have been formalized, resulting in more than 100 MTRS test scripts. In addition, the MTRS offers a special test case that starts a MIP gateway with some default behavior. It can be used to perform interoperability tests over a longer period without focusing on one particular test objective. By the end of December 2007, more than 9,000 SLT 1 tests have been executed with the MTRS. For SLT 2, the MIP community has defined more than 150 test cases. Corresponding MTRS test scripts will be available in January 2008.

The early adoption of the MIP solution made it possible to gain experience with the draft standards while the specifications were written. Various corrections and improvements found their way back into the specifications. Among others, faulty entries were fixed in the MIRD and the DEM PDU grammar was simplified. The state transition tables for the DataMan protocol have been redesigned in order to enhance error handling and reporting and to formalize those aspects that were only described in textual form before. The price to pay was that significant parts of the MTRS MIP gateway implementation had to be rewritten during the specification process.

In addition to the MTRS server, the FKIE hosts a separate web server with a project management environment based on Trac [1] and Subversion [8]. At https://trac.fkie.fgan.de/MTRS, MIP nations can download the MTRS client as well as a stand-alone DEM protocol analyzer tool. All test-related MIP documents as well as the MTRS test scripts are put under version control in a Subversion repository. The Wiki documentation includes several animated tutorials for the MTRS client based on Adobe Flash technology. Furthermore, a built-in ticketing system can be used to report defects, ask questions, etc.

Our current work opens the door for many future enhancements. First, MIP tests are still specified in an ad hoc manner. Since large parts of the MIP specifications are given in a formal representation, it is possible to apply automatic test generation algorithms. For SLT 2, some of the JC3IEDM test data have already been generated automatically based on the MIP information resource dictionary.

Where such algorithms cannot be applied (due to time constraints or technical complexity), it is beneficial to get at least some information on the coverage of the existing tests. For instance, it would be interesting to know which parts of the JC3IEDM are actually covered during the final MIP operational level test (MOLT).

Another interesting testing aspect is the validation of the data exchange between two C2IS during the MOLT by network sniffing (passive testing).

Finally, our findings and tools can be generalized beyond the scope of MIP, such that they become applicable to other interoperability programs and to other Java component frameworks in general. In particular, the test language extensions may be useful for a larger software development community.

References

[1] Trac – integrated SCM and project management, Edgewall Software, 2007, http://trac.edgewall.org/

[2] "Information Technology – Open Systems Interconnection – Conformance testing methodology and framework", Parts 1 to 7, ISO/IEC 9646:1994.

[3] MIP – Multilateral Interoperability Programme. Statement of intent for the Multilateral Interoperability Programme (MIP), (the anzio agreement), Nov. 2003, http://www.mip-site.org/

[4] MIP – Multilateral Interoperability Programme. MIP standard briefing, Dec. 2006, http://www.mip-site.org

[5] MIP – Multilateral Interoperability Programme. MIP home page, 2007, http://www.mip-site.org/

[6] MIP – Multilateral Interoperability Programme. The joint C3 information exchange data model (JC3IEDM main), ed. 3.1b, Dec. 2007, http://www.mip-site.org

[7] MIP – Multilateral Interoperability Programme. The MIP test and evaluation master plan (MTEMP), ed. 3.1, May 2007, http://www.mip-site.org

[8] Subversion: version control system, Tigris.org, 2007, http://subversion.tigris.org/

[9] FindBugs^{TM} – Find Bugs in Java programs, University of Maryland, 2007, http://findbugs.sourceforge.net/

[10] T. Walter, I. Schieferdecker, and J. Grabowski, "Test architectures for distributed systems – state of the art and beyond", in *Test. Commun. Syst. IFIP TC6 11th Int. Worksh. Test. Commun. Syst. IWTCS*, A. Petrenko and N. Yevtushenko, Eds., Tomsk, Russia, 1998, vol. 131, pp. 149–174.

Nico Bau studied computer science at the Bonn-Rhein-Sieg University of Applied Sciences, Germany. In 1999, he founded his own company, in which he worked as a software engineer and consultant. In May 2006, he joined the Research Institute for Communications, Information Processing, and Ergonomics (FKIE) of the Research Establishment for Applied Science (FGAN) in Wachtberg, contributing more than seven years of experience in Java programming to the development and design of the Multilateral Interoperability Programme Test Reference System.
e-mail: bau@fgan.de
Research Institute for Communications,
Information Processing, and Ergonomics (FKIE)
Department ITF
Research Establishment for Applied Science (FGAN)
Neuenahrer st 20
D-53343 Wachtberg-Werthhoven, Germany

Michael Gerz studied computer science with focus on computational linguistics at the University of Koblenz, Germany. He worked as a Research Assistant at the University of Lübeck and at the Institute for Telematics in Trier, Germany. In 2003, he received his Ph.D. from the University of Göttingen. His dissertation deals with automatic test generation based on formal specifications. Since 2004, he is a Senior Researcher at the Research Institute for Communications, Information Processing, and Ergonomics (FKIE) of the Research Establishment for Applied Science (FGAN) in Wachtberg. Currently, he is the project manager for the Multilateral Interoperability Programme Test Reference System.
e-mail: gerz@fgan.de
Research Institute for Communications,
Information Processing, and Ergonomics (FKIE)
Department ITF
Research Establishment for Applied Science (FGAN)
Neuenahrer st 20
D-53343 Wachtberg-Werthhoven, Germany

Michael Glauer received the information systems degree with distinction from the University of Applied Sciences of Hof/Saale, Germany, in 2004. Since July 2006, he is a Research Associate at the Research Institute for Communications, Information Processing, and Ergonomics (FKIE) of the Research Establishment for Applied Science (FGAN) in Wachtberg and participates in the Multilateral Interoperability Programme. Prior, he was an independent IT consultant and software developer for mobile computing. His current research interests include software testing, distributed systems, and virtual object databases.
e-mail: glauer@fgan.de
Research Institute for Communications,
Information Processing, and Ergonomics (FKIE)
Department ITF
Research Establishment for Applied Science (FGAN)
Neuenahrer st 20
D-53343 Wachtberg-Werthhoven, Germany