Assessing and Improving Interoperability of Distributed Systems

Dissertation
zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen

im Promotionsprogramm Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Thomas Rings
aus Gotha

Göttingen, 2012

Betreuungsausschuss

Prof. Dr. Jens Grabowski,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Stephan Waack,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Helmut Neukirchen,
Faculty of Industrial Engineering, Mechanical Engineering and Computer Science,
University of Iceland

Mitglieder der Prüfungskommission

Referent:	Prof. Dr. Jens Grabowski, Institut für Informatik, Georg-August Universität Göttingen
Korreferent:	Prof. Dr. Arnulf Quadt, II. Physikalisches Institut, Georg-August Universität Göttingen
Korreferent:	Prof. Dr. César Viho, Education and Research Department in Computer Science and Electrical Engineering, ISTIC-University of Rennes 1

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Dieter Hogrefe,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Ulrich Sax,
Geschäftsbereich Informationstechnologie, Universitätsmedizin Göttingen

Prof. Dr. Ramin Yahyapour,
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 23. Januar 2013

Abstract

Achieving interoperability of distributed systems offers means for the development of new and innovative business solutions. Interoperability allows the combination of existing services provided on different systems, into new or extended services. Such an integration can also increase the reliability of the provided service. However, achieving and assessing interoperability is a technical challenge that requires high effort regarding time and costs. The reasons are manifold and include differing implementations of standards as well as the provision of proprietary interfaces. The implementations need to be engineered to be interoperable. Techniques that assess and improve interoperability systematically are required.

For the assurance of reliable interoperation between systems, interoperability needs to be assessed and improved in a systematic manner. To this aim, we present the *Interoperability Assessment and Improvement* (IAI) process, which describes in three phases how interoperability of distributed homogeneous and heterogeneous systems can be improved and assessed systematically. The interoperability assessment is achieved by means of interoperability testing, which is typically performed manually. For the automation of interoperability test execution, we present a new methodology including a generic development process for a complete and automated interoperability test system. This methodology provides means for a formalized and systematic assessment of systems' interoperability in an automated manner. Compared to manual interoperability testing, the application of our methodology has the following benefits: wider test coverage, consistent test execution, and test repeatability.

We evaluate the IAI process and the methodology for automated interoperability testing in three case studies. Within the first case study, we instantiate the IAI process and the methodology for *Internet Protocol Multimedia Subsystem* (IMS) networks, which were previously assessed for interoperability only in a manual manner. Within the second and third case study, we apply the IAI process to assess and improve the interoperability of grid and cloud computing systems. Their interoperability assessment and improvement is challenging, since cloud and grid systems are, in contrast to IMS networks, heterogeneous. We develop integration and interoperability solutions for grids and *Infrastructure as a Service* (IaaS) clouds as well as for grids and *Platform as a Service* (PaaS) clouds. These solutions are unique and foster complementary usage of grids and clouds, simplified migration of grid applications into the cloud, as well as efficient resource utilization. In addition, we assess the interoperability of the grid-cloud interoperability solutions. While the tests for grid-IaaS clouds are performed manually, we applied our methodology for automated interoperability testing for the assessment of interoperability to grid-PaaS cloud interoperability successfully. These interoperability assessments are unique in the grid-cloud community and provide a basis for the development of standardized interfaces improving the interoperability between grids and clouds.

Zusammenfassung

Interoperabilität von verteilten Systemen ist eine Grundlage für die Entwicklung von neuen und innovativen Geschäftslösungen. Sie erlaubt es existierende Dienste, die auf verschiedenen Systemen angeboten werden, so miteinander zu verknüpfen, dass neue oder erweiterte Dienste zur Verfügung gestellt werden können. Außerdem kann durch diese Integration die Zuverlässigkeit von Diensten erhöht werden. Das Erreichen und Bewerten von Interoperabilität stellt jedoch eine finanzielle und zeitliche Herausforderung dar. Zur Sicherstellung und Bewertung von Interoperabilität werden systematische Methoden benötigt.

Um systematisch Interoperabilität von Systemen erreichen und bewerten zu können, wurde im Rahmen der vorliegenden Arbeit ein Prozess zur *Verbesserung und Beurteilung von Interoperabilität* (IAI) entwickelt. Der IAI-Prozess beinhaltet drei Phasen und kann die Interoperabilität von verteilten, homogenen und auch heterogenen Systemen bewerten und verbessern. Die Bewertung erfolgt dabei durch Interoperabilitätstests, die manuell oder automatisiert ausgeführt werden können. Für die Automatisierung von Interoperabilitätstests wird eine neue Methodik vorgestellt, die einen Entwicklungsprozess für automatisierte Interoperabilitätstestsysteme beinhaltet. Die vorgestellte Methodik erleichtert die formale und systematische Bewertung der Interoperabilität von verteilten Systemen. Im Vergleich zur manuellen Prüfung von Interoperabilität gewährleistet die hier vorgestellte Methodik eine höhere Testabdeckung, eine konsistente Testdurchführung und wiederholbare Interoperabilitätstests.

Die praktische Anwendbarkeit des IAI-Prozesses und der Methodik für automatisierte Interoperabilitätstests wird durch drei Fallstudien belegt. In der ersten Fallstudie werden Prozess und Methodik für *Internet Protocol Multimedia Subsystem* (IMS) Netzwerke instanziiert. Die Interoperabilität von IMS-Netzwerken wurde bisher nur manuell getestet. In der zweiten und dritten Fallstudie wird der IAI-Prozess zur Beurteilung und Verbesserung der Interoperabilität von Grid- und Cloud-Systemen angewendet. Die Bewertung und Verbesserung dieser Interoperabilität ist eine Herausforderung, da Grid- und Cloud-Systeme im Gegensatz zu IMS-Netzwerken heterogen sind. Im Rahmen der Fallstudien werden Möglichkeiten für Integrations- und Interoperabilitätslösungen von Grid- und *Infrastructure as a Service* (IaaS) Cloud-Systemen sowie von Grid- und *Platform as a Service* (PaaS) Cloud-Systemen aufgezeigt. Die vorgestellten Lösungen sind in der Literatur bisher nicht dokumentiert worden. Sie ermöglichen die komplementäre Nutzung von Grid- und Cloud-Systemen, eine vereinfachte Migration von Grid-Anwendungen in ein Cloud-System sowie eine effiziente Ressourcennutzung. Die Interoperabilitätslösungen werden mit Hilfe des IAI-Prozesses bewertet. Die Durchführung der Tests für Grid-IaaS-Cloud-Systeme erfolgte manuell. Die Interoperabilität von Grid-PaaS-Cloud-Systemen wird mit Hilfe der Methodik für automatisierte Interoperabilitätstests bewertet. Interoperabilitätstests und deren Beurteilung wurden bisher in der Grid- und Cloud-Community nicht diskutiert, obwohl sie eine Basis für die Entwicklung von standardisierten Schnittstellen zum Erreichen von Interoperabilität zwischen Grid- und Cloud-Systemen bieten.

Acknowledgements

At this point, I thank all the people that supported me in writing this thesis. First of all, I would like to thank my doctoral supervisor Prof. Dr. Jens Grabowski, who made it possible for me to conduct research in a stimulating environment and under excellent conditions. Not only his friendly guidance over the last years as well as his financial support for lots of trips were the basis for the successful completion of this thesis.

Also, I am very grateful to Prof. Dr. Arnulf Quadt and Prof. Dr. César Viho for agreeing to act as a referee for this thesis. I would also like to thank the remaining members of my defense committee: Prof. Dr. Dieter Hogrefe, Prof. Dr. Ulrich Sax, and Prof. Dr. Ramin Yahyapour.

Moreover, I want to thank all my current and prior colleagues at the Institute of Computer Science and at the European Telecommunications Standards Institute (STF331, STF370) for an enjoyable and scientifically inspiring environment, as well as various after-work activities. Sharing individual problems as well as discussing novel solutions have made my day-to-day work balanced and varied. I am especially grateful for the effort that Steffen Herbold, Patrick Harms, Michael Cohrs, and Philip Makedonski put into the proof reading of this thesis. I also thank all diploma students and student workers that assisted me in trying several directions and approaches helping me to save time with their implementations.

Especially, I would like to thank Benjamin Zeiss, Stephan Schulz, and Tibor Kálmán for the interesting scientific discussions, cooperation and for their constant encouragement.

Special thanks go to my parents Barbara and Hartmut for their education and support during my schooldays and studies. They have always encouraged me and my sister to do the best in all matters of life.

I thank especially my wife Fernanda for her understanding for my work and her enduring patience for months of traveling. Without her unconditional love and her bringing up of our little son Adrian, this thesis would never have been finished.

Last but not least, I want to thank my family and all my friends for their support.

Contents

1 Introduction

Interoperability of distributed systems is vital to succeed in today's market. On one hand, interoperability can be leveraged to open new markets, to foster innovation, and to enable mass markets by creating new and innovative solutions through the composition of interoperable systems. This allows service enrichment by integrating services only available in another system and to increase productivity by consuming such extended services. Furthermore, interoperability provides means to increase system availability and reliability. On the other hand, customers demand interoperable and diverse systems as well as competition in a market, which are both fostered by interoperability. An example for interoperability is the possibility to use a cell phone in different networks implemented by different vendors. For this scenario, the different networks are required to be interoperable.

The development of new solutions by combining purchased or in-house systems improves the quality of the resulting system and allows a faster development of new solutions leading to a shorter time to market [69]. A system A that is developed by vendor X should be able to interoperate with a system B, which provides the same or complementary functionality as system A but is implemented by vendor Y. Both systems need to be engineered to be interoperable. An interim approach is an interoperability gateway solution that allows communication between systems. An interoperability gateway converts messages received by one system into a representation understandable by another system to allow their interoperation. The long-term approach to achieve interoperable systems is the implementation of a common set of open standards[1]. Standards define architectures and interfaces as well as specify protocols to be used for communication via these interfaces. Ideal standards are independent of implementations and leave space for innovation. Even if standards are assumed unambiguous, which is rarely the case, testing is needed to validate that implementations conform to standards. A further step is to test whether implementations are able to interoperate, because the implementation of the same standard does not necessarily mean that systems are able to interoperate. One of the reasons is that standards are often specified ambiguously [144] and can, therefore, be interpreted differently by developers or vendors. Furthermore, options within a standard might lead to inconsistencies. Therefore, the standards themselves need to be assessed and engineered for interoperability, as well.

[1] Throughout the whole thesis we use "standard", which can be exchanged with "specification" depending on the progress of the standardization. We consider such a standard to be standardized and published by an organization such as *Open Grid Forum* (OGF), *World Wide Web Consortium* (W3C), or *European Telecommunications Standards Institute* (ETSI).

Interoperability testing assesses the end-to-end service provision between systems provided by different vendors. Ideally, all participating systems are tested and assessed for interoperability against one reference implementation, which is a fully functional implementation of one or more standards. Today's interoperability testing is still largely performed in a time consuming and resource intensive manual manner [14]. This is caused by the high number of systems and standards that are involved in complex distributed systems. The implementations, their interfaces as well as standards need to be engineered to be interoperable. In addition, interoperation needs to be reliable and, therefore, assessed for correct functioning. The interoperability engineering and assessment has many constraints regarding interoperability solutions and interoperable systems. Therefore, measures for interoperability engineering and assessment can only be developed and applied by experts of the systems. Techniques for a systematic assessment and improvement of interoperability are required. In addition, interoperability testing is not transitive [146]: If a system A interoperates with a system B and system B interoperates with a system C, it does not necessarily mean that system A interoperates with system C. This also results in a large amount of required test executions, which grows exponentially with the number of systems involved. Furthermore, after a new version of one of the systems is released, all interoperability tests need to be re-executed against all other participating systems to assess their interoperability.

In this thesis, we present a process to systemically assess and improve interoperability of distributed homogeneous and heterogeneous systems in a systematic manner to cope with the issues described above. We develop a new methodology for automated assessment of interoperability that enables a systematic specification of an automated interoperability test system. We show the practical application of the process and the methodology for homogeneous systems, i.e., for interoperability of *IP Multimedia Subsystem* (IMS) networks and for heterogeneous systems, i.e., for interoperability of grid and cloud systems.

1.1 Contribution of the Thesis

This thesis advances the state-of-the-art regarding improving and assessing interoperability of distributed systems with the following contributions.

- The ***Interoperability Assessment and Improvement* (IAI) process** (Chapter 3) describes how systems are analyzed, improved, and assessed for interoperability in three phases. The analyses are based on documents that specify the functionalities of the systems as well as on interoperability initiatives. The improvement is either based on standards or on interoperability gateways. The assessment is done by means of interoperability tests. The IAI process is applicable for homogeneous and heterogeneous distributed systems. Homogeneous systems are systems that implement the same standards. Heterogeneous systems do not implement the same standards, but provide either common or complementary functionality as the basis for the interoperation of the systems.

- A **methodology for automated interoperability testing** (Chapter 4) that is comprised of four main parts. 1) We specify a generic environment for interoperability tests with message checks, which builds the basis for a development of automated interoperability tests. 2) We provide guidelines for interoperability test design and test automation. 3) We describe a generic library for automated interoperability tests using TTCN-3 that implements the generic environment as well as the guidelines. 4) We develop a generic development process for the systematic specification of a complete and structured automated interoperability test system.
  This methodology provides a first step towards a formalized and systematic assessment of interoperability in an automated manner and can be utilized in the IAI process.

We evaluate the two contributions in case studies. From the results of the case studies, we present three further contributions to the state-of-the-art of interoperability testing of IMS networks as well as of interoperable grid and clouds computing systems:

- **Automated execution of interoperability tests for IMS networks** with an interoperability test suite for IMS implemented using TTCN-3 and developed by instantiating the methodology for automated interoperability testing (Section 5.1). The automated interoperability test execution avoids the previously performed manual execution and improves the efficiency if IMS interoperability testing.
- Two **feasibility studies of the integration of grid computing systems and clouds**, which show that grid systems and clouds are able to interoperate on different levels, i.e., between grid systems and *Infrastructure as a Service* (IaaS) clouds (Section 6.2) as well as between grid systems and *Platform as a Service* (PaaS) clouds (Section 7.2). We present unique solutions to achieve their interoperability.
- The **assessment of the interoperability of grid computing systems and clouds** by application of interoperability tests for interoperable grids and IaaS clouds (Section 6.3) as well as for grids and PaaS clouds (Section 7.3). Both are unique in the grid-cloud community and have neither been developed nor executed, yet. The results of the assessment can be used as a basis for grid and cloud standardization.

1.2 Impact

The results of this dissertation have been peer-reviewed and published in three international journals and three international conference proceedings. The subsequent list presents the journal articles:

- Springer International Journal on Software Tools for Technology Transfer (STTT, accepted, to appear in 2013): *A Generic Interoperability Testing Framework and a Systematic Development Process for Automated Interoperability Testing.* Thomas Rings, Patrick Poglitsch, Stephan Schulz, Luca Serazio, and Theofanis Vassiliou-Gioles.

- IARIA International Journal On Advances in Systems and Measurements (Vol. 3(1&2) 2011): *A Testing Framework for Assessing Grid and Cloud Infrastructure Interoperability.* Thomas Rings, Jens Grabowski, and Stephan Schulz.
- Springer Journal of Grid Computing: Special Issue on Grid Interoperability (JoGC Vol. 7(3) 2009): *Grid and Cloud Computing: Opportunities for Integration with the Next Generation Network.* Thomas Rings, Geoff Caryer, Julian Gallop, Jens Grabowski, Tatiana Kovacikova, Stephan Schulz, and Ian Stokes-Rees.

In the following, we list the conference publications:

- IEEE 5th International Conference on Cloud Computing (CLOUD 2012): *Pragmatic Integration of Cloud and Grid Computing Infrastructures.* Thomas Rings and Jens Grabowski.
- IARIA 2nd International Conference on Advances in System Testing and Validation Lifecycle (VALID 2010): *On the Standardization of a Testing Framework for Application Deployment on Grid and Cloud Infrastructures.* Thomas Rings, Jens Grabowski, and Stephan Schulz.
- 13th International Conference on Intelligence in Next Generation Networks (ICIN 2009): *Grid/Cloud Computing Interoperability, Standardization and the Next Generation Network.* Geoff Caryer, Julian Gallop, Jens Grabowski, Tatiana Kovacikova, Thomas Rings, Stephan Schulz, Ian Stokes-Rees.

Furthermore, the author identified the topics for and supervised one Master thesis, one Bachelor thesis, and one student project with relation to the overall topic of this thesis:

- Maik Doleys: *Using Cloud Computing Resources in Grid Systems: An Integration of Amazon Web Services into UNICORE 6.* Bachelor Thesis. 2011.
- Dalia Dahman: *Extension of a Globus Toolkit 4 Grid System by a Virtual Runtime Environment based on Eucalyptus.* Master Thesis. 2010.
- Dalia Dahman: *Establishment and Configuration of a Grid Environment Based on Globus Toolkit 4 (GT4) Using Torque Portable Batch System (PBS) and the Deployment of a Grid Application.* Student Project. 2010.

1.3 Structure of the Thesis

This thesis is structured as follows. In Chapter 2, we introduce the prerequisites that are needed across all chapters. We describe concepts related to interoperability and software testing as well as to the systems under study. In Chapter 3, we present the IAI process that is applied for assessing and improving interoperability of systems. The IAI process comprises activities for analyzing interoperability, engineering interoperability, and interoperability testing. In Chapter 4, we present a methodology for automated interoperability

testing that can be applied in the third phase of the IAI process. In Chapter 5, we apply the IAI process and the methodology for automated interoperability testing for the IMS. In Chapter 6, we apply the IAI process for grid systems and IaaS clouds with manual interoperability assessment due to their diverse interfaces. In Chapter 7, we assess and improve the interoperability of grid and PaaS cloud systems by the application of the IAI process. We conclude this thesis with a summary, a discussion, and an outlook in Chapter 8.

2 Prerequisites

This chapter describes the prerequisites that are the basis for this entire work. In Section 2.1, we define interoperability and discuss the different categories and levels of interoperability. In Section 2.2, we describe the main concepts of software testing including types of testing, a test specification development process, as well as *Testing and Test Control Notation Version 3* (TTCN-3). Afterwards, in Section 2.3, we focus on the systems that we analyze and apply in our case studies. They include mainly cloud and grid systems, as well as the telecommunication service IMS. This chapter is partly adapted from [117, 118, 119, 120, 121, 122].

2.1 Interoperability

Interoperability is the "ability of two or more systems or components to exchange information and to use the information that has been exchanged" [71]. The information is exchanged across possibly standardized interfaces using communication protocols and procedures to provide end-to-end functionalities to end users of the systems. These functionalities are specified by standards and implemented within components of different systems, which need to be assessed for interoperability with other systems. A system is "a collection of components organized to accomplish a specific function or set of functions" [71].

Closely related but distinct to interoperability is portability. Portability is "the ease with which a system or component can be transferred from one hardware or software environment to another" [71]. A software is portable in case the software does not rely on features that are unique to a particular type of computer or software environment. For example, a portable software can be installed on a Linux as well as on a Microsoft Windows operating system. However, this does not inherently mean that the operating systems are interoperable.

Interoperability is crucial to ensure delivery of services across systems from different vendors. It can be distinguished into four levels, which are from the bottom to the top: technical, syntactical, semantical, and organizational interoperability [144]. The upper levels rely on the lower levels, such as that semantical interoperability cannot take place without syntactical interoperability.

Technical interoperability means to enable machine-to-machine communication based on hardware or software systems. It focuses mainly on communication protocols and the infrastructure that is required for their operation. Syntactical interoperability considers the

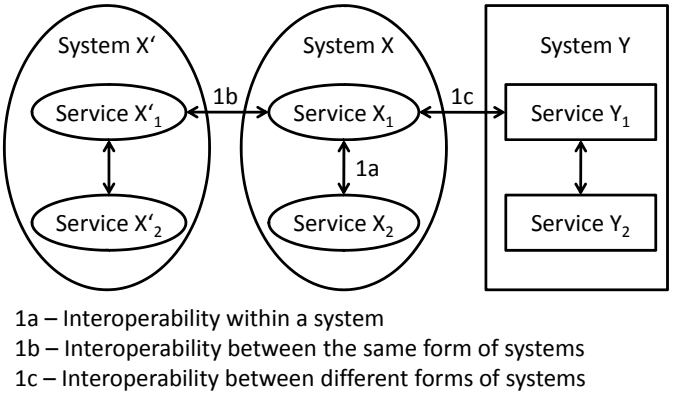


Figure 2.1: Types of interoperability

data formats that are used in communication protocols. This means that a well defined syntax and encoding is required for messages that are transferred by communication protocols. Semantical interoperability is related to the meaning of content. If semantical interoperability is fulfilled, humans understand the same when interpreting the same content. It is not centered on machine interpretation. Organizational interoperability is the ability of a meaningful communication of data over different infrastructures between different organizations. Organizational interoperability includes the linking of business processes [85]. The objects for interoperability differ in each level: signals in technical interoperability, data in syntactical interoperability, information in semantical interoperability, and processes in organization interoperability [85]. In this thesis, we discuss technical and syntactical interoperability, which we both call “interoperability” in the remainder of this work. We do not consider semantical or organizational interoperability [144].

Depending on the kind of distributed systems, interoperability can be interpreted differently. In general, we distinguish between three different types of technical interoperability: interoperability within a system, between the same form of systems, and between different forms of systems [34]. Figure 2.1 depicts these different types.

Interoperability within a system is the ability of services provided by a single system to communicate by well defined interfaces (Figure 2.1–1a). This means that the services within a specific system are able to interoperate through common, standardized, or otherwise agreed upon interfaces inside the infrastructure. A practical example is the requirement to utilize two different components such as a billing and a monitoring service implemented by different vendors that need to communicate within one system. This type is also called integration, which is “the process of combining software components, hardware components, or both into an overall system” [71].

Interoperability between systems is usually located at user domain level, i.e., interoperability between end users. Figure 2.1–1b shows the interoperation between two systems of

the same form, such as two cloud environments. The systems X and X' need to communicate and exchange data through one or more standardized interfaces. More specifically, the services provided by system X understand the services provided by system X'. In practice, this means, for example, a service is able to use an execution service of another system to reduce computational time. However, this also often involves interoperability of other services such as authentication and authorization.

Another type of interoperability is interoperability between different forms of systems, e.g., between a system X and a system Y of another form as depicted in Figure 2.1–1c. Despite other considerations, it needs to be determined if the services that need to interoperate are provided by the systems in either a substitutable or complementary way. The systems should be able to interact in order to exchange information and data, or provide access to resources. This type could involve, for example, the interoperability between a grid and a cloud system. A grid system could be extended with storage offered by a cloud computing environment.

Within this thesis, we consider interoperability between the same form of systems, and between different forms of systems. The integration within a single system is out of our scope.

2.2 Software Testing

Software testing is an analytic activity for evaluating the quality of software [147], which is part of the activities of *Software Quality Assurance* (SQA) [97]. SQA additionally contains activities of organizing examinations of software to avoid errors [88]. They contain software project management as well as constructive activities including software engineering techniques.

Testing examines test objects by their execution to check if the test objects execute as expected. A test object is a part of a software system or the software system as a whole. The goal of testing is the detection of failures, which indicate defects of the tested software. Therefore, testing itself provides a basis for debugging. Besides this, testing can increase the confidence in a software product, measure quality, and avoid defects through analyzing programs or their documentations [81, 132].

In the following, we define relevant terminology of software testing and discuss different types of testing. In addition, we describe a process for the development of test specifications, as well as TTCN-3, which is a standardized language for specifying test suites.

2.2.1 Definition of Required Test Terminologies

The following terms and definitions are used throughout this thesis.

- **Test**: A test means "an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component." [71].
- ***Implementation Under Test* (IUT)**: An IUT is "an implementation of one or more *Open Systems Interconnection* (OSI) protocols in an adjacent user/provider relationship, being that part of a real open system which is to be studied by testing" [73].
- ***Equipment Under Test* (EUT)**: An EUT corresponds to a complete system that can consist of several soft- and hardware components. An EUT will be tested for interoperability against other EUTs. This definition updates the original definition for this term provided in [36].
- ***System Under Test* (SUT)**: An SUT is "the real open system in which the IUT" [73] or respectively the EUTs "reside" [73]. The collection of all EUTs is called the SUT [36].
- **Requirement**: A requirement is "(1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2)" [71]. Related to testing, this means that a requirement describes a specific behavior of the IUT or respectively EUT, i.e., a series of stimuli to and expected outputs from the IUT or respectively EUT that can be assessed by means of a test [118].
- ***Implementation Conformance Statement* (ICS)**: A protocol ICS is "a statement made by the supplier of an OSI implementation or system, stating which capabilities have been implemented for a given OSI protocol" [73]. An ICS is basically a "checklist for providing information about an implementation to a specification, by presenting in a uniform manner the implemented capabilities (e.g., functions, features) and options as well as limitations of the implementation" [152].
- **Test Architecture**: A test architecture is an "abstract description of logical entities as well as their interfaces and communication links involved in a test" [35] related to the SUT.
- **Test Configuration**: A test configuration is a "concrete instance of a test architecture defined on the basis of test components, ports and their connection" [35] related to the whole test system.
- **Test Purpose**: A test purpose is "a prose description of a narrowly defined objective of testing, focusing on a single conformance requirement as specified in the appropriate OSI International Standard or CCITT Recommendation (e.g. verifying the support

of a specific value of a specific parameter" [73]. A test purpose specifies which cataloged requirement should be assessed in the context of a given test architecture. Each test purpose includes at least one reference to the clause in a specification, where the requirement to be assessed is described. It should have a unique identifier reflecting its place in the test suite structure. A test purpose is also referred to as a test condition [75].

- **Test Description** A test description is a detailed but informal specification of the preconditions and test steps needed to cover one or more given test purposes. It also specifies the equipment required for a test, equipment operations, observations, as well as protocol messages or procedures to be checked between systems [122]. A test description shall contain the following information [119]:

  - *Identifier*: A unique identifier that relates a test to its group and sub-group.
  - *Summary*: A unique description of the test purposes covered by this test.
  - *Test Architecture*: A reference to all equipments required for the execution of this test as well as their connections.
  - *Specification References*: One or more references to clauses in the standard for which the test purposes have been specified.
  - *Pre-test Conditions*: A list of all conditions that have to be fulfilled prior to the execution of a test. These conditions should identify the features that are required to be supported by participating equipment to be able to execute this test.
  - *Test Sequence*: A test sequence is written in terms of external actors and their ability to interact and observe the services provided by the system, i.e., end-to-end behavior. Based on its success, a test verdict reflecting the interoperability of all systems participating in a test is derived.

  If further information is required to accurately describe a test, the list of information fields can be extended.
- **Test Case**: A test case is "a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement" [71]. A test case can be generic, abstract, or executable as described in [73].
- **Test Suite**: A test suite is "a set of several test cases for a component or system under test" [75].

2.2.2 Types of Testing

In the following, we describe the types of testing considered in this thesis. They are conformance testing, interoperability testing, and their combination.

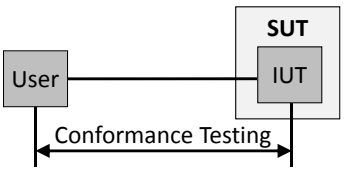


Figure 2.2: Conformance testing

2.2.2.1 Conformance Testing

Conformance testing is "testing the extent to which an Implementation Under Test (IUT) satisfies both static and dynamic conformance requirements" [36]. This means that conformance testing is generally used to check whether an implementation follows the requirements stated in a standard.

In conformance testing, one IUT is tested with functional black-box tests to check if the IUT is conform to a standard. Figure 2.2 schematically depicts this test setup. The IUT is embedded in the SUT, which is a testing environment that also includes parts that are required by the IUT to provide its service or functionality to the user. Conformance testing usually requires the development and implementation of sophisticated testing tools, e.g., based on TTCN-3 [32]. Such tools support the simulation of the environment, which is needed for a proper execution of the IUT.

2.2.2.2 Interoperability Testing

Interoperability is assessed through interoperability testing, which is the "activity of proving that end-to-end functionality between (at least) two communicating systems is as required by the base standard(s) on which those systems are based" [36]. In interoperability testing, all participating systems are usually tested and assessed for interoperability against a qualified equipment [36], which is illustrated in Figure 2.3. A *Qualified Equipment* (QE) is a reference implementation, which is a fully functional implementation of one or more standards. But the determination of a reference implementation for interoperability testing is difficult; because it needs to be assured that the reference implementation implements the standards correctly. However, each participating system implementation should be able to interoperate with all the others, not only with the reference implementation. Systems should rather be tested for interoperability against each other. Therefore, we updated the definition of interoperability testing described in [36]. We removed the QE and, therefore, avoid its determination. Each tested system in interoperability testing is called an EUT. The collection of all EUTs is called the SUT [36]. Figure 2.4 depicts the interoperability test setup. Using this approach, interoperability testing provides a feasible way to assess if two or more systems are able to communicate or interoperate, i.e., to understand exchanged data.

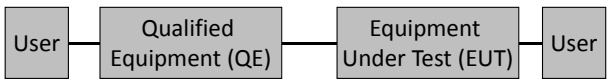


Figure 2.3: Interoperability testing with a qualified equipment [36]

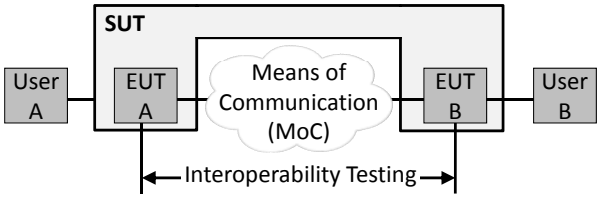


Figure 2.4: Interoperability testing

The EUTs interoperate via an abstract *Means of Communication* (MoC). It is generally assumed that the communication services used between EUTs are compliant to underlying standards. Interoperability testing is usually driven manually because of the proprietary nature of end user interfaces.

Interoperability tests are applied at interoperability events, where vendors test the ability of their systems for interoperation with systems provided by other vendors and based on the same standards. The basis for each interoperability event is a previously agreed upon interoperability test specification. During the event, implementations of different vendors are plugged to each other and assigned to test sessions. The test sessions are executed in a parallel manner and have usually a specific time limit. Within this limit, it is attempted to execute as many applicable tests as possible. Examples for such interoperability events are the Plugtests^{TM} [30] events that are organized by ETSI. Depending on the concrete interoperability event, customers of the vendors and research partners are also allowed to attend for observing the test sessions.

2.2.2.3 Interoperability Testing with Message Checks

Comparing interoperability testing to conformance testing, each of them has its advantages and drawbacks. Conformance tests alone cannot guarantee system interoperability especially for the application layer. Even if the IUT passes the conformance tests, it does not automatically prove that the IUT is interoperable with other systems implementing the same standard, because the standards may contain implementation options and leave space for interpreting requirement specifications, which can lead to interoperability problems [144]. The benefit of interoperability testing is that it can verify a correct service provision to end users. However, it may require a complex setup, e.g., a *Universal Mobile Telecommunications System* (UMTS) network including the configuration of all involved nodes and does not ensure adherence to standards.

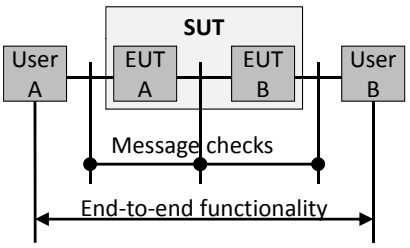


Figure 2.5: Interoperability testing with two EUTs and message checks

In our approach, we use interoperability testing in combination with conformance testing so that it is possible to check the conformance to a standard related to the interoperation and, in addition, to check if the EUTs are interoperable [119, 122]. This approach extends interoperability testing with the monitoring of the communication among the EUTs and updates the approach described by [36] by removing the QE as well as defining the SUT. In the remainder of this thesis, we call this combination of testing interoperability testing with message checks. This means that during the execution of interoperability tests, messages are recorded within test execution traces at (possibly standardized) interfaces between different EUTs by monitors to analyze the compliance of the recorded messages to the standards. This allows the verification of the correctness of protocol procedures, while the assessment of interoperability takes place. Message checks also provide a basis for fault analysis. In contrast to traditional conformance testing, message checks assess requirements that are only related to the interoperation. An interoperability test setup combining interoperability tests with message checks is depicted in Figure 2.5. The end-to-end functionality is assessed from the end user points of view while the message checks take place at the intermediate interfaces. Although this approach is not a replacement for conformance testing, it offers an economic alternative to gain insights about the conformance of equipment participating in an interoperability test to a standard.

Interoperability tests with message checks are also described in the literature using different terminologies. The main idea of combining conformance testing with interoperability testing has been presented, e.g., by [140, 146, 148]. Viho et al. [146] provide a formal framework for interoperability testing. They present a general architecture of interoperability based on lower and upper testers as defined by the international ISO/IEC multipart standard 9646 OSI *Conformance Testing Methodology and Framework* (CTMF) [74]. The CTMF standards define the upper tester and the lower tester strongly related with the OSI model. Interoperability testing with message checks can be used independent of the OSI model.

If interoperability tests with message checks are applied in interoperability events, the validation of standards can be performed in addition to the assessment of interoperability. The results of the tests including interoperability issues as well as discrepancy of the applied standards are reported to the responsible technical committee. This feedback is then used to improve the standards.

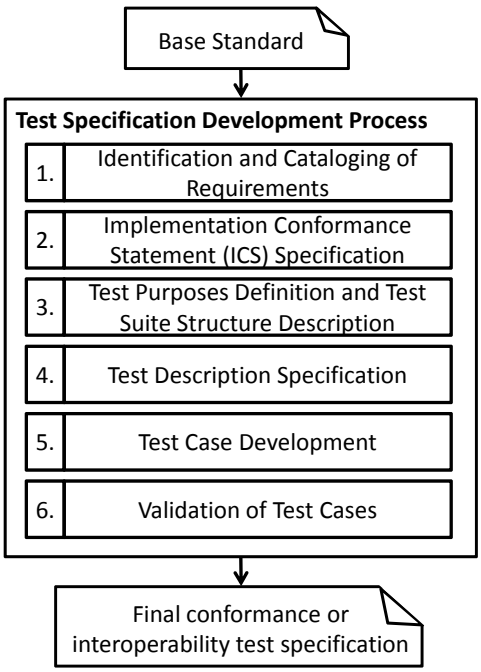


Figure 2.6: Test specification development process

2.2.3 Test Specification Development Process

Based on the test specification development process for *Internet Protocol* (IP) testing established by ETSI [31], we developed the generic test specification development process [118], which is depicted in Figure 2.6. The steps of the process build a bridge over the large gap in the levels of abstraction between a base standard and a final conformance or interoperability test specification.

In Step 1, requirements are identified from relevant base specifications. Requirements may be published in a requirements catalog. Then, in Step 2, the ICS is specified. This step is essentially a high level check list of features and capabilities supported by the IUT. The ICS can be used to quickly identify if two implementations of the same standard have the potential to interoperate. In Step 3, test purposes are specified for the identified requirements and a logical grouping of the test purposes, the *Test Suite Structure* (TSS) is defined. If a requirement can be assessed using a given form of testing then a test purpose specifies verdict criteria for a test. After that, in Step 4, for each test purpose an informal test description is developed. In Step 5, either test purpose-based or test description-based test cases are specified.

The final Step 6 includes the validation of the test cases and is normally not done by the test developers. The validation ensures that the test cases are specified correctly. It may

be done by executing the test cases at an interoperability event or by running test cases by means of conformance test tools against a number of different implementations of a given standard. Problems detected during the validation should be reported to the test developers and can lead to changes in the test case specifications. The validated test cases form the final interoperability or conformance test specification.

2.2.4 Testing and Test Control Notation Version 3 (TTCN-3)

TTCN-3 [43] is an internationally standardized language, which is specifically designed for the specification of tests. It is developed and maintained by the ETSI *Technical Committee for Methods for Testing and Specification* (TC MTS), a team of leading testing experts from industry and research. TTCN-3 has been in use in standardization effort as well as in the industry for over 10 years. TTCN-3 can be applied to a variety of application domains and types of testing. It has been proven to work in very large and complex industrial tests, e.g., for *3rd Generation Mobile Telecommunications* (3G) network elements. There are TTCN-3 test suites for, e.g., IMS, *Long Term Evolution* (LTE), and the *Session Initiation Protocol* (SIP). TTCN-3 can be used not only for specifying and implementing functional tests, but also for scalability, robustness, and stress tests. In this work, we apply TTCN-3 for interoperability testing.

The TTCN-3 language is similar to typical general purpose programming languages' textual syntax. Most 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 ease the specification of test suites.

TTCN-3 supports distributed testing through the notion of test components: *Parallel Test Component*s (PTCs) can be created dynamically in addition to the *Main Test Component* (MTC). Each test component runs concurrently and may, therefore, execute test behavior in parallel to other test components. For the communication between test components and between test components and the SUT, operations such as *send* and *receive* can be used to transfer messages via *ports*. The values of these messages are specified using *templates*. TTCN-3 templates may involve wildcards and provide a powerful matching mechanism to validate expected test data.

Further concepts that ease the specification of tests are: test verdict handling, logging, timeout timers, and defaults. The first three concepts are self-explanatory. 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. Figure 2.7 depicts the TTCN-3 test system archi-

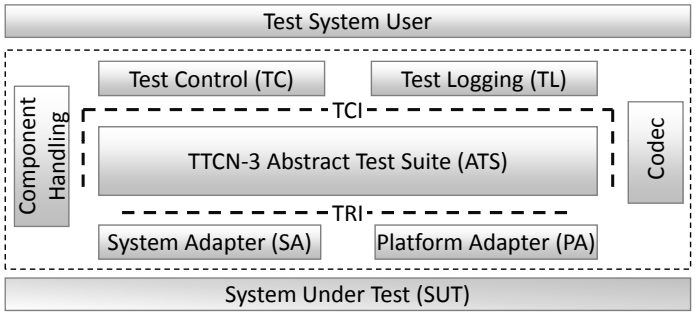


Figure 2.7: TTCN-3 test system architecture

tecture. Test cases are specified in TTCN-3 within an *Abstract Test Suite* (ATS). A *System Adapter* (SA) entity that implements operations of the *TTCN-3 Runtime Interface* (TRI) [32] and a *Coding/Decoding* (CD) entity that implements operations of the *TTCN-3 Control Interface* (TCI) [32, 126] must also be realized. For those ports that are mapped to *Points of Control and Observation* (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 the following advantages in comparison to proprietary test languages or low-level test implementations. The high abstraction level speeds up test development. The re-usability is higher, because both the abstract TTCN-3 test specifications and the adapters can be re-used independent of each other. Furthermore, due to the fact that TTCN-3 is standardized and various TTCN-3 tools are available, a vendor lock-in is avoided. For further introduction to TTCN-3, the reader is referred to [149].

2.3 Systems Under Study

The systems that we studied for interoperability in the case studies are distributed systems. A distributed system is "a collection of independent computers that appears to its users as a single coherent system" [137]. In this section, we describe the systems that we analyzed and applied in the case studies: Web services, compute clusters, grid computing systems, cloud computing systems, and IMS.

2.3.1 Web Services

Most grid and cloud systems commonly leverage Web service technology. According to the *World Wide Web Consortium* (W3C), a Web service is "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface de-

scribed in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" [153]. In the following, we describe five basic Web service standards relevant for this work:

1. ***eXtensible Markup Language* (XML)** [150] is a markup language, that defines a set of rules for encoding documents in a common format to which messages comply that facilitates data sharing across different interfaces of Web services.

2. **SOAP** [151] is a protocol specification that defines XML grammar to allow communication of Web services independent of their utilized platform. SOAP forms the foundation of the Web service protocol stack. SOAP messages are usually transmitted over *Hypertext Transfer Protocol* (HTTP).

3. **HTTP** "is an application-level protocol for distributed, collaborative, hypermedia information systems" [52].

4. ***Web Services Description Language* (WSDL)** [17] is an XML dialect used for the specification of the functionality that is offered by a Web service using XML. With WSDL, methods of a service are described in an abstract and programming language independent way to allow platform independent access.

5. ***Universal Description, Discovery and Integration* (UDDI)** [110] is used to specify an XML-based registry that is utilized for finding Web services. It allows organizations to publish information about their Web services, which can be found and bound by other Web services.

A Web service by itself is stateless, i.e., the Web service cannot remember information, or persist its state, from one invocation to another. The *Web Services Resource Framework* (WSRF) [7] is a specification that provides means to keep the state of a Web service. However, the state is not integrated into the Web service. A separate entity, which is called resource, stores the state information. Each resource has a unique key for its identification and can keep multiple values of, e.g., complex data types. The Web service together with its belonging resource is called a *Web Service Resource* (WS-Resource). The *Endpoint Reference* (EPR) is the address of a WS-Resource [130].

2.3.2 Cluster Computing Systems

Compute clusters are tightly interconnected but operationally independent computers, on which user accessible software runs to manage and control concurrent computing tasks that instantiate a common application program [134]. The independent computers are called

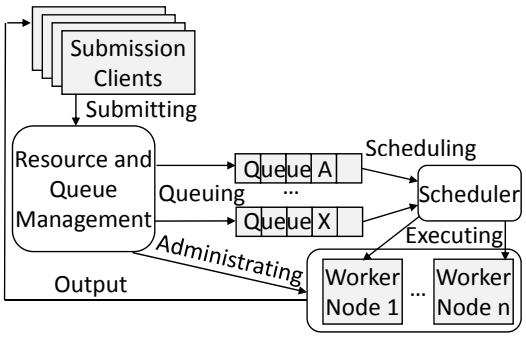


Figure 2.8: Cluster architecture

worker nodes since they execute the actual task. A cluster system contains entities for resource and queue management to control the worker nodes and to implement queues.

A generic cluster architecture is depicted in Figure 2.8. A user submits tasks to the management entity using the submission client. Depending on the characteristic of the task, the task is placed in an appropriated queue. A scheduler then distributes the tasks to specific worker nodes. After the execution of a task is finished, the output is transferred to a network-shared directory. A cluster is usually located in a private network and, therefore, not directly accessible from the Internet.

An open source implementation of a local resource and queue management system is the TORQUE Resource Manager [2], which is based on the *Portable Batch System* (PBS) project [6]. Other implementations include IBM LoadLeveler [2], Oracle Grid Engine [109], and Microsoft Windows *High Performance Computing* (HPC) Server [94]. The interfaces of the local resource and queue management systems are proprietary and not standardized.

2.3.3 Grid Computing Systems

Grid systems allow efficient and dynamic sharing and management of local resources between any interested parties of different organizations. It relies heavily on a grid middleware, which provides secure access to diverse resources that are managed in a decentralized manner. A grid system provides nontrivial qualities of service through standardized, general purpose protocols and interfaces [58].

Referring to the grid architecture defined in [57] and from our practical experiences [118], we developed the conceptual model of grid computing as depicted in Figure 2.9. On the bottom layer, the model includes local resources. In the sense of grid computing, local resources are entities that fulfill job requests [84] and are usually deployed within private networks. A job is usually a description of parallel and computing intensive tasks that are executed on local resources. Grid systems integrate different types of resources including

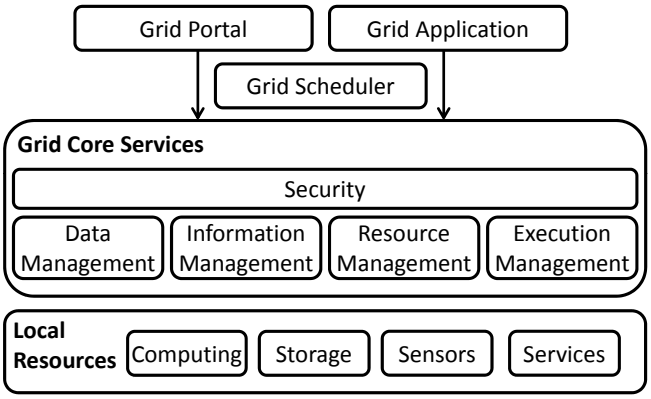


Figure 2.9: Layered conceptual model of grid computing

computing, storage, sensors, and services. These resources usually deploy a predefined software stack. For example, on compute clusters, which belong to the computing resource type, user accessible software runs to control computing tasks as described in Section 2.3.2. Similarly, the other resource types deploy an already pre-configured infrastructure that is usable within private networks by utilization of their specific protocols and interfaces. The protocols are then utilized by the grid core services offered by a grid middleware to access the local resources from a public network and of other organizations. The grid core services include services for the management of information, data, execution, and resources.

The grid core services are utilized by grid schedulers that schedule jobs over several grid infrastructures. In addition, these services are directly usable via grid portals or grid applications. Grid middleware systems deploy security services that provide authentication and authorization functionalities for the entire grid core services.

2.3.3.1 Open Grid Service Architecture (OGSA)

The *Open Grid Services Architecture* (OGSA) [56], which is maintained by the *Open Grid Forum* (OGF) [104], is a *Service Oriented Architecture* (SOA) that defines and standardizes the grid core services for the implementation of a basic grid computing system in an abstract manner [56]. OGSA leverages existing Web service specifications and makes them suitable for the grid environment by adding grid specific characteristics. These grid enhanced Web services are called grid services [91]. The grid core services are largely independent of each other and do not need to be present in an OGSA system. OGSA aims to enable interoperability between heterogeneous and distributed grid systems as well as reduce the administration complexity [56]. OGSA can be extended by other standards that specify specific areas of the grid core services. The extensions include *OGSA-Basic Execution Service* (OGSA-BES) [54], *OGSA-Resource Usage Service* (OGSA-RUS) [101], and *OGSA-Data Access and Integration* (OGSA-DAI) [24].

2.3.3.2 Globus Toolkit 4

Globus Toolkit 4 (GT4) [59] is a grid middleware that provides all required components for the deployment of a grid computing system. It is maintained by the Globus Alliance [63] as a community-based and open-source set of services and software libraries. The toolkit includes components for security, information infrastructure, resource management, execution management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop grid applications [55].

GT4 defines protocols as well as APIs for each component. In addition, it provides open-source reference implementations in C and Java for client-side APIs. A wide variety of higher-level services, tools and applications have been developed based on GT4. Several of these services and tools are integrated in the components of GT4, while others are distributed through other sources [59]. GT4 implements the WSRF and meets the requirements of the OGSA, which both foster interoperability [64].

2.3.3.3 UNICORE 6

Uniform Interface to Computing Resources (UNICORE) 6 is a grid middleware that provides access to distributed computing and storage systems [82]. It is maintained by the Jülich Supercomputing Centre [53].

UNICORE implements a three-layered architecture: the client layer, the service layer, and the system layer [136]. The client layer on the top of the architecture includes three different kinds of clients [83] that can be utilized to access UNICORE resources: the UNICORE command line client; the UNICORE rich client, a graphical user interface based on the Eclipse Rich Client Platform [23]; and the open source *High Level Application Programming Interface* (HiLA) shell that allows development of grid clients using Java.

The core of the architecture is the service layer. It comprises all services and components that are required for accessing a UNICORE grid. They include an authentication service, an information service, a central registry, and a workflow engine. *UNICORE's internal execution management engine* (XNJS) maps the abstract job description to concrete job descriptions for a specific resource, e.g., a compute cluster.

The system layer on the bottom of the architecture includes the *Test System Interface* (TSI) component, which provides the access to the actual resource management or compute cluster system. This means that the TSI translates abstracted commands (from the upper layer) into system specific commands (to the lower layer) [136].

Regarding interoperability, UNICORE supports a variety of standards. UNICORE implements the full Web service stack based on WSRF and allows to access the XNJS via standardized OGSA interfaces for job management. In addition, UNICORE supports the *Grid Laboratory for a Uniform Environment* (GLUE) 2.0 information model and OGSA-ByteIO for data transfer [136].

2.3.3.4 Grid Component Model (GCM)

The provision of common interfaces for the allocation of resources for application deployment in different computing and storage systems is a crucial requirement, because users wish to access multiple resources of several systems simultaneously and in a cost saving way. An approach towards such an interface is described in the ETSI *Grid Component Model* (GCM) standards. The main objective of GCM is the creation of a uniform interface for allocating resources for applications, where resources may be provided across different grid systems. The GCM is an interoperability gateway approach with a standardized and abstract communication protocol based on XML descriptors, i.e., the GCM *Deployment Descriptor* (DD) [33] and the GCM *Application Descriptor* (AD) [40]. GCM DD and GCM AD provide formal specifications of resource information for the involved and possibly heterogeneous systems [34].

2.3.4 Cloud Computing Systems

Cloud computing "is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [92]. Cloud computing systems fulfill five essential properties [92]:

- **On-demand self-service**: Consumers can instantiate computing, storage, and network capabilities in an automated manner according to their demand;
- **Broad network access**: The capabilities can be used and accessed by common interfaces over the network by any kind of client platform;
- **Resource pooling**: Physical resources are pooled dynamically into virtual resources, which are utilized in a multi-tenant manner having a sense of data location independence without having control or knowledge about the exact data location;
- **Rapid elasticity**: Resources can be allocated and released rapidly (in the orders of minutes) according to the demand;
- **Measured service**: Cloud systems utilize a metering capability to control and optimize resources automatically [92].

Cloud systems are classified in a layered service model containing the following layers from bottom to top as depicted in Figure 2.10: *Infrastructure as a Service* (IaaS), PaaS, and *Software as a Service* (SaaS) [92]. Within the illustrated clouds on each level, the figure depicts the interfaces to the services that are provisioned by each layer, respectively. IaaS clouds include virtualized resources, e.g., storage, processors, and networks. Within the virtualized resources, network architects are able to deploy and run arbitrary software via resource management interfaces. Network architects check the status of the IaaS clouds

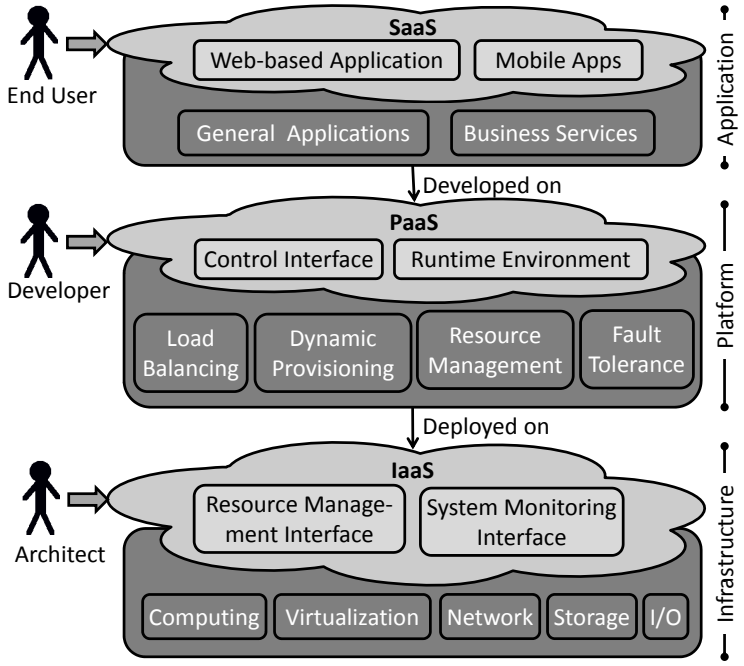


Figure 2.10: Service model of cloud computing

via system monitoring interfaces. On the PaaS level, services for automated resource management, fault tolerance, dynamic provisioning, and load balancing are deployed. These functionalities are utilized within the runtime environment in a transparent manner via a control interface, e.g., an *Application Programming Interface* (API) that is used by an application developer. The developer only has control over the deployed applications, but not over the resources. The SaaS layer provides Web interfaces for end users to access applications without requiring local software installations. The users are only able to apply application specific configurations, but cannot control the cloud infrastructure. Each layer deploys security mechanisms to protect the resources and services they offer.

A cloud system is deployed in one of the following models: private, public, hybrid, and community [79, 92]. Public clouds offer services for public use and are physically located on the premises of the cloud provider. Public clouds serve multiple customers using a multi-tenant model. Private clouds can be hosted on the premises of a single organization but also off the premises managed by a cloud provider. The access to a private cloud is dedicated and restricted to a single customer. A hybrid cloud is the mixed employment of private and public clouds, e.g., to maintain control over sensitive data, which is only stored in the private cloud [79] while also being able to use the computational power of a public cloud. In addition, community clouds share their infrastructure by several organizations and support a specific community that has shared concerns, e.g., a mission, security requirements, policies, and compliance considerations [92].

2.3.4.1 Amazon Web Services (AWS)

Amazon Web Services (AWS) [3] offers a public IaaS cloud system. AWS provides various Web services to access their cloud infrastructure and to allocate cloud resources on demand. Basic AWS services are the *Simple Storage Service* (S3) that provides storage in the cloud, the *Elastic Compute Cloud* (EC2) service that delivers compute capacity in the cloud, the *Simple Queue Service* (SQS), which is a messaging service over HTTP, and the SimpleDB service, which is a non-relational database store service. For a complete list of the cloud services that are offered by AWS, the reader is referred to [87]. All AWS services can be integrated and used in a complementary way to build an AWS cloud application. The physical architecture of AWS has not been published, yet.

2.3.4.2 Eucalyptus

Eucalyptus [100], which is maintained by Eucalyptus Systems [46], is an open source IaaS cloud software to build private clouds. A Eucalyptus cloud is based on Web services and consists of the following components: node controller, cluster controller, cloud controller, storage controller, and Walrus. The node controller manages, e.g., starts and stops one or more *Virtual Machine*s (VMs) on the physical machine, on which the node controller is installed. The cluster controller acquires information about node controller sets and schedules instantiations of VMs on specific node controllers. The cluster controller controls only the node controllers that are deployed in their subnet. The cloud controller is the entry point to the Eucalyptus cloud. It gathers information about resources from the node controllers and sends high-level scheduling requests to the cluster controllers [100]. The storage controller provides functionalities to attach storage volumes to VMs. However, storage volumes cannot be shared between VMs, but they store data persistently, i.e., the storage volumes persists after the VM is terminated. Walrus provides mechanisms to store VM images and user data persistently, which are accessible by all VMs as well as externally by a client [47].

The main advantage of the Eucalyptus cloud compared to the AWS cloud is the controllability of the physical resources. The Eucalyptus cloud allows the deployment of a private cloud. However, the Eucalyptus cloud infrastructure needs to be maintained and updated regularly by the deploying organization, which is not the case in the AWS cloud.

2.3.4.3 Google App Engine (GAE)

Google App Engine (GAE) [65] offers a public PaaS cloud system to allow the development and execution of scalable cloud applications on Google's infrastructure. GAE includes the following cloud features: persistent cloud storage, automatic scaling and load balancing, and dynamic Web serving.

GAE supports and provides runtime and development environments for Java, Python, and Go [68]. Depending on the programming language choice, GAE offers APIs for various cloud services, e.g., a memory cache service for distributed in-memory data called

memcache, prospective search that allows matching of a large set of queries simultaneously against a stream of input documents, and capabilities that allow the detection of outages and scheduled downtime of API capabilities [66]. Depending on the specific requirements of a GAE application, GAE offers three different cloud storage options [68]:

1. **GAE Datastore**, which is a NoSQL schema-less object datastore,
2. **Google Cloud SQL**, which is a relational SQL database service,
3. **Google Cloud Storage**, which is a storage service for objects and files up to terabytes in size.

2.3.5 Internet Protocol Multimedia Subsystem (IMS)

IMS [1] is a specification for a telecommunications service. Telecommunications is "the transmission, between or among points specified by the user, of information of the user's choosing, without change in the form or content of the information as sent and received" [50]. A telecommunications service is "any service provided by a telecommunication provider" [99], or "a specified set of user-information transfer capabilities provided to a group of users by a telecommunications system" [99]. This means that when a telecommunications service is used, the user of the telecommunications service is responsible for the content of transmitted messages while the provider of the telecommunications service is responsible for the acceptance, transmission, and delivery of the messages created by the user [99].

IMS provides services beyond voice calls that offer the capability to share video and media content between users. IMS is based on the *3rd Generation Partnership Project* (3GPP) standard and is one of the key enablers of the next generation networks [113]. IMS relies on an IP based peer-to-peer architecture, which is split into user, control, and service layers. The signaling of IMS is mainly based on SIP [123], which is a text based signaling protocol on the application level. SIP is used to set up, modify, and terminate real-time sessions between users over an IP network [133]. SIP enables clients to invite other clients to a session and negotiate control information about the media channels needed for the session, which are required for IMS.

Figure 2.11 shows a simplified IMS network, which provides *Call Session Control Function*s (CSCFs) that are basically SIP servers or proxies used to process SIP signaling packets in the IMS. Users access the IMS network with a *User Equipment* (UE), e.g., a mobile phone, and connect through the *Gm* interface to the IMS network entry point. This is the *Proxy-CSCF* (P-CSCF), which provides subscriber authentication. A P-CSCF is connected to a *Serving-CSCF* (S-CSCF), which is a SIP server that also performs session control. For example, an S-CSCF handles SIP registrations and provides routing services. The IP address of an *Interrogating-CSCF* (I-CSCF) is published in the *Domain Name System* (DNS), where remote servers find the I-CSCF and use it to forward SIP packets. In

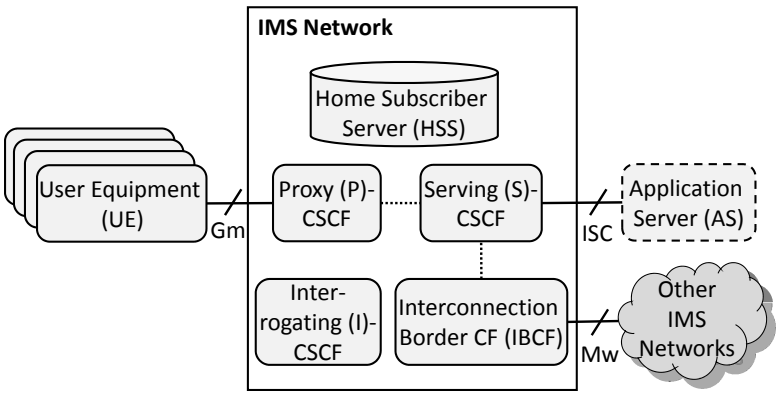


Figure 2.11: IMS network architecture

addition, *Network Address Translation* (NAT) and firewall functions are provided by the *Interconnection Border Control Function* (IBCF) so that the IBCF can be used as a gateway to external networks through the *Mw* interface. The application server deploys services for execution and is accessed via the *ISC* interface by the S-CSCF.

3 Assessment and Improvement of Interoperability

Interoperability is a prerequisite to allow users to access systems implemented by different vendors seamlessly. Interoperability needs to be implemented in a reliable manner to meet customers' requirements regarding systems' interoperation. To this aim, we present a structured way to assess and to improve interoperability of systems in this chapter. We establish the *Interoperability Assessment and Improvement* (IAI) process. We give an overview of the process in Section 3.1 and describe the details of the process' phases in sections 3.2– 3.4. We conclude this chapter with related work in Section 3.5.

3.1 Process Overview

A process is "a set of interrelated activities, which transform inputs into outputs" [75]. In addition, a process defines roles that are associated with activities and output documents. An overview of our IAI process is depicted in Figure 3.1. The IAI process provides a structured way to analyze systems for interoperability opportunities and implement interoperability solutions. The process consists of three phases: investigation of the fulfillment of prerequisites for interoperability (Phase I), improvement of interoperability (Phase II), and assessment of interoperability (Phase III).

Phase I includes the investigation of the fulfillment of prerequisites for interoperability of systems. These prerequisites include the existence of common and complementary functionalities, requirements on the system's architecture, and the need of interoperability.

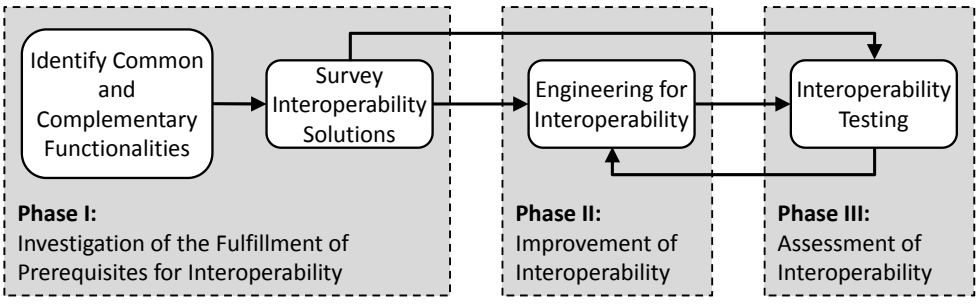


Figure 3.1: Generic process for assessing and improving interoperability of systems

Furthermore, Phase I contains the identification of involved standardization organizations, interoperability initiatives, and already available interoperability solutions for the systems under study.

Depending on the results of Phase I, the IAI process is either continued with the improvement of interoperability (Phase II) or with the assessment of interoperability (Phase III). If no interoperability solutions have been identified, the process continues in Phase II, where an interoperability solution will be engineered. As soon as an interoperability solution is available (either identified in Phase I, or engineered in Phase II), it needs to be assessed in Phase III by means of interoperability testing.

We identified the following roles required for the IAI process: the interoperability analyst, the interoperability engineer, and the interoperability tester. The interoperability analyst is responsible for Phase I and identifies if the prerequisites of interoperability are fulfilled by the involved systems. The interoperability engineer is responsible to implement or to improve interoperability solutions of the involved systems in Phase II. The interoperability tester develops and executes an interoperability test suite in Phase III to assess the implemented interoperability solution. The results are analyzed by the interoperability engineer in Phase II.

The process is generically applicable, e.g., to homogeneous and heterogeneous architectures. Depending on the analyzed architecture, different techniques need to be applied to establish as well as to assess their interoperability. The procedure, required documentation, and the content of the different phases of the IAI process are described in detail in the subsequent sections.

3.2 Phase I: Prerequisites for Interoperability

Prerequisites for interoperability are the existence of common and/or complementary functionalities of the systems that are required to interoperate. A common functionality is the base for interoperation of systems while a complementary functionality is the base for their interworking. We consider interworking as a type of interoperability and do not distinguish between the terms.

In Phase I of the IAI process, the interoperability analyst identifies if the prerequisites for interoperability are fulfilled by conducting an analysis to determine the current state of the interoperability between the involved systems. The analyst evaluates the architecture of the involved systems to determine common and complementary functionalities, which are identified based on standards, manuals, and descriptive documents of the involved systems. These documents help to identify protocols and interfaces for the access to a system's functionality. The interoperability analyst needs to be an expert of the involved system to determine if the prerequisites of interoperability between the involved systems are fulfilled. If no common or complementary functionalities are identified, interoperability is not achievable as well as not reasonable. Otherwise, the analysis report of the interfaces of the

functionalities serves as input for Phase II and builds the foundation of the interoperability engineering.

To report the results of this phase, the analyst prepares a document that describes each externally accessible functionality of a system with at least the following attributes: name, description, interface information, communication protocol including data formats of input and output data, specification reference, and dependencies to other functionalities. From such a list, the identification of interoperable functionalities between different systems can be facilitated by pairing common attributes. In addition, the dependency relation attribute is a hint for the identification of complementary functionalities if the other involved system provides the depending functionality as well.

In addition to the identification of the common and complementary functionalities, a survey on already available approaches for achieving interoperability of the involved systems should be done. This includes the identification of involved standard organizations, interoperability initiatives, and related research. We describe the approaches for achieving interoperability in the next section. They can be based on standards or interoperability gateways. Identified interoperability approaches for the involved systems should be documented in a report. This report is input for Phase III of the IAI process, where the quality related to the interoperability of the approaches is assessed.

From the existing interoperable functionalities and from the results of the survey on already available interoperability approaches, interoperability gaps can be derived. The gaps should be reported by the interoperability analyst so that the interoperability engineer can fill them in Phase II.

If no interoperability approaches exist yet, the benefits of achieving interoperability of the involved systems need to be determined and analyzed. Benefits for interoperation include facilitating migrations, gaining a greater business value by provision of high level services and service enrichment, as well as increasing the productivity by leveraging idling resources. The analysis should also include an estimation of the required effort to realize an interoperability solution, as this needs to be weighed against the benefits. The analysis if it is worth to implement an interoperability solution needs to be documented in a report. This report and the report about the identified common and complementary functionalities are inputs for Phase II of the IAI process, where an interoperability solution for the involved systems is developed.

3.3 Phase II: Improvement of Interoperability

Based on the outputs of Phase I, the interoperability engineer designs and develops an interoperability solution. Several approaches for achieving interoperability between computing, service, and storage systems exist. In general, these approaches are classified in interoperability gateways and standardized interfaces [34].

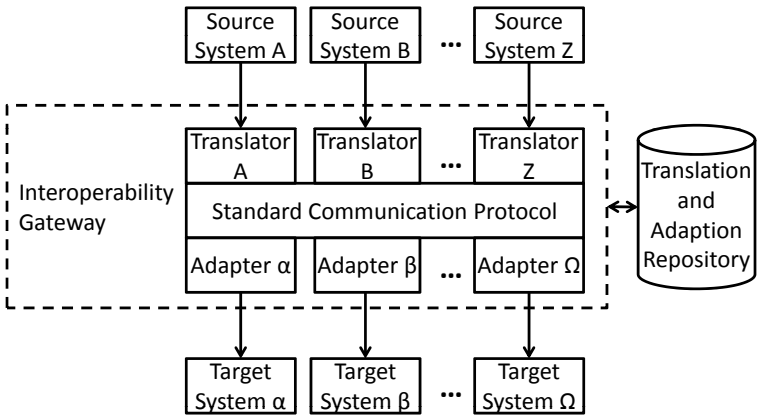


Figure 3.2: Interoperability gateway

An interoperability gateway contains several translators and adapters as depicted in Figure 3.2. Interoperation takes place on the same level of abstraction, i.e., implementations need to adapt to the same communication protocol. The translator transforms data from a source system into a common and agreed standard communication protocol, e.g., an XML scheme to allow systems using different protocols to be connected to the interoperability gateway. The adapter converts the standard communication protocol into the specific protocol used by the target system and communicates the information to the target system. Note that the roles of source and target systems are not disjunctive. If a translator and an adapter are both available for a system, this system can be both source and target, i.e., send and receive information. In a one-to-one scenario, it is possible to translate and adapt directly into the required protocol of the target or source system instead of into an agreed intermediate representation. This is commonly referred to as an adapter. The data of the involved protocols and the translation schemes can be stored in a translation and adaptation repository that can be accessed for translating purposes. Figure 3.2 shows a many-to-many scenario, where the interoperability gateway comprises all involved translators and the adapters.

Interoperability gateways should be considered as interim solutions, as they do not scale well [51]. If the number of systems increases, the interoperability gateway performance decreases. It is an expensive approach, because for each protocol, a translator and an adapter need to be developed and integrated. Therefore, interoperability gateway solutions are not viable in ad-hoc scenarios or emergency cases.

The long-term approach to address interoperability is the use of open and standardized interfaces. The interfaces that need standardization can evolve from the interoperability gateway deployment since the mapping to different infrastructures has already been identified. However, the drawback of this approach is that the different involved stakeholders agree to a common set of standard interfaces that also meet production system requirements. However, standardization can enable interoperability in a multi-vendor, multi-network, and

multi-service environment without the scalability problems of the interoperability gateway approach.

If the interoperability solution is based on an interoperability gateway approach, the interoperability engineer needs to describe and specify the interoperability gateway including translation rules and data formats adequately in a specification report, which is input for Phase III. If the solution is based on standards, the standards' documents are input for Phase III. There are various reasons why the interoperability of implementations of the same standards needs to be assessed in Phase III. Standards can be specified unintentionally incomplete so that features required for interoperability are not or only partly defined. Relevant interfaces required for interoperability might not be clearly defined. This means that the standards can be interpreted differently. In addition, consequences of options may not clearly be stated, or if too many options are specified, inconsistencies between them can arise [144]. This means that programmers can choose different kinds of options, which are then not understood by the other implementation, for which these options were not implemented. In addition, different versions of the standard might be incompatible. The standards or respectively the specification report build the basis for the development of the interoperability solution but also for the specification of interoperability tests, which assess the developed solution.

3.4 Phase III: Assessment of Interoperability

In Phase III, the interoperability tester assesses developed interoperability solutions of systems by the application of interoperability testing with message checks, which we present in Figure 2.5. This assessment is based on one of the following documents: the report about identified interoperability solutions developed by the interoperability analyst in Phase I, standards identified by the interoperability engineer in Phase II, or the specification report of an interoperability gateway developed by the interoperability engineer in Phase II. These documents are used by the test designer [132] to develop the test architecture and test descriptions, which are both described in detail in Section 2.2.1. The test architecture builds the base for the development of test configurations. The test descriptions are extended with the attribute of conformance criteria to specify the conformance test purposes. Together with the other attributes, this then leads to interoperability tests with message checks. The test sequence specified in the test description is then executed by the tester [132] in a manual manner. In Chapter 4, we present a framework to assess interoperability in an automated manner.

When executing interoperability tests with message checks, a large amount of information is gained during the tests due to the number of interfaces that are monitored. In addition, conformance testing takes place on the level of the EUTs while interoperability testing takes place on the level of the SUT. Because of these different abstractions, we separate the conformance and the interoperability test verdict management at the level of test descriptions. This speeds up troubleshooting and identification of wrong behavior of the SUT.

Interoperability verdict	**Conformance verdict**	**Evaluation**
Pass	Pass	EUTs interoperate and communicate according to the standard.
Pass	Fail	EUTs interoperate, but do not communicate according to the standard. This can be due to a problem in the base standard, the test system, or one of the EUTs, e.g., a mandatory check is used optionally.
Pass or Fail	Inconclusive	EUTs do or do not interoperate; all captured communication conforms to the standard, but at least one interface with message checks was not available for evaluation.
Fail	Fail	EUTs do not interoperate and do not communicate according to the standard. This can be due to a problem in the standard, the test system, or one of the EUTs.

Table 3.1: Summary of the verdicts' interrelations related to a single test description

First, we determine the interoperability verdict, which is based on the observation of the behavior of the SUT at its end points for a single test description. The interoperability verdict can be either *pass* or *fail*. A *pass* means that the EUTs interoperate when providing end-to-end functionality, while a *fail* means they do not provide end-to-end interoperability. The interoperability verdict is set for each test description and is related to the SUT.

Secondly, we determine the conformance verdict, which is based on the observation of protocol procedures and messages exchanged at interfaces between EUTs of a single test description. It can be either *pass*, *fail*, or *inconclusive*. A *pass* means that the test shows conformance to the normative requirements tested by the associated test case. In case of a *fail*, the test shows non-conformance to at least one of the normative requirements tested by the associated test case. *Inconclusive* means that neither a *pass* nor a *fail* verdict can be given, because the available information is insufficient. The conformance verdict is set for a single test description and relates to single EUTs.

Both verdicts pursue different purposes and evaluate different and orthogonal aspects: interoperability or conformance. Hence, it is not possible and not reasonable to determine a single verdict that combines both aspects. Table 3.1 shows different possible scenarios in verdict observations and potential errors for single test descriptions. For the assessment, verdicts alone do not allow direct identification of the cause of a test failure. Resolution of test failures is generally non-trivial and requires troubleshooting.

Each test step in an interoperability test can lead to an intermediate verdict. Inserting a capture of intermediate verdicts for each significant event in a test can help to speed up the evaluation of the test execution results. The verdict of a single component can also be determined while the test runs.

Interoperability tests with message checks are applied at interoperability events that provide opportunities for vendors of systems to assess and demonstrate the interoperability with systems from other vendors directly. A reference implementation is not required. However, not only the implementations, but also the interfaces are evaluated for possible improvement. The developers and vendors give feedback to the interoperability tester. The interoperability tester collects all the information and results of all executed tests from the participating vendors. All executed tests, their results, and conclusions drawn from the results are documented. This report is input for Phase II to allow further improvements of the interoperability solution and interoperable interfaces. The assessment and the improvement build a circle in the process.

If the interoperability solution is based on the interoperability gateway approach, the report with the test results is used by the interoperability engineer to improve the interoperability gateway. If the interoperability solution is based on standards, the report can be committed to the committee in charge to maintain the tested standard. The report also contains the results of the message checks, which validate if the systems follow the standards related to the interoperability assessment. The interoperability of two systems is improved indirectly through the message checks by the identification of ambiguousness of the involved standard, e.g., by assessing two different implementations of a message specified ambiguously in the standard. After the responsible committee removes the ambiguousness from the standard, updates of system implementations, as required in the updated version of the standard, improve the interoperability of the systems. Therefore, the standards themselves can be assessed by message checks. The updates are applied and developed by the interoperability engineer based on the updated standards.

3.5 Related work

The European Commission presents an interoperability framework for the European public services [48]. Within this framework, they describe twelve principles in the context of deciding and implementing European public services including accessibility, security, and multilingualism. Related to our work, they give recommendations on implementing public services in an aggregated way. This means that the public services of the member states need to be interoperable. The European Commission provides abstract guidelines on achieving interoperability between public service implementations of different European Union member states. We provide means and measures of assessing interoperability of systems. The interoperability assessment phase of the IAI process can be applied to assess the interoperability of the public services of the European Union.

Sayogo et al. analyze the implementation of an interoperable data architecture providing trusted product information to help consumers with their choices in purchasing sustainable food products [125]. This study focuses on organic and fair trade coffee and analyzes the different stakeholders involved in the supply-chain. The stakeholders are supposed to pro-

vide the product information via interoperable means. Hence, their work mainly focuses on analysis of organizational interoperability. However, the background for technical interoperability is described only in a schematic way. Their approach implements an interoperability gateway to connect different stakeholders with each other. They map proprietary data formats of the stakeholders' systems to a generic format based on an XML scheme. Our IAI process focuses on the assessment and improvement of interoperability in general on the technical level. Their approach is similar to Phase II of the IAI process, but it is not clear how they analyze the systems of the stakeholders. Such an analysis is similar to Phase I. However, they focus on the stakeholder analysis based on the five-step process described by Bunn et al. [13].

Bhuta and Boehm present a framework for identification and resolution of interoperability mismatches in *commercial-off-the-shelf* (COTS) products [11]. They model a COTS interoperability assessment framework based on three key components: attributes that define interoperability characteristics of COTS products, assessment rules for interoperability, and a COTS interoperability evaluation process. They describe the successful application of their framework in an empirical study. The application of their framework is similar to the Phase I and II of our IAI process, but does not replace the need for thorough interoperability testing, i.e., Phase III of the IAI process. Ma et al. assess interoperability of components using translation rules that also consider the goals of stakeholders based on interoperability requirements [89]. In their approach, declarative requirements of components are compared to the stakeholders' goals, which are both translated or respectively defined in the declarative language called Alloy [78]. They focus on reusability of components and build a model, from which conclusions about the interoperability of the components can be drawn. We assess interoperability by the application of interoperability tests with message checks. Our approach can be applied on system level.

Pokraev et al. describe a conceptual framework for modeling services and their interfaces [114]. The model is the basis for the assessment if a system composed of services meets their identified requirements for interoperability. The main issue of this approach is that already implemented systems are not specified using their modeling notation. In addition, the results of the assessment are only yes-no answers to their requirements. Our interoperability assessment approach also assesses the messages. Furthermore, the application of the IAI process is not limited to service compositions. Fang et al. also present an interoperability assessment model for service compositions [49]. Their assessment model considers the different levels of interoperability, for which they define interoperability in a mathematical way. However, they only consider assessing interoperability of service compositions. Similar interoperability assessment of service compositions is also done by Quartel and van Sinderen [115]. Both works only consider the interoperability of homogeneous systems. Our IAI process can be applied to homogeneous as well as heterogeneous systems. In addition, we focus on interoperability of already implemented systems.

Interoperability tests are usually applied in an ad-hoc fashion in interoperability events, where the participants are usually known only weeks before. With Phase III of the IAI process, we provide means that can be applied in interoperability events organized by, e.g., ETSI [30] or *Open Mobile Alliance* (OMA) [106].

4 A Methodology for Automated Assessment of Interoperability

A strategic requirement in a competitive environment is the reduction of the time to market for a new system, service, or architecture. An essential and feasible practice to decrease the time to market of a system is to apply automation at several development steps. Automation considerably helps to minimize the time for testing and also avoids repetitive manual activities, which are prone to error. Applying test automation techniques usually reduces the usage of the test equipment and EUT resources, decreases the acceptance costs both for suppliers and for customers, and limits manual interaction related to test execution, e.g., for the analysis of test execution traces including message contents and reports. Compared to manual interoperability testing, the application of our methodology has the following other benefits: wider test coverage, consistent test execution, and repeatable interoperability tests.

However, assessing the interoperability of systems is usually done in a manual and ad-hoc fashion with interoperability testing. Therefore, in this chapter, we present a methodology for automated assessment of interoperability. The methodology comprises four main parts. First, a generic environment for interoperability tests with message checks (Section 4.1), which builds the basis for a development of automated interoperability tests. Second, guidelines for interoperability test design and test automation (Section 4.2). Third, a generic library for automated interoperability tests using TTCN-3 (Section 4.3) that implements the generic environment as well as the guidelines. Fourth, a generic development process for the systematic specification of a complete and structured automated interoperability test system with message checks (Section 4.4). This methodology provides a first step to a formalized and systematic assessment of systems' interoperability in an automated manner. It can be applied in Phase III of the IAI process to facilitate the assessment of interoperability, which we describe in Section 3.4. This chapter is partially adapted from [8, 122].

4.1 A Generic Environment for Automated Interoperability Tests

As a basis for the development of automated interoperability test systems, we present a generic environment for automated interoperability tests, which we adapted from the methodology presented in [36]. The generic interoperability test environment is depicted in Figure 4.1. It contains the SUT, which is composed of two or more EUTs, an interconnecting network, optionally one or more application support nodes, and different means of

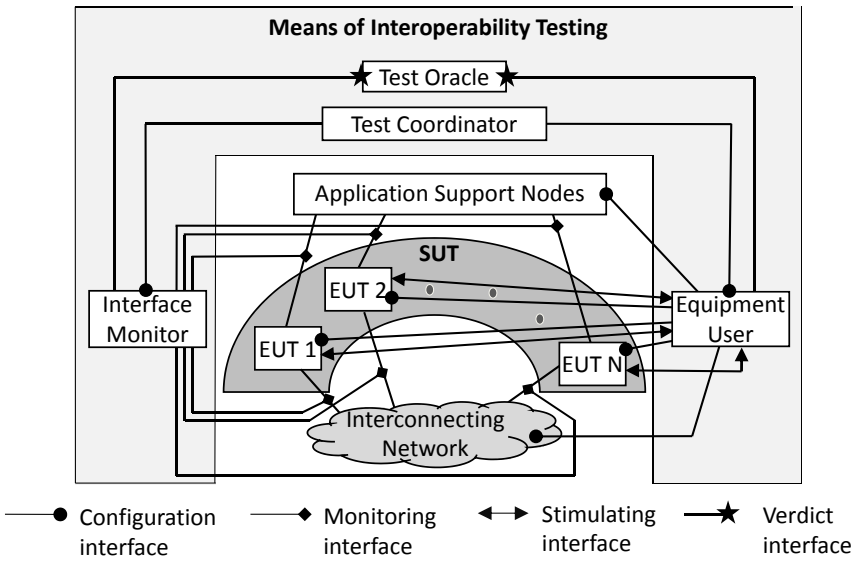


Figure 4.1: Generic interoperability test environment

interoperability testing. We designed the environment independent of the SUT, as well as of the test language used to implement the means of interoperability testing.

For the interoperation of the tested equipment, the interconnecting network and application support nodes are essential. They usually belong to the service that is provided to the end user. However, they are neither part of the SUT nor of the means of testing. Application support nodes include all the devices involved in providing the service or functionality to the end user, but which are not object of the test, e.g., a database with user data. The interconnecting network includes all the devices needed to provide the required network connections. These devices do not need to be tested, because we assume that they were previously tested and properly configured with an initial configuration. However, their initial configuration can be changed during test execution.

The means of interoperability testing allow the structured specification of interoperability tests with the focus on test automation and are comprised of four test entities: the equipment user; the interface monitor; the test coordinator; and the test oracle. The equipment user primarily acts as the end user of a service or functionality, but can also configure EUTs, the interconnecting network, and application support nodes. The equipment user is mainly used to trigger the interoperability tests in an automated way from end to end user. The interface monitor checks messages and protocol procedures on communication links for their conformance to a standard during the execution of an interoperability test. The test coordinator coordinates and synchronizes all instances of the interface monitor and the equipment user involved in a test execution. The test oracle collects all verdict information from the instantiated interface monitor and equipment user to resolve the interoperability and conformance verdicts of the executed test.

4.2 Guidelines for Specifying Automated Interoperability Tests

The following guidelines contain two main aspects that need to be followed during the specification of automated interoperability tests: the adherence to test design guidelines and crucial aspects that need to be considered for test automation. The application of the guidelines help the interoperability tester to structure and to automate interoperability tests.

4.2.1 Test Design Guidelines

We provide design guidelines for developing interoperability tests to keep the interoperability test specification consistent and maintainable. These guidelines include concepts that are already used in common software development. Therefore, in the following, we distinguish between common guidelines and guidelines specific for automated interoperability testing. The common test design guidelines are:

- The **application of naming conventions** increases code readability, consistency, and maintainability. Naming conventions facilitate the detection of semantic errors and allow a better understanding of the code for a distributed development team.
- **Using an identical programming language version** for the development of the test suite avoids conflicts of concepts available in different versions.
- The **modularization of the test suite** enables the reuse of definitions for interoperability testing, which should be isolated from definitions specific to the test suite. The test suite should be designed in libraries [127].
- The **use of functions** clearly separates and isolates behavior specific to the test suite, interoperability testing, specific protocols, and the SUT to their respective libraries. This functional design approach maximizes the reuse of functions.
- **Documentation** of the source code of the test suite makes it easier to understand. The documentation should be independent of *Integrated Development Environment*s (IDEs). For this purpose, standardized documentation tags [42] should be used.

The test design guidelines specific for automated interoperability testing are:

- The **use of test case orchestration** by specifying multiple test components in a test suite. Each test component handles different and independent tasks. The test components are only used if they are required. For example, one test component should be specified for the configuration of the equipment during the preamble and another one for monitoring an interface during the execution of the test body.
- The **application of message template design** allows the extension of message types with additional message fields with minimal changes of the affected template. It can be identified by analyzing the protocols and interfaces implemented by the SUT. The design approach allows the extension of message types with additional message

fields with minimal changes of the affected template. All other related templates do not require updates. The adoption of this design approach in template additions to the test suite helps to improve the maintainability and readability of the test suite.

- The **abstract handling of proprietary interfaces** allows their common and transparent access. Equipment operations should be implemented in an abstract way based on a command request and response paradigm. Commands are abstract descriptions of actions to be taken, e.g., "enter a contact" or "initiate a *Voice over Internet Protocol* (VoIP) call". Abstract primitives can have abstract parameters, e.g., identifier of the terminating user. The interfaces are then translated in the lower adaptation layer.
- The **implementation of message skipping** avoids analyzing messages that are insignificant for the test. Message skipping is required if complex messages that are part of a longer message exchange on the monitored interface need to be checked. Dispensable messages may appear in between and can be part of the preamble or other unanticipated traffic that offsets the message observation from its anticipated occurrence in the test description call flow. The beginning of the sequence to be analyzed needs to be located, which means that all preceding messages are skipped. This issue is unique to interoperability testing.
- The **management of EUT interface information** is needed as part of the test system, because the configuration of a test suite with EUT interface information can be a laborious and tedious task. Also, it is possible that the interface information changes during different interoperability tests. Management of EUT interface information facilitates updates of interface information quickly without introducing errors. The management should be generic to make it applicable for interfaces of different EUTs. It should be compilation independent so that information changes are applied without rebuilding or recompiling the test system.

4.2.2 Test Automation

A typical application of test automation is regression testing, where the tests need to be re-executed, because a functionality of the SUT was changed or added. In case the test execution is automated, test experts can concentrate on the evaluation of test reports. During interoperability events, test automation is also applied for the evaluation of test results to highlight critical parts. This reduces the manual assessment of the test results and facilitates the trace analyses for executions of the same test for different vendor pairings.

For test automation, the test interfaces need to be available and accessible so that they can be used by the tests to stimulate, to observe, and to monitor the SUT. Ideally, these interfaces are open and standardized to facilitate their access independent of vendors.

Before test automation is applied, a cost/benefit analysis must be performed to identify to which degree the application of automation of interoperability test case execution is suitable and acceptable in terms of development costs and effort. This means that limitations

regarding the test automation need to be identified. To this aim, several aspects regarding the SUT, including its topology, involved protocols, the complexity of the protocols, and the stability of the standards are considered. In addition, required resources to develop and execute the automated interoperability test system, e.g., manpower, hardware devices, and software are identified. The required resources are then adjusted to the available resources, e.g., by increasing or decreasing the degree of test automation.

However, the degree of test automation is often a compromise between testing requirements and feasibility. The development of a fully automated test system and the entire testing process may not be profitable since the resources required to implement all parts of the tests need high effort and are, therefore, prohibitively expensive. There are cases, where interfaces are easier accessible in a manual than in an automated manner. This is especially the case if proprietary interfaces are involved in the test.

Different activities of interoperability testing can be automated:

- configuration of the EUTs and the interconnecting network,
- monitoring of relevant interfaces between EUTs,
- validation of the EUT communication,
- simulation or emulation of equipment (can be external),
- operation of all equipment involved in an interoperability test, e.g., EUTs,
- computation of test verdicts,
- execution of tests, and
- generation of test reports.

Ideally, all the tasks are automated. The degree of test automation depends on the identified limitations and on the test execution environment. For example, in interoperability testing in the context of interoperability events, an emulation of complex interfaces of the EUT can lead to the introduction of interoperability problems or even mask errors that exist in the EUT. Therefore, real equipment should be used and its operation should ideally be automated. In practice, automating all the activities mentioned above is rarely achieved. Often, manual checks and actions complement automated test steps.

To create a suitable test system, it should be possible to control the degree of automation related to the EUTs. The test system should be adaptable to the limitations of different equipments. Therefore, it should be possible to adjust the interfaces that are monitored and the interfaces that are operated for each test according to the SUT. In our approach, test automation is controlled during the tests by setting the tests to live or to offline mode. The differences between both modes are depicted in Table 4.1. In the live mode, the analyses of messages are performed automatically on live capture during testing. The operation of an equipment can be done either manually or automated. In the offline mode, relevant traffic of all interfaces is captured in traffic capture trace files and the analyses of messages are done automatically after test execution. Therefore, automated equipment operations are disabled for test executions in the offline mode.

Capture Mode	offline	live
Message checking	Performed on traffic capture files after testing	Performed on live capture during testing
Equipment operation	Disabled	Manually or automatically

Table 4.1: Live and offline modes

4.3 TTCN-3 Library for Automated Interoperability Tests

We developed the TTCN-3 *Interoperability Test Library* (LibIot)[2], which implements the generic environment for automated interoperability testing and applies the test design guidelines. We choose TTCN-3 to specify the test suite, because TTCN-3 supports the definition of concurrent tests, allows dynamic test configurations, and implements a rich type system, which includes concepts like verdicts and native list types.

LibIot follows our test design rules and is the basis for test component types, test interfaces, and test parameter definitions of automated interoperability tests. LibIot is used together with other TTCN-3 libraries, e.g., the LibCommon and the LibUpperTester. The LibCommon contains basic definitions for test suite implementation, e.g., type definitions, verdict handling, timing, test components, and synchronization. The LibUpperTester is a collection of reusable TTCN-3 definitions that are related to upper tester specification for conformance and/or interoperability testing including an abstract equipment operation protocol. Figure 4.2 shows the library dependencies of a test suite, called SpecificATS, which is an ATS that imports the generic libraries and specifies test cases specifically related to the SUT. The protocols that are utilized within the SUT should also be developed in libraries (LibSpecificProtocol) to allow their reuse for other systems based on the same protocols.

LibIot implements the basic functionalities of the means of interoperability testing, i.e., the test coordinator, the equipment user, the interface monitor, and the test oracle, as depicted in the generic environment for automated interoperability tests in Figure 4.1. Even though we use TTCN-3, both the generic environment for automated interoperability testing and the library are designed in a generic way and are, therefore, applicable in a variety of domains.

The test oracle is implemented by an oracle server and one or more oracle client components. The oracle server collects the verdict predictions from the oracle clients, which are used by other components. Figure 4.3 depicts the relations between the component types that are specified in LibIot. The oracle server TTCN-3 component type is extended by the test coordinator component type. The test coordinator component is dedicated to coordinate and synchronize the behavior of all other test components, which work on tasks independently. The test coordinator also controls the overall execution and manages the testing phases, as well as the final conformance verdicts and the interoperability verdicts.

[2]The TTCN-3 specification of LibIot can be downloaded from `http://t3tools.informatik.uni-goettingen.de/trac/browser/trunk/ETSI-Testsuites/ETSI_auto_IOT/ttcn/LibIot`

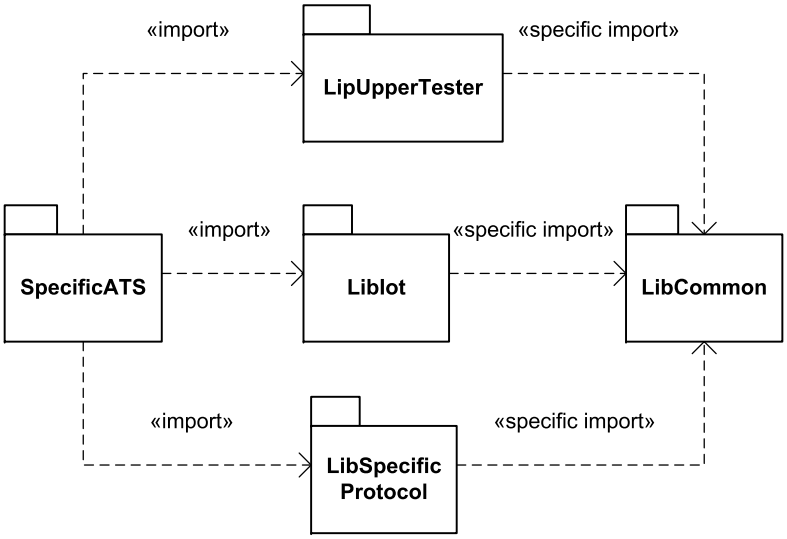


Figure 4.2: Library dependencies

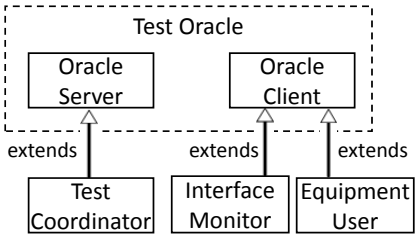


Figure 4.3: Component type relations in LibIot

The oracle clients can be extended by the interface monitor or the equipment user to send local verdicts to the oracle server. The equipment user acts as the end user of the service and can be used for configuring or triggering an EUT, the interconnecting network, or an application support node. The interface monitor component type is dedicated to monitor a logical interface either between two EUTs or between an EUT and an application support node. It checks messages and procedures on communication links and also calculates conformance verdicts.

Only a basic structure for test automation of the different types of interoperability testing activities are defined in LibIot. Foremost, this is the definition of the aforementioned means of interoperability testing. They are the basis for the test automation and are used and specified in the ATS with consideration of the identified limitations for the SUT. In addition, LibIot defines a basic structure for the configuration of the EUTs and the interconnecting network. LibIot specifies test parameters for EUT interface information of each system

participating in an interoperability event, e.g., supported interfaces, domain names, and user identities. Furthermore, from a previously defined list, the systems participating in the interoperability test are selected. This means that the test case specifications are independent of specific EUT configuration information. It is also possible to disable observations of specific interfaces in order to adjust tests in case an interface is not accessible. This results in disabling the execution of the associated monitor component. The test parameters support the setting of the capture mode, i.e., to switch between live and offline message checking during the interoperability test execution.

Time limit parameters are set in general and are applied throughout the whole test suite. They can be used to set the maximum time for a monitor test component to wait for an incoming message.

4.4 Development Process for Automated Interoperability Tests

We present the *Development of an Automated Interoperability Test System* (DAITS) to show how automated interoperability tests can be developed in a systematic and formalized manner based on LibIot and the test design guidelines. The DAITS process contains methods for the analysis of critical aspects that require considerations before and during the development of a complete interoperability test system. The test system developed by the application of the presented DAITS process can be implemented in any language and specified for any distributed system.

Our DAITS process comprises four main phases: prerequisites, interoperability test design, test case specification, and validation. Figure 4.4 shows the complete DAITS process. Throughout all phases, the generic environment for automated interoperability tests described in Section 4.1 and the guidelines for specifying for automated interoperability tests described in Section 4.2 are applied and form the basis of DAITS.

4.4.1 Roles

Roles in software testing are: test manager, test designer, test automator, test administrator, and tester. Spillner et al. define and describe each of these roles in [132]. These roles also apply for the development of a complete automated interoperability test suite, which is developed by multiple stakeholders taking different roles. However, the development of an automated interoperability test suite is mainly done by the test designer and the test automator. For finer granularity and clearer matching of the roles on tasks of the DAITS process, we split the role of the test automator into the following roles.

- **Test library implementer** defines generic test libraries.
- **Test system architect** specifies the test system.
- **Test case implementer** develops test cases.

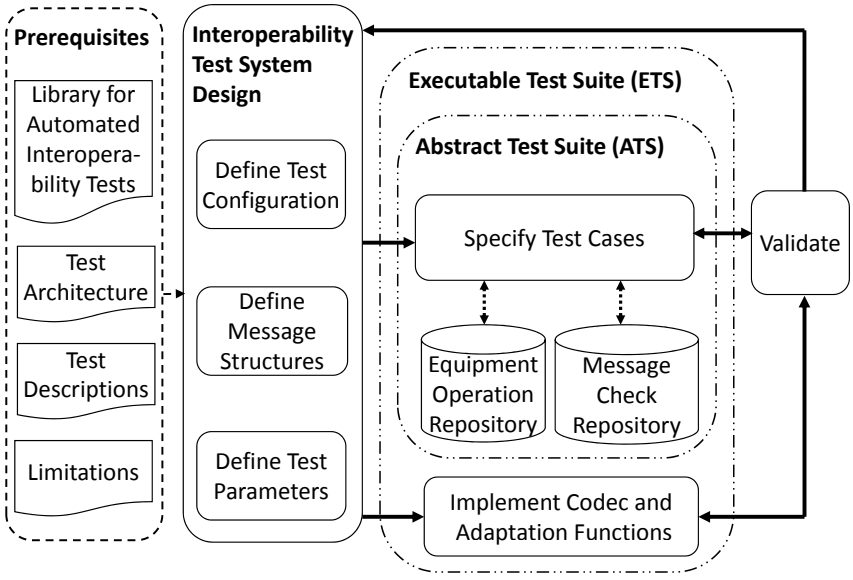


Figure 4.4: Process for the Development of an Automated Interoperability Test System

- **System adapter and codec implementer** implements the interface between the test suite and the SUT.

For test automation purposes, all four roles of the test automator use a programming or specification language. In the remainder of this section, we map the roles to the different tasks of the DAITS process.

4.4.2 Prerequisites

The prerequisites for the DAITS process are the test architecture, test descriptions, the library for automated interoperability tests, and the identified limitations. The library for automated interoperability tests is described in Section 4.3 specifically for TTCN-3 and provides functionalities regarding interoperability testing for test system specifications. It is the basis for the definition of test component types, the test interfaces, and the test parameters. It is developed by the test library implementer. If a standardized common protocol, e.g., SIP, is involved in the test, a test library for this protocol can be implemented by the test library implementer. This library is then input for the test development and can be reused by other test suites.

The two inputs test architecture and the test descriptions are described in the previous chapter in Section 3.4. As a further input to the DAITS process, the SUT is assessed for limitations to determine the possible degree of test automation. This can include considering security issues, e.g., encrypted messages, accessibility of entities of other providers,

and the configuration and operation of involved equipment. This assessment is especially interesting when interfaces involved in the interoperability tests are not standardized. The test designer develops the test architecture, specifies the test descriptions, and identifies the limitations.

4.4.3 Interoperability Test Design

The test system architect conducts the interoperability test design, which leads to the definition of all structural information required for the definition of the test entities. This also includes the specification of their behavior as well as the communication between them and with the SUT. This means that the test system architect defines the test configurations, the message structures, and the test parameters.

For the definition of a test configuration, the test system architect creates a concrete instance of a test architecture on the basis of test entities, communication links and their connection. For this, the test system architect identifies the EUTs, interfaces for monitoring, and test entities in the test architecture. The test configuration defines the entity instances, which realize the functionality of one or more test entity types. This increases reusability of test behavior across a test suite and makes test case specification less dependent on the availability of interfaces.

Afterwards, the test system architect defines the message structure for all interfaces based on their respective protocol specification. In case the protocol specification standardizes the abstract syntax of messages, e.g., by using *Abstract Syntax Notation One* (ASN.1), this step is not needed. In case that proprietary interfaces need to be accessed, an abstract access handler should be implemented.

The definition of test parameters is required to ensure the adaptability of the test specification for different execution environments or SUTs without modifying executable code. In general, test parameters should be specified for different generic functionalities, e.g.:

- The **EUT interface information** parameter is used to set configuration data, e.g., identifiers and addresses of involved equipments, to allow an easy adaption of test cases to equipment provided by a different vendor.
- **Time limit parameters** are used to set timeout values, e.g., for a response time or for the time length of an activity in a test case.
- **Message parameters** allow adapting test cases to specific needs of a testing environment or of EUTs, e.g., the setting of specific user identities.
- **Interface availability parameters** allow enabling and disabling observations of specific interfaces that are not accessible in an automated manner.
- The **capture mode parameter** switches the message checks between live and offline mode.

4.4.4 Test Case Specification

The next phase of the DAITS process comprises the specification of test cases, which is done by the test case implementer. A test case is a sequence of test steps that are executed on different test entities. A test step is typically a group of elementary actions and checks that together fulfill a specific goal. Similar to subroutines and functions in conventional programming languages, it is useful that the test case implementer collects test steps in a test step repository and reuses them across a test suite. The test step definitions should be parameterized to improve their potential for reuse.

For the definition and specification of a test step, the following guideline should be followed:

1. Select and isolate the sequence of actions and message checks that fulfill the same abstract purpose.
2. Specify the content of the messages that are sent and expected as defined in the standard based on specific criteria.
3. Introduce guards to avoid deadlocks and define timeouts in case of no activity.
4. Handle exceptions.
5. Assign an appropriate test verdict type, i.e., interoperability or conformance verdict type, to the test steps.

The selection of the actions and checks for their integration into a test step is mainly determined from the testing experience of the test case implementer. Feasible sources for the identification of test steps are test descriptions, which already specify abstract sequences of actions and checks. Developing test steps from test descriptions and conformance test purposes has the following advantages. The terminology and identifier naming is aligned between a test case specification and a test description. With a test description, it is easy to identify the relations between a test specification and an SUT specification. This allows easier maintenance of the test steps in case the test description changes. In addition, the readability of interoperability test execution traces is improved.

Reusable interoperability test steps are categorized and stored in two repositories. The equipment operation repository includes all actions required to configure and operate a specific EUT as well as application support nodes and the interconnecting network. The message check repository contains the message checks applied to the monitored interfaces. The repositories are built on-the-fly during the specification of test cases.

An interoperability test case is specified from a test description. The specification of each test case is structured into the following five successive parts.

1. The **test configuration setup** instantiates the interoperability test entities and their communication links.

2. The **preamble** consists of a preliminary set of steps to put the SUT and the network elements that are involved in a test into the state that is described in the pre-test conditions. This includes configuration actions, e.g., the registration of a user.

3. The **test body** is the sequence of test steps and contains actions and checks for each test entity instance. These are needed to achieve the interoperability and the conformance test purposes.

4. The **postamble** includes a set of test steps to return the SUT and the involved network elements into their initial configuration.

5. The **tear down** establishes the initial state of the test environment. The test system resources, e.g., interfaces, users and memory, should be released by invoking a general purpose function to release resources. The creation of the initial states prevents abnormal test terminations on the following test case executions.

In general, tests should be specified in a way that handles abnormal test termination properly. To this aim, general purpose functions to manage exceptional events are defined. These functions are called in case a timeout occurs, i.e., the test system receives no input for a specified amount of time.

The specification of the test coordinator behavior includes the creation and management of test entities required for the tests. In addition, the management of final test verdicts based on the execution of other test entities is specified.

The test case implementer specifies the test cases as part of the ATS. The ATS consists of test suite specific definitions, e.g., test configuration management, test case statements, and test purpose checking functions. Reusable definitions are isolated from ATS specific definitions in libraries. The ATS and library specific modules are distinguished by their prefix as “Ats” or “Lib” respectively. An ATS for interoperability tests is structured in a systematic way and consists of the following modules:

1. The **TypesAndValues** module defines ATS specific types and values, without test components.

2. The **Templates** module defines ATS specific templates.

3. The **TestSystem** module defines component types used to create MTC and PTCs, and the test system component type.

4. The **TestConfiguration** module defines functions that establish the configuration of the test system.

5. The **Behavior** module defines ATS specific functions for message checks related to test purposes as well as conformance criteria associated with test descriptions.

6. The **Functions** module defines ATS specific functions that are not directly related to a specific test purpose.

7. The **TestCases** module defines test cases that can be split into multiple modules, e.g., for grouping test cases according to functionalities.

8. The **TestControl** module defines the control part for test case selection.

The test case specification completes the ATS specification. At this point, tests are written in a formal manner, but are still not executable. Therefore, the codec and adaptation functions are implemented by the system adapter and codec implementer to complete the executable test system. Within these functions, messages that are exchanged with the SUT are converted from the abstract representation into the transfer protocol syntax and vice versa (encoding and decoding respectively). After encoding and decoding, messages are transferred via test interfaces to and respectively from the relevant EUT in the SUT. The adaptation functions also provide means for the test entities to communicate with other test equipment including application support nodes and the interconnecting network. This can be used to apply protocol analyzers or to jam the equipment, e.g., in stress testing. Furthermore, related to the message checks, the message traffic of a monitored interface can be filtered according to specific rules to a logical interface used by the traffic capture entity. The adaptation layer also implements the mapping of abstract operations of an abstracted EUT interface to a concrete, proprietary interface. This is used for the configuration of EUTs. For each vendor, a mapping to their specific interfaces is implemented. Only on the abstract level of the test case specification, the EUTs can be used independently from the vendor. The adaptation layer makes it possible to actually execute the test cases and control their execution. After compiling and linking the ATS with these functions, an executable code, the *Executable Test Suite* (ETS) is obtained.

4.4.5 Validation

Validation assures that the test system faithfully reproduces the behavior specified for each test description at its interface. First of all, the test system specification is reviewed against the test descriptions by independent test experts. Second, the test system is connected and executed against a real SUT by the tester. If no SUT is available, the test system can be tested in a back to back configuration or with an SUT emulator. Afterwards, the results of the tests are reviewed and validated. According to the results of the validation, improvements of the test specification and corrections of errors in the system adapter are applied after the validation step is completed.

4.5 Related work

The foundations of this chapter are described in [36], where concepts for interoperability testing, a description of a test architecture as well as a process for developing and executing interoperability test specifications are presented. However, the approaches described in [36] focus on certification of systems. For this, a so called golden or reference implementation, against which all the other implementations are tested, is required to run the tests. Our generic environment for automated interoperability tests and the DAITS process is applicable in interoperability events. Therefore, our approach does not need a reference implementation, since all implementations are tested against each other. Our generic environment, the test design guidelines, and the DAITS process complement and extend the concepts described in [36] by covering aspects related to the testing of distributed systems and the automation of interoperability testing for the purpose of validating standards and of validating systems against standards. The concepts presented in this chapter are discussed in detail in three ETSI documents: ETSI EG 202 810 [35], ETSI TR 102 788 [37], and an ETSI white paper [8].

In the following, we compare our contribution to work that has been done in the field of automated interoperability testing. Vega et al. present a design of a test framework for automated interoperability testing of healthcare information systems [145]. Their approach is not generic since it is only applicable in the healthcare area. Furthermore, it is not clear to which degree they apply automation. The difference of the presented approach is that our methodology is generic and, therefore, independent of the SUT. In addition, we provide a definition of test automation and its degree.

Brammer investigates the automation of terminal interoperability tests [98]. He suggests tasks, which can be automated in interoperability testing of mobile terminals from different vendors. We discuss automation related to the automated test life cycle methodology presented in [22]. However, our process and methodology focus on interoperability testing and, therefore, take important interoperability aspects and concepts into consideration. Brammer only presents a domain specific solution. Our interoperability methodology can be used independent of the SUT.

Dibuz and Kremer present a framework for automated interoperability tests and its application to *Robust Header Compression* (ROHC) [20]. This framework is defined in TTCN-3. Our methodology and the test development process can be instantiated in the programming language of choice. In addition, it is also unclear to which degree Dibuz and Kremer applied automation of the tests.

Many papers consider automated generation of interoperability test cases, e.g., [10, 19, 128]. In contrast, we present an approach for automated interoperability test execution and automated interoperability assessment, which does not consider interoperability test generation.

5 Interoperability of IP Multimedia Subsystems

For the initial case study, we choose to analyze the interoperability of homogeneous systems for which stable standards already exist, because the interoperability analysis and interoperability assessment is easier than for heterogeneous systems. A homogeneous system is specified in the IMS standards [1]. The IMS standards comprise a set of specifications that enable implementations of IP-based networks to use services simultaneously with equipment that is attached to fixed and mobile networks. IMS standards already specify interfaces for interoperation with other IMS networks.

In this case study, we instantiate the IAI process that we describe in Chapter 3 for IMS networks. As part of the IAI process, we assess interoperability of IMS networks in an automated manner as we describe in Chapter 4. Since IMS networks have well defined interfaces, the development of an automated interoperability test suite is not as limited as in heterogeneous environments. Before the utilization of our methodology, IMS interoperability testing was largely performed in a costly and manual manner. Therefore, we advance the state-of-the-art of IMS interoperability testing.

This chapter is structured as follows. In Section 5.1, we discuss Phase I of the IAI process, which includes the identification of the prerequisites for interoperability of IMS networks. As the standards are already engineered for interoperability, we can skip Phase II. We describe the development of an interoperability test system for IMS using TTCN-3[3] as part of Phase III of the IAI process in Section 5.2. The test system assesses the interoperability of IMS networks in an automated manner through the application of the generic methodology for automated interoperability assessment including the instantiation of the DAITS process. Based on the results of Phase III, we briefly describe how interoperability of IMS networks can be improved as part of Phase II of the IAI process in Section 5.3. We conclude this chapter with related work in Section 5.4. This chapter is partially adapted from our previous publication [122].

[3]The IMS interoperability test system can be downloaded from `http://t3tools.informatik.uni-goettingen.de/trac/browser/trunk/ETSI-Testsuites/ETSI_auto_IOT`

5.1 Phase I: Interoperability Prerequisites

The interoperability analyst analyzes the IMS standards [1] from which the analyst extracts common and complementary functionalities of IMS networks. Common capabilities of IMS networks are, e.g., the registration in the IMS network, voice calls, message exchange, and the usage of application servers.

An IMS core network is required to interoperate with other IMS core networks over IP. The *Network-to-Network Interface*s (NNIs) of IMS core networks are specified in the IMS standards. From the analysis of the IMS standards and of the IMS architecture, which is schematically depicted in Figure 2.11, the interoperability analyst determines that the standardized *Gm*, *ISC*, and *Mw* NNIs, which are most important for IMS interoperability, need to be assessed in Phase III. The input for Phase III are the IMS standards as well as the report about interoperable interfaces of IMS.

5.2 Phase III: Automated IMS Interoperability Testing

The interoperability of IMS network implementations and the quality of the IMS standards related to interoperability need to be assessed by means of interoperability testing with message checks to identify interoperability gaps. Therefore, we developed an interoperability test suite, which is used to assess the interoperability as well as the standards of IMS implementations. We applied the methodology for automated interoperability assessment and instantiated the DAITS process. The test suite comprise scenarios, where the interoperability of two IMS network implementations is evaluated at their standardized NNIs. This means that interoperability and conformance are assessed in different configurations, e.g., IMS interworking, IMS roaming, and topology hiding.

5.2.1 DAITS Process Prerequisites

For the development of an automated interoperability test system for IMS, four prerequisites are required by the DAITS process as depicted in Figure 4.4. The test system architect utilizes the LibIot library as the first prerequisite. The TTCN-3 ATS for IMS interoperability, which is specified in the interoperability test design phase, imports the LibIot as Figure 4.2 shows.

As the second prerequisite, the test system architect receives the test architecture from the test designer. Figure 5.1 depicts one example IMS test architecture, which is based on the IMS network architecture. The IMS networks A and B are connected via the *Mw* interface. Additionally, IMS network A is connected to *User Equipment A* (UE_A) and IMS network B is connected to *User Equipment B* (UE_B). The test designer identifies IMS network A as EUT_A and IMS network B as EUT_B. The tester maps IMS networks that are implemented by different vendors to these abstract identifiers.

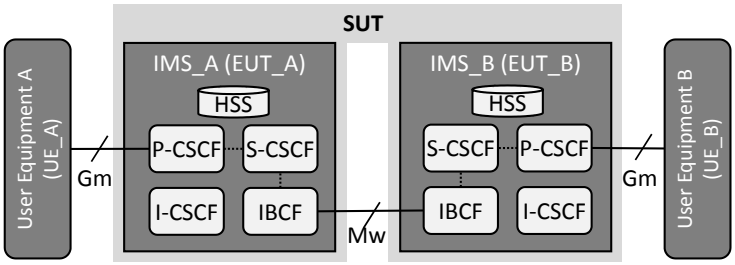


Figure 5.1: Example test architecture - interworking IMS networks

As the third prerequisite, the test system architect receives the test descriptions from the test designer. The test designer specifies the test descriptions based on the IMS standards, IMS architecture, and the report of Phase I. Figure 5.2 shows an example of a test description for the interoperability test "IMS Call Cancelation by Calling User". The identifier of this test description is TD_IMS_CALL_0014, which can be mapped to a concrete test case. The interoperability test purpose is to check if the IMS network handles cancelations from a calling user before the call is established correctly. The test architecture that is used in this test is presented in Figure 5.1. The referenced standard, which will be assessed, is specified in the document TS 124 229 [44]. Before executing the test, UE_A and UE_B are registered in EUT_A and EUT_B respectively.

The test sequence includes the verification of end-to-end functionality. Figure 5.3 depicts the test sequence in a sequence diagram. UE_A is used by user A to call UE_B of user B. As a result, EUT_A contacts EUT_B to initiate the call. After UE_B started ringing and before user B answers, user A decides to cancel the call.

Messages that are expected during the execution of the test sequence are specified in the conformance criteria. In this case, the criteria refer to one conformance test purpose, which checks the attributes of the CANCEL message received by EUT_B and sent by UE_A to UE_B. We use this test description as a running example for the application of the DAITS process throughout this section. For other test descriptions for IMS NNI testing, the reader is referred to [41].

For the fourth prerequisite, which are the limitations, we identified three limiting factors for the IMS network architecture.

1. **Authentication and security**: It is not possible to check encrypted messages on the *Gm* interface. To enable monitoring, IPsec authentication has been disabled to conduct this case study.

2. **Interface accessibility**: Although the *ISC* interface is standardized, it may not be externally accessible when the IMS network and the application server are implemented by the same vendor.

Test Description		
Identifier	TD_IMS_CALL_0014	
Summary	IMS network handles cancels of a call by a calling user correctly before the call is established	
Test Architecture	Interworking IMS Networks (CF_INT_CALL)	
Specification Reference	TS 124 229 [44], clause 5.4.3.2 ¶119 (item 1 in 8th numbered list)	
Pre-test Conditions	• Home Subscriber Server (HSS) of IMS_A and of IMS_B is configured according to table 1 of ETSI TS 186 011-2 [41] • UE_A is registered in IMS_A using any user identity • UE_B is registered in IMS_B using any user identity	
Test Sequence	**Step**	
	1	User A calls User B
	2	Verify that User B is informed of incoming call of User A
	3	Verify that User A is informed that UE_B is ringing
	4	User A cancels the call, before User B answers or before network timeout
	5	Verify that User B is informed that call has been cancelled
	6	Verify that User A is informed that call is terminated
Conformance Criteria	**Check**	
	1	TP_IMS_5107_03 in CFW step 24(CANCEL): ensure that { when { UE_A sends CANCEL to UE_B } then { IMS_B receives the CANCEL containing no Route_header indicating the S-CSCF_SIP_URI of IMS_A }}

Figure 5.2: IMS test description: “IMS Call Cancelation by Calling User” [41]

3. **Equipment accessibility**: Interfaces to configure and operate UEs, e.g., to initiate a call, as well as interfaces to configure the IMS network elements, e.g., to enter or block a user in the *Home Subscriber Server* (HSS), are not standardized and, therefore, hard to automate.

The limited access from the test suite to specific IMS network entities and interfaces is marked in Figure 5.4.

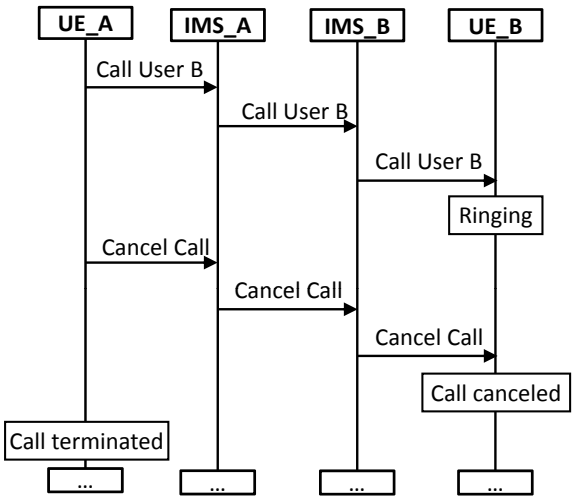


Figure 5.3: Test sequence for IMS call cancelation by calling user

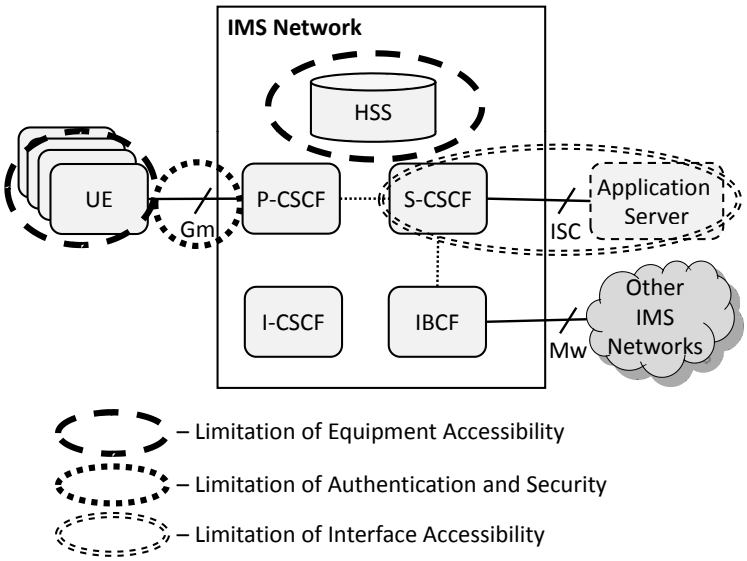


Figure 5.4: Limitations for automation of IMS network interoperability tests

5.2.2 Interoperability Test Design

For the definition of an IMS test configuration, the required test entities for the test architecture, which are shown in Figure 5.1, are identified by the test system architect. The following test component types are imported from LibIot and instantiated:

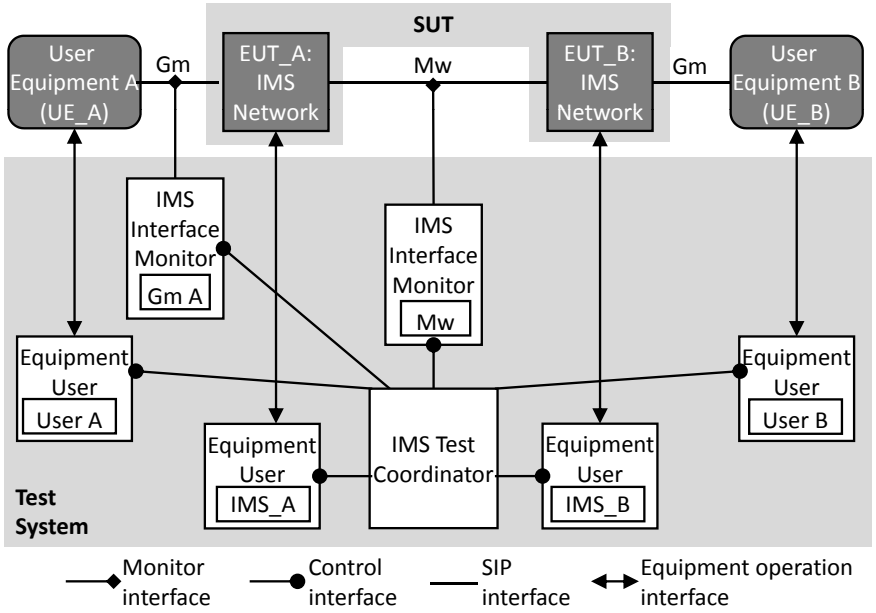


Figure 5.5: Test entities, which constitute an IMS test configuration

- four instances of the `EquipmentUser`[4] to trigger and configure two UEs and two IMS networks, i.e., the EUTs,
- two instances of the `InterfaceMonitor` to monitor the *Gm* and *Mw* interfaces, and
- one instance of the `TestCoordinator` to manage the different instantiations of the test entities.

For IMS, the test system architect extends the `InterfaceMonitor` component type to the `IMSInterfaceMonitor` component type as well as the `TestCoordinator` to the `IMSTestCoordinator` to add IMS specific ports. Figure 5.5 depicts the relations between the test entities and the SUT. Each EUT participating in a test has a dedicated test component of type `EquipmentUser`. In this test configuration, the UEs are part of the system adapter and not assessed for interoperability. The `IMSTestCoordinator` acts as the MTC. Each monitored IMS interface is paired with a dedicated PTC of component type `IMSInterfaceMonitor`, which receives all relevant message information from the system adapter via the abstract TSI. These PTCs check the correctness of the message information according to the conformance criteria. The conformance criteria of the test description do not require message checks on the *ISC* interface and on the *Gm* interface between EUT_B and UE_B. Therefore, the test configuration that Figure 5.5 depicts does not consider the ISC interface.

[4] In the remainder of this work, we use the typewriter font to highlight TTCN-3 identifiers.

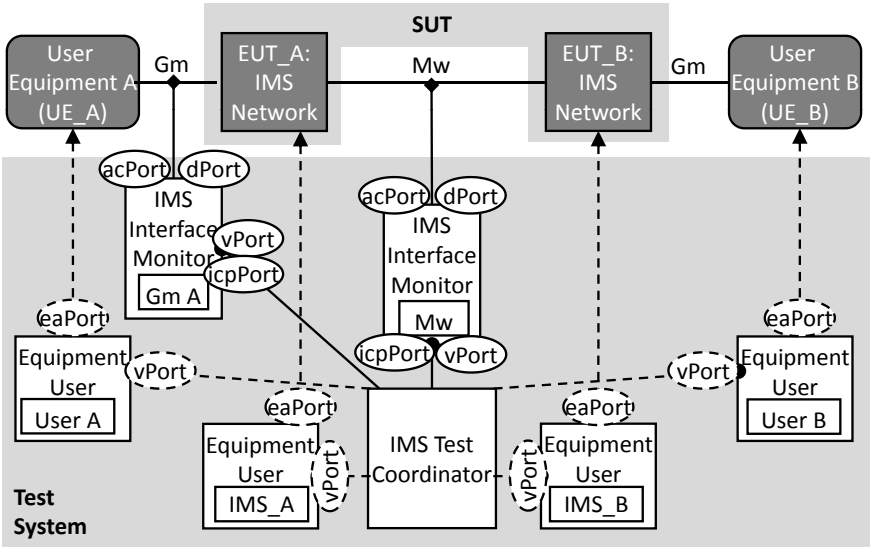


Figure 5.6: Fully developed test configuration - interworking IMS networks

To allow message exchange between the test entities and the SUT, the ports of the test entities are mapped and connected. TTCN-3 components can communicate with the adapter via the abstract TSI, which is the system component. Within the ATS, test entities are mapped to the TSI component via three ports: the *adapter configuration port* (`acPort`), the *data port* (`dPort`), and the *equipment access port* (`eaPort`). The `acPort` is the interface for TTCN-3 components to communicate with the adapter to perform general configurations, to set filters for traffic monitoring in the lower test adapter, and to start and stop traffic capture. The test adapter uses the `dPort` to send messages captured and filtered from EUT traffic to `IMSInterfaceMonitor` test components. `EquipmentUser` test components use the `eaPort` to request operations of an EUT via the adapter, e.g., for triggering a registration of an UE. In addition, the test components are connected to each other via two ports: the `icpPort` and the `vPort`. The `icpPort` forwards test system internal messages to the `IMSTestCoordinator` if required. It is of the type `IMSCoordinationPort` to allow receiving and sending of SIP messages. The `vPort` communicates local verdicts to the oracle server. Figure 5.6 schematically shows the abstract port mappings and connections of the test components comprising the fully developed test configuration. The test components with dashed lines are only started in live mode. In the offline mode, the PTCs of component type `EquipmentUser` are not started, because the EUTs are driven manually.

Within the interoperability test design, the test system architect defines the message structures. Listing 5.1 shows the SIP request type definition as an example for defining message types. These definitions are based on generic request and response types. The `type record Request` is a generic type of a SIP request message. Its definition includes the following four fields.

```
type record Request { // SIP Request from IETF RFC 3261
  RequestLine requestLine,
  MessageHeader msgHeader,
  MessageBody messageBody optional,
  Payload payload optional
}
type record RequestLine {
  Method method,
  SipUrl requestUri,
  charstring sipVersion
}
type enumerated Method {
   ACK_E,
   BYE_E,
   CANCEL_E,
   INVITE_E
}
```

Listing 5.1: TTCN-3 types for a SIP Cancel

1. `requestLine` contains a method name, e.g., `CANCEL`; a Request-URI; and the protocol version.

2. `msgHeader` includes all possible header fields that are allowed to be present according to the *Request for Comments* (RFC) 3261.

3. `messageBody` includes messages, e.g., *Session Description Protocol* (SDP) messages, which depend on the request method.

4. `payload` contains the whole message as it has been received in its text format.

After defining all required types, the test system architect defines basic templates in the ATS based on these types. In our running example, the method field of the `RequestLine` definition is set to the value `CANCEL_E`.

The definition of test parameters concludes the interoperability test design. In case of TTCN-3, test parameters are module parameters. Listing 5.2 shows the module parameter that captures the information for all supported interfaces of a system participating in an interoperability event. The parameter defines the product name and specifies interfaces with their IP addresses and ports. Within LibIot, the test library implementer defines module parameters for the generic functionalities as described in Section 4.4.3.

To summarize, in the interoperability test design phase, the test system architect defines test configurations, develops a basic and generic message structure for the SUT, and defines all test parameters to configure the tests. All definitions of the interoperability test design phase are reused during the test case specification.

```
1  modulepar ProductList PX_PRODUCTS := {
2    { productName := "Super IMS",
3      monitorInterfaces := {
4        { interfaceName := "Mw",
5          interfaceInfo  := {
6            IpInterfaceInfo  := {
7            { domainName := "pcscf.ims.etsi",
8              IpAddress := "192.86.1.97",
9              portNumbers := {5060}
10           },
11           { domainName := "icscf.ims.etsi",
12             IpAddress := "192.86.1.98",
13             portNumbers := {5060}
14 }}}}}}}
```

Listing 5.2: Example TTCN-3 module parameter definition for EUT interface information

```
1  testcase TC_IMS_CALL_0014() runs on IMSTestCoordinator
2  system IOTSystemInterface {
3    // Part 1. Test configuration setup
4    // Part 2. Preamble
5    // Part 3. Test body
6    // Part 4. Postamble
7    // Part 5. Tear down
8  }
```

Listing 5.3: Interoperability test case specification structure

5.2.3 Test Case Specification

We exemplify the test case development with our running example. The test description, which is shown in Figure 5.2, is transformed into a test case, which implements the five parts as described in Section 4.4.4. Listing 5.3 schematically shows the test case IMS Call 0014 of our running example. The `IMSTestCoordinator` is chosen to be the MTC and the `IotSystemInterface` reflects the interface to upper and lower test adapters.

The test begins with the establishment of the test configuration "Interworking IMS networks", which is depicted in Figure 5.6. To check the conformance criteria as described in the test description, the `IMSInterfaceMonitor` test components for the *Gm A* and the *Mw* interfaces are configured and started. Since the test should be executed in live mode, the lower test adapter is configured for using live mode. For this, a configuration message is sent from the `IMSTestCoordinator` to the system component. After all lower test adapter configurations are completed, traffic capture processing is initiated by the `IMSTestCoordinator`.

The pre-test conditions of the test description are executed within the preamble. We assume that the IMS networks were manually configured as required. UE_A and UE_B are then registered in their respective home network. Figure 5.7 shows the message exchange

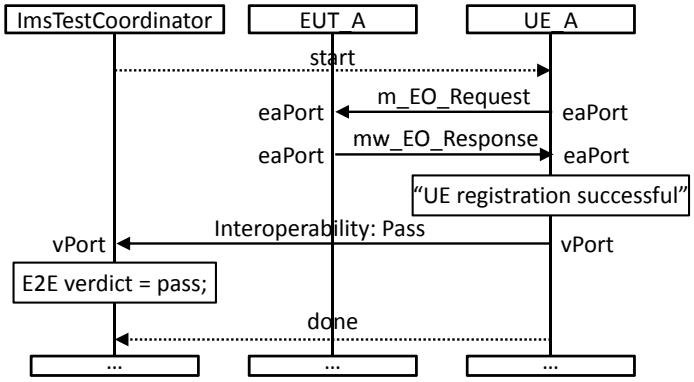


Figure 5.7: Successful Registration of UE_A in EUT_A

```
1 function f_userRegistration(
2   in charstring p_publicId, in charstring p_privateId,
3   in charstring p_pw)
4 runs on EquipmentUser {
5   // Liblot function
6   f_sendEquipmentCmd(valueof(
7     m_EO_Request(c_UE_REGISTRATION,
8       {p_publicId, p_privateId, p_pw})));
9 }
```

Listing 5.4: User registration function

for performing the task of a registration of UE_A in EUT_A. The `vPort` is used to communicate the result of the task to the `IMSTestCoordinator` test component. Listing 5.4 depicts the function for the registration of a user. This function utilizes the generic equipment operation function `f_sendEquipmentCmd` that is implemented in the LibIot. The constant `c_UE_REGISTRATION` is used as an abstract equipment command for the test adapter to trigger the registration of the UE.

Listing 5.5 depicts the function calls that specify the test body. The `IMSTestCoordinator` triggers the behavior by starting the test components passed as a parameter within each function call. Only after a specific input is sent as an equipment operation to the SUT, e.g., initiate a call, it is possible to execute the message checks related to this input. In the following, the function `f_mtc_userInitiateCall` is discussed as an example for an equipment operation and the functions `f_mtc_check_TP_IMS_5107_03_gm` and `f_mtc_check_TP_IMS_5107_03_mw` as examples for the message checks. Other functions of the test body and of other tests are specified in an analogous way.

Listing 5.6 depicts the specification of the function `f_mtc_userInitiateCall`. The requested `EquipmentUser` test component is started to execute a function implemented in the LibIms UpperTester library reflecting the action to be taken. In this case, the action

```
1 f_mtc_userInitiateCall (v_ueA,v_userInfoB); // Step 1
2 f_mtc_userCheckRinging(v_ueB); // Step 2
3 f_mtc_userCheckPeerIsRinging(v_ueA); // Step 3
4 f_mtc_userTriggerCancelCall(v_ueA); // Step 4
5 f_mtc_check_TP_IMS_5107_03_gm(v_gmA,false); // Check1
6 f_mtc_check_TP_IMS_5107_03_mw(v_mw,false); // Check1
7 f_mtc_userCheckCallCancelled(v_ueB); // Step 5
8 f_mtc_userCheckCallEnded(v_ueA); // Step 6
```

Listing 5.5: Test body

```
1  function f_mtc_userInitiateCall(
2    EquipmentUser p_userCompRef, ImsUserInfo p_calledParty)
3  runs on IMSTestCoordinator return boolean {
4    var boolean v_success := false;
5    p_userCompRef.start(
6      f_userCallInitiation  (p_calledParty.publicID));
7    p_userCompRef.done;
8    if (f_getE2EVerdict() == pass) {v_success := true;}
9    return v_success;
10 }
```

Listing 5.6: Exemplified function specification for an equipment operation

implements the sending of a message to initiate a call from UE_A to UE_B. The function `f_userCallInitiation` is a wrapper function for the LibIot function `f_sendEquipmentCmd`, which is also called in Listing 5.4. Similar to the actions of the sequence diagram depicted in Figure 5.7, the `IMSTestCoordinator` starts the PTC of UE_A. However, in this case, the equipment operation with template `m_EO_Request` sent by user A to the system component contains instruction parameters that initiate a call instead of registering a user. In case the call is initialized successfully, the system component replies with a success, which is communicated via the `vPort` to the `IMSTestCoordinator`.

For checking the conformance criteria of the test description, message checks are specified. Each message check requires checks on two interfaces as listed in Listing 5.5 (lines 5 and 6). The `IMSTestCoordinator` function starts relevant `IMSInterfaceMonitor` test components for the interfaces involved in the message check - one for the *Gm* interface and one for the *Mw* interface. The function specification of the *Gm* interface check is depicted in Listing 5.7.

The `IMSInterfaceMonitor` test component executes a generic IMS checking function. All message checks are realized by calling the same generic function `f_imsIot_receive`. This function is customized via its parameters, which define checks that are performed according to the respective test purpose listed in the test descriptions. Figure 5.8 visualizes the meanings of the different parameters. The function sets the test component conformance verdict and sends this verdict to the `IMSTestCoordinator`. It determines a pass if the received message matches a pass-template and a fail if it matches a fail-template. If there

```
1  function f_mtc_check_TP_IMS_5107_03_gm(
2        ImsInterfaceMonitor p_monitorCompRef,
3        boolean p_checkMessage )
4  runs on IMSTestCoordinator {
5    p_monitorCompRef.start(
6      f_imsIot_receive( // LibIms function
7        {mw_SipRequest(mw_CANCEL_Request_Base(?))}, {},
8        {0, omit},"TP_IMS_5107_03", false, p_checkMessage )
9    );
10   p_monitorCompRef.done;
11 }
```

Listing 5.7: Function for a message check

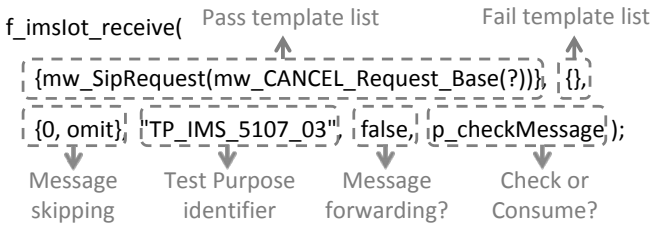


Figure 5.8: Meaning of the parameters of the generic receive function

is no match, it waits for the next message via the interface until a match arises or the guard timer expires.

Additional function parameters allow configurations to skip messages, forward matched messages to the `IMSTestCoordinator`, and check or consume messages. The function skips messages in case capture traces contain insignificant traffic or known messages that do not have to be checked, e.g., from the preamble. This traffic is ignored so that its assessment does not lead to a fail conformance verdict. Matched messages are forwarded to the `IMSTestCoordinator` in case that locally received content has to be compared or derived from messages sent or received in another local scope. The last parameter either removes or does not remove a message from the port queue. The latter is required in case another check needs to be performed.

The `mw_CANCEL_Request_Base`[5] is a base template for SIP message checks. Complex checks are specified in templates derived from the base templates and often named after the test purpose in the test description. For example, the template `mdw_TP_IMS_5107_03_mw` modifies `mw_CANCEL_Request_Base` and is shown in Listing 5.8.

The invocation of the function `f_mtc_check_TP_IMS_5107_03_mw` of the test body handles the part of the conformance test purpose relevant to the *Mw* interface. The function uses the template `mdw_TP_IMS_5107_03_mw`, which Listing 5.8 shows. This template spec-

[5]The prefix `mw` is not referring to the *Mw* interface of IMS but a naming convention indicating a template or message that contains wildcards. Further naming conventions and their explanations can be found in [37].

Conformance Criteria	Check	
	1	TP_IMS_5107_03 in CFW step 24(CANCEL): ensure that { when { UE_A sends CANCEL to UE_B } then { IMS_B receives the CANCEL containing no Route_header indicating the S-CSCF_SIP_URI of_IMS_A }}

Figure 5.9: Conformance criteria of the test description for the Mw message check

```
1  template CANCEL_Request mdw_TP_IMS_5107_03_mw (
2    template CallId p_callId, template SipUrl p_SCSCF_SIP_URI)
3  modifies mw_CANCEL_Request_Base := {
4    msgHeader := {
5      route := (
6        omit,
7        { fieldName := ROUTE_E,
8          routeBody := {
9            *, complement(mw_routeBody(p_SCSCF_SIP_URI)), *
10 }}}}}
```

Listing 5.8: Specification of the Mw message check

ifies the content of all mandatory fields as described in the conformance criteria of the test description of our running example as Figure 5.9 depicts. Listing 5.8 and Figure 5.9 shows the relation between the test description, which is only executable in a manual fashion, and the test specification, which can be executed in an automated manner.

After the definition of the test body, the post condition that is executed within the postamble is specified. In our running example, all involved IMS users are deregistered. These requests are sent to the system adapter similar to the ones that establish the test configuration.

Furthermore, the initial state of the test environment that is specified in the tear down part of the test case is established. This includes stopping traffic capture in the lower test adapter, disconnecting and unmapping test component ports, and removing any selection requirements for the next test to be executed.

In addition to the specification of the ATS, the system adapter and codec implementer specifies the codec and adaptation functions for the ETS. To this aim, the system adapter and codec implementer develops codec functions for SIP and extensions required by the IMS, lower test adapter functions for capturing IP traffic, and upper test adapter functions for converting equipment operation requests into instructions for equipment operators. However, the interoperability test adapter is independent of IMS and SIP.

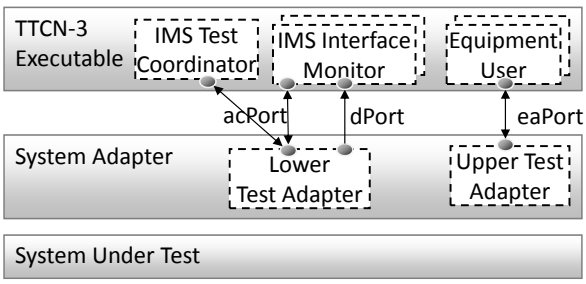


Figure 5.10: System adapter for the IMS interoperability test suite

Figure 5.10 shows the mapping of the test adapter to the test components. The `IMSTest-Coordinator` as well as each `IMSInterfaceMonitor` test component is mapped to the lower test adapter via the `acPort`. The `IMSInterfaceMonitor` is also mapped to the lower test adapter via the `dPort`. Each `EquipmentUser` test component is mapped via the `eaPort` to the upper test adapter. The ETS is built by linking and compiling the ATS, the codec, and the adapter functions. The ETS is then executed against the SUT.

5.2.4 Validation and Application of the Test System

Within the final phase of the DAITS process, the test system is validated. This was done through reviews by test experts of ETSI. In addition, the tester executes the ETS against a real SUT and evaluates the results according to the test specification. We applied the IMS interoperability test system in the third ETSI IMS Plugtests™ [28], where eight IMS network vendors attended. At this event, the tester executed tests related to basic IMS call functionality, messaging, IMS roaming, topology hiding, MMTEL supplementary services, and the presence service.

The IMS Plugtests™ series is organized in test sessions. Each test session is managed by an independent test session chair, who is appointed by ETSI. The tests of each test session are executed at match stations in the presence of two IMS network vendor teams and observers. Before executing a test, each IMS network vendor team connects an IMS UE to their IMS network. For the test execution, a member of each team operates the UE based on instructions of the test session chair. The IMS network traffic at the *Gm*, *Mw*, and *ISC* interfaces are monitored and saved by the test session chair according to the test description and the test session.

For each test session, 52 tests were available for execution. In the first part of each test session, the tester executed as many of these tests as possible having one IMS network vendor in the role of EUT_A and another IMS network vendor in the role of EUT_B. Afterwards, in the second part, the roles of EUT_A and EUT_B were reversed and the tester executed again as many of the 52 tests as possible. The first as well as the second part had a time limit of 90 minutes. Within these two parts, the focus was on the assessment of

the interoperability of the two involved systems and excluded conformance analysis. The interoperability was determined on mutual agreement of all involved parties and recorded by the test session chair.

In the third part, all test executions were stopped and a manual analysis of the conformance according to the test descriptions was performed. This part was limited to 60 minutes. For the analysis, as many tests as possible were selected and reviewed by the participating vendors and the test session chair using available trace capture tools. For each reviewed test, a conformance verdict was determined. All remaining tests were checked with an automated interoperability testing tool by ETSI representatives. The results of the conformance assessment were recorded with comments and issues during the test execution and conformance analysis in ETSI test session reports. For the main interoperability issues in IMS networks determined in this event and IMS interoperability test automation, the reader is referred to [38].

495 of 2,805 potential IMS NNI tests were executed out of which 317 were automatically analyzed for conformance. Analyses of the results of the test executions showed a high rate of passed interoperability verdicts, which proves a high level of interoperability of IMS networks. In total, 89% of the executed tests demonstrated interoperability. However, the conformance of the involved systems to the 3GPP base standard was lower. Only 55% of the tests showed conformance of the system to the standard. In addition, 13% of all potential tests were not executed caused by issues outside of the IMS networks, e.g., by a lack of the support of a feature by a participating IMS network.

5.3 Phase II: IMS Interoperability Improvement

The phase of improving the interoperability of IMS of the IAI process is based on the assessment of the standards. For this, the implementations were assessed in the interoperability event that we described above. The results of the interoperability event must be analyzed and comparisons between the same tests applied to different equipments need to be done to determine if the interoperability issues occurred are related to the implementation or to the standards. This is done by experts of IMS and standardization such as representatives of ETSI.

If interoperability issues are caused by the standards, the standards should be updated regarding the identified issues. If interoperability issues are caused by the implementation, the implementation needs to be updated according to the result of the assessment. The main sources of interoperability issues related to the implementations are programming errors caused by humans. If the issues were caused by the standards, the implementation needs to be updated as well after the revised standards are released.

After all required updates and improvements related to the standards and their implementations are done, a new interoperability event can be executed. The test system described above can be reused to assess the updated IMS implementations in Phase III. If new interoperability features are specified in a new release of the standards, the test system must be extended.

5.4 Related Work

This section extends the related work that we describe in Section 4.5 with focus on interoperability test systems and their development for IMS.

Several works have been done related to IMS conformance testing. These include the Ixia's IMS test solution, which is a tool that supports conformance tests but not interoperability tests for IMS [77]. Mulyana et al. deployed a prototyped testbed only for research purposes to identify issues of IMS implementations related to conformance [95]. However, they also investigated interoperability of IMS networks with non-IMS clients. This is also an important topic, but was not the focus of our work. Our focus was to test the interoperability between IMS implementations. Tang et al. developed a conformance test bed for IMS [138]. Their case study is based on the open source IMS implementation Open IMS Core [61]. They do not consider interoperability testing. Bormann et al. present a conformance test framework for complex services that they applied only for the presence service of IMS [12]. They only apply their framework to a fraction of IMS functionalities and do not consider interoperability testing.

Maarabani et al. established a testbed for interoperability tests of the IMS presence service [90]. Their experiments only cover the basic functionalities of the IMS presence service within a single domain. They followed the test specification for the IMS presence service described by OMA [105]. Our tests are based on the IMS test descriptions provided by ETSI [41]. In addition, it is not clear if their test drivers are executed manually or automatically. In our test system, the UEs can be stimulated and configured in an automated manner by instantiations of the equipment user entity. Ernits et al. present an approach for model-based testing for IMS [25]. They focus on the generation of tests from requirements, but only cover a fraction of IMS. They provide a feasibility study. In contrast, we presented a full test system for IMS interoperability that has successfully been applied in several interoperability events in an automated manner. In contrast to all listed related work with focus on IMS interoperability testing, we apply interoperability tests with message checks that are executed in an automated manner. This allows to check the conformance of messages that are sent between the EUTs during interoperation. In addition, the discussed related work does not consider automation. Our tests are executable in an automated manner.

6 Interoperability of Grid and IaaS Cloud Systems

In this chapter, we analyze the interoperability between grid systems and IaaS clouds with our IAI process. Both types of systems offer access to distributed and pooled computing resources and services. In comparison to our study of the interoperability of IMS in Chapter 5, grid systems and IaaS clouds are heterogeneous, which exhibits significantly greater challenges for both the design of interoperability solutions and the assessment of interoperability.

This chapter is structured as follows. In Section 6.1, we consider the prerequisites of interoperability between grid systems and IaaS clouds as part of Phase I of the IAI process. We compare both models based on their architectures to identify common and complementary functionalities. In Section 6.2, we apply Phase II of the IAI process and describe solutions for interoperability. This includes the description of several grid-cloud interoperability gateway implementations. We show a unique solution for the integration of grids within IaaS clouds, with which we advance the state-of-the-art of grid-cloud interoperability. In Section 6.3, we assess one of the presented interoperability solutions based on the GCM standards as part of Phase III of the IAI process. To the best of our knowledge, the interoperability of several grids and clouds has not been assessed before. Based on the results, we identify characteristics of both system that can be used as a basis for grid-cloud standardization. We conclude this chapter with related work in Section 6.4. This chapter is partially adapted from our publications [116, 118, 119, 120].

6.1 Phase I: Comparison of Grid Systems and IaaS Clouds

In Phase I of the IAI process, we analyze the interoperability of grid systems and IaaS clouds. We identify the prerequisites of interoperability as well as describe the survey about interoperability solutions. This task is performed manually by the interoperability analyst.

6.1.1 Common and Complementary Functionalities

Based on the direct comparison of the grid and cloud models, which is depicted in Figure 6.1, we identify common as well as complementary functionalities. Both systems are based on physical hardware. Within grid systems, the local resources are directly deployed on physical hardware. In contrast, IaaS clouds offer a resource management interface to

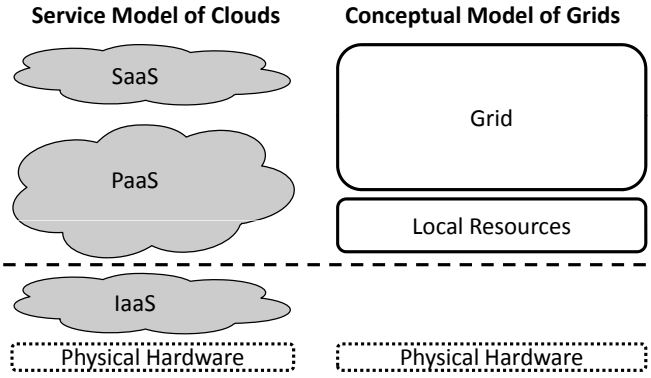


Figure 6.1: Comparing the layers of cloud with grid

install operating systems and applications dynamically within virtual machines and virtual infrastructures that are deployed on the underlying hardware. The installation of operating systems and applications is already done in our grid model, since the local resources on the bottom layer deploy a pre-configured software stack directly on the physical hardware. Therefore, a layer with similar functionalities as provided by IaaS clouds does not exist in our grid model. All grid functionalities including local resource functionalities are on the same level of abstraction as PaaS and SaaS clouds. However, the deployment of grid resources on IaaS clouds offers a valuable opportunity for interoperation.

The needs for grid-cloud interoperability are manifold. A combined usage of grid and IaaS clouds fosters an efficient resource use and resources on demand. IaaS clouds can be used for data replication and to decrease costs by choosing the best suited solution. In addition, a grid-cloud interoperability solution would allow the preservation of previous investment in grid application and system development. For example, it would be possible to migrate a well engineered grid application and grid environment into an IaaS cloud without any changes.

6.1.2 Survey Interoperability Solutions

To our best knowledge, standardization institutes do not consider the development of a grid-cloud interoperability standard. This is mainly caused by the heterogeneity of both systems. Additionally, there is only low commercial interest in grid systems that are mainly deployed and applied for academic research. However, some researchers worked on the integration of grid and cloud systems. In contrast to our work, these approaches usually do not distinguish between the different cloud layers in relation to the grid model. We describe the related work in Section 6.4.

In the grid and cloud domains, interoperability approaches exist mainly for interoperability between systems of the same domain. These approaches lay the foundation for

	Standard	Unicore 6	GT 4	GLite	GOS v3.2	Fura
Security	X.509	✓	✓	✓	✓	✓
	Security Assertion Markup Language (SAML)	✓	✓	✓	✓	
	extensible Access Control Markup Language (XACML)	✓	✓	✓		
	Virtual Organization Membership Service (VOMS)	✓	✓	✓		
	WS-Security (Transport Level Security (TLS))	✓	✓	✓	✓	✓
Execution	Job Submission Description Language (JSDL)	✓	✓	✓	✓	
	OGSA-BES (Basic Execution Service)	✓		✓	✓	
	Distributed Resource Management Application API (DRMAA)	✓	✓	✓		
Data	OGSA-ByteIO	✓				
	GridFTP (defacto)		✓	✓		
Information	Web Service Resource Framework (WSRF)	✓	✓	✓		
	OGSA-RUS (Resource Usage Service)	✓			✓	
	OGSA-UR (Usage Record)	✓		✓	✓	

Table 6.1: Comparison of implemented standards in grid systems

grid-cloud interoperability. For interoperability between clouds, interoperability gateways and standards have been proposed. Many interoperability gateways are implemented within Cloud APIs. For example, Deltacloud [4] or Libcloud [5], define connectors for several cloud systems. In addition to interoperability gateways, several cloud standards emerged, e.g., *Open Cloud Computing Interface* (OCCI) [102] developed by the OGF and *Cloud Data Management Interface* (CDMI) [135] published by the *Storage Network Industry Association* (SNIA) [129]. However, cloud standards are still in their infancy and need to be improved further before global cloud stakeholders adapt them. Standardization effort by different organizations for cloud systems is described in [96].

In the grid domain, the interoperability gateway approach is implemented, e.g., in the HiLA for grid applications as well as in the *Java Grid Application Toolkit* (JavaGAT) [70] that allow to access grid core services of different grid middleware implementations. Table 6.1 illustrates the standards that are in use in popular grid software packages: GT4, UNICORE 6, *lightweight middleware for grid computing* (gLite) [62, 86], *Grid Operation System* (GOS) 3.2 [155], and Fura [131]. These are dominated by OGF standards associated with OGSA and WSRF. Furthermore, the *Public-Key-Infrastructure* (PKI) X.509 certificate system has found wide-spread adoption in all grid domains. The table also shows that the OGSA-BES, OGSA-RUS, and *OGSA-Usage Records* (OGSA-UR) standards have the broadest adoption amongst the middleware under consideration.

One aspect that is not captured by this table are grid systems that utilize very few standards. In the *United States of America* (USA), TeraGrid [139] and *Open Science Grid* (OSG) [107] are both dominated by custom-made software, or packages distributed through the *Virtual Data Toolkit* (VDT) [141]. In these cases, the only visible standards are GridFTP and the X.509 identity system. Even authorization by X.509 certificates is handled differently in different grids. In actuality, no operational grid systems rely exclusively (or even predominantly) on grid standards, but instead use a patchwork of custom-made and third-party software packages, expecting sites to be running the same version of the software to be interoperable.

It is also essential to note that higher level grid applications often rely on underlying grid middleware services in a way that inhibits interoperability, even when two implementations support the same underlying set of standards. The reason for this are incomplete implementations or patchwork standards environments.

6.2 Phase II: Integration of Grid Systems and IaaS Clouds

In the following, we describe the integration of grid systems and IaaS clouds, which is applied by the interoperability engineer in Phase II of the IAI process. We present evolutionary steps towards a feasible solution. Ideally, the different cloud and grid systems implement standards to allow an extension with systems implementing the same standards.

IaaS clouds provide virtual infrastructures for the deployment of virtual machine images to start and instantiate virtual machines. If several instances of a virtual machine are started, software for their balanced utilization as well as for executing and managing parallel tasks needs to be installed and configured so that the instances can be accessed via abstracted interfaces. In order to reuse the services provided by a grid system, we deploy grid core services into the IaaS cloud. The grid system can then be utilized to execute already existing grid applications within the cloud system and offer new services based on grid technology within the cloud.

An IaaS cloud that utilizes grid protocols can also be used to extend existing and configured grid systems on-demand during peak times. This setup is depicted in Figure 6.2. For utilizing the IaaS cloud as well as the grid system, applications and clients communicate via the same grid core service protocols. This allows an indirect communication between the grid and the IaaS cloud system based on the protocols implemented in the grid core services. The management of the cloud environment includes the instantiation of the virtual machine images, on which the grid core services are readily pre-configured. After cloud resources are instantiated, the grid core services are automatically started and registered with the information service of the existing grid. Afterwards, these services can be used by the grid client. In addition, cluster and storage resources can be deployed in the cloud system as local resource management systems and connected to the grid core services of the cloud. Through the deployment of the grid core services and the local resources management software stack, the IaaS cloud becomes a PaaS cloud that offers computing services as well as an API provided by the grid middleware.

To increase the usability of managing the grid resources deployed in the cloud, we integrate the management of cloud computing resources into the client layer of the grid environment. Figure 6.3 shows the extension of the grid client with the cloud client using the API of the cloud environment as well as the deployment of the grid core services into the cloud. The user uses only one client to setup the cloud environment and to send computational tasks to the grid core services deployed in a cloud environment. In the remainder of

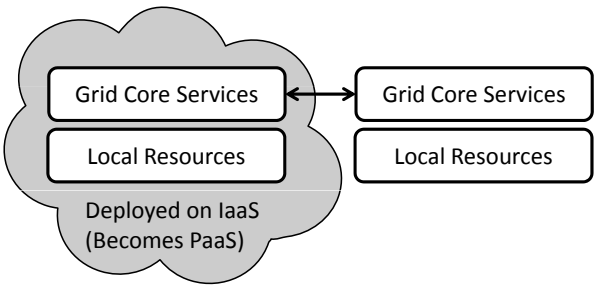


Figure 6.2: Grid-cloud integration on the infrastructure level

this thesis, we call the grid core services deployed in a cloud environment shortly grid-in-cloud-services.

Depending on the cloud deployment model, we identified different connectivity options for the integration. For the initial connectivity option, we deploy the grid-in-cloud-services in a public cloud. The public cloud is configured and managed through the cloud extension of the grid client via a custom-made cloud interface. After deployment, the grid-in-cloud-services are available directly within a public network and can be accessed via a grid gateway, which is an intermediate access point to multiple grid systems and their local resources. The grid gateway is required if more than one grid system is accessed, because the grid core services of both grid deployments do not interact directly with each other. A management entity, e.g., a grid scheduler is required to schedule tasks between the two grid deployments from an upper layer. In our case, the grid client offers scheduling functionalities to utilize both grid systems in parallel. The grid-in-cloud-services register with the information service inside the existing grid via the grid gateway. Then, the grid client polls this registry to utilize both systems through the grid gateway.

If the public domain of the cloud is accessible via the Internet, the grid-in-cloud-services are exposed to threats. For the protection of the grid-in-cloud-services, security measures on instance level are required. Each cloud instance needs to be treated with a specific security configuration. This includes the application of firewalls and security software for each instance. This design has a high complexity and overhead for security configurations and is, therefore, error prone.

We overcome these security issues through the deployment of the grid-in-cloud-services within a private cloud, which is itself located in a public cloud as shown in Figure 6.3. The grid-in-cloud-services are only accessible via the cloud gateway, which offers means to access the resources located in a virtualized dedicated network of our private cloud. All communications between the grid-in-cloud-services and the grid core services of our existing grid take place through the cloud gateway. The resources of the private cloud are not publicly visible, because they are logically separated from the public cloud. Each resource is protected by the security concepts of the private cloud. Only the cloud gateway needs to be properly configured to fulfill security requirements.

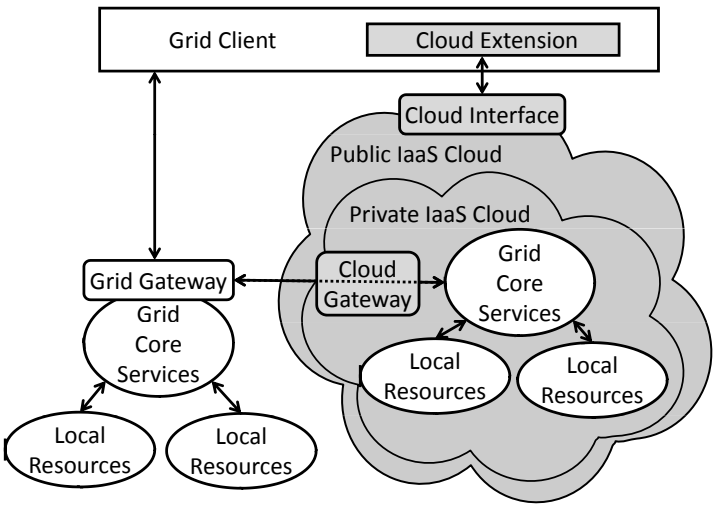


Figure 6.3: Grid core services deployment in a private IaaS cloud

The connection between the cloud gateway and the grid gateway is also exposed to threats. Encryption mechanisms can be applied on transferred data using an encryption service of a grid middleware. Another possibility is the extension of the local network of the existing grid with the private network of the cloud via an encrypted *Virtual Private Network* (VPN) tunnel. In the VPN scenario, the cloud gateway becomes obsolete. The grid-in-cloud-services appear locally inside the existing grid environment. This solution provides a complete separation from public networks and, therefore, a high level of security.

In the following, we describe the integration of specific grid implementations with specific IaaS cloud systems. We integrated several implementations with the aim to identify commonalities. A major commonality is that the interoperability gateway is implemented as a cloud extension in the client layer. The user can control and manage grid and IaaS cloud resources with an integrated client.

6.2.1 Integration of UNICORE and Amazon Web Services

In the first integration scenario, we integrated the grid middleware UNICORE 6 (Section 2.3.3.3) and the IaaS cloud AWS (Section 2.3.4.1). We decided to use these implementations because they are widely used. AWS interfaces are widely adopted by other cloud systems and can, therefore, be seen as de-facto cloud standards. UNICORE implements major grid standards, e.g., OGSA-BES and HPC-BP. In contrast, OCCI is still in its infancy and covers only basics of cloud interfaces.

Figure 6.4 shows the concrete integration based on the abstract solution depicted in Figure 6.3. Both the existing grid system and the private cloud deploy UNICORE, which can

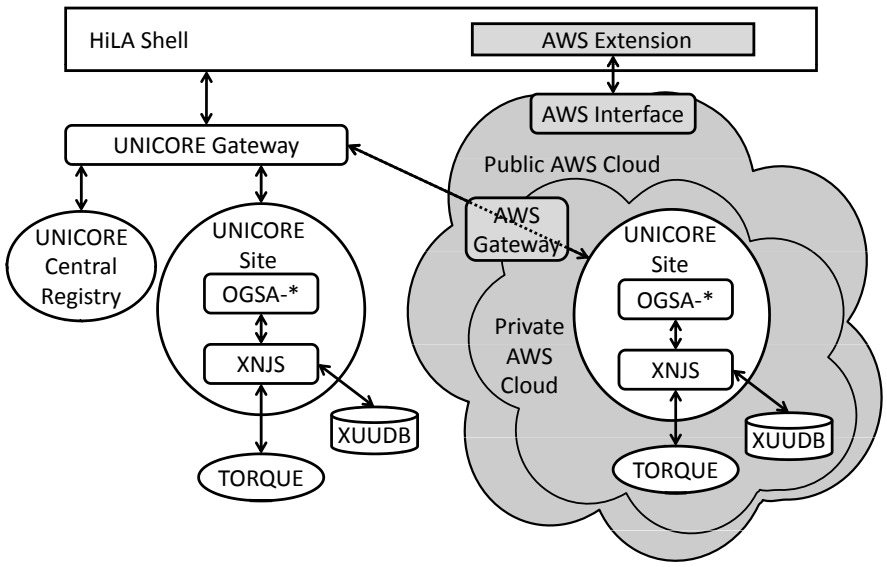


Figure 6.4: Deployment of the UNICORE grid middleware in the IaaS AWS cloud

be accessed via OGSA-* interfaces that access the XNJS. The *UNICORE User Database* (XUUDB) service performs authentication and authorization in the respective grid environment. In addition, the existing grid provides a central registry, where all UNICORE services are registered. In both environments, we deployed the open source TORQUE Resource Manager.

The HiLA shell allows for the development of grid clients using Java. Such clients can be extended with the cloud extension with little effort. To this aim, we utilize the HiLA shell to implement our grid client, which we extended by utilizing the AWS API. HiLA provides a uniform way to access grid core services of different grid middlewares including UNICORE. This facilitates the goal to support more than one grid system or cloud system using the same cloud-extended grid client.

The AWS cloud extension needs to fulfill two functionalities: the management of the AWS environment as well as the control of UNICORE resources in the cloud in an automated manner. These functionalities are implemented as AWS extension for the HiLA shell in a package of Java classes. The classes for the automatic AWS cloud management contain methods to configure, start, and stop the AWS environment utilizing the AWS API.

The classes also allow the instantiation of more than one UNICORE server in the AWS environment. Figure 6.5 shows the configuration of our solution. We establish a private subnet for each UNICORE server. We use a 24-bit subnet mask to be able to configure 20 private subnets, which is the maximum number of allowed AWS subnets. In each subnet, a maximum of 254 instances can be started. The instances can be used for the deployment of either TORQUE resources or further UNICORE servers. The UNICORE servers communi-

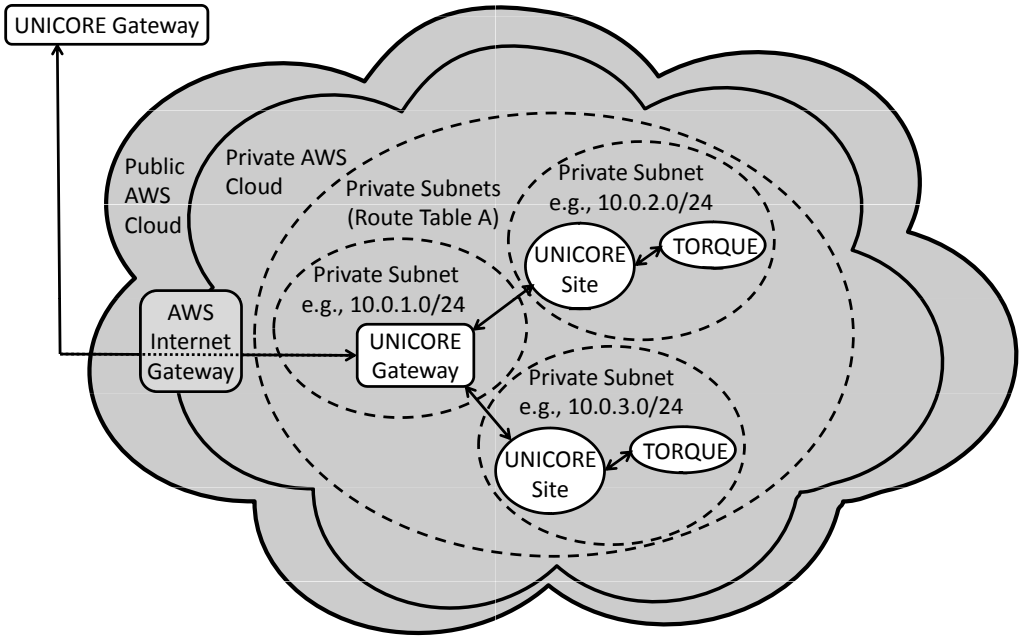


Figure 6.5: Deployment of the UNICORE grid middleware in the IaaS AWS cloud in multiple private subnets

cate with the UNICORE gateway of the existing grid system through the internal UNICORE gateway, which uses the AWS Internet gateway. The UNICORE gateway is part of the UNICORE package and can be used for the communication of several UNICORE grid systems. Public IP addresses are not needed for the UNICORE servers within the private subnets. Only the UNICORE gateway within the private cloud needs a public IP address to register directly with the external UNICORE gateway. To access the UNICORE servers deployed in a private subnet, a NAT instance with a public IP address can be started. Afterwards, all instances within a private subnet can be accessed as well as communicate with services outside the private cloud using the NAT service.

By separating UNICORE servers into different subnets, we can serve organizations independent of each other. The separation provides load balancing, isolates behavior of different applications, and gives flexibility in arranging UNICORE and TORQUE resources.

The classes for the automated deployment of UNICORE resources on AWS instances contain methods to configure and to start the UNICORE components within the AWS cloud environment. This includes the configuration of appropriate users on each instance, as well as the configuration of the UNICORE gateway and the automatic registration of the started UNICORE components in the registry of the existing grid system. In addition, we configured the UNICORE TSI [142] for using the TORQUE resource management system. We deployed the TORQUE software manually.

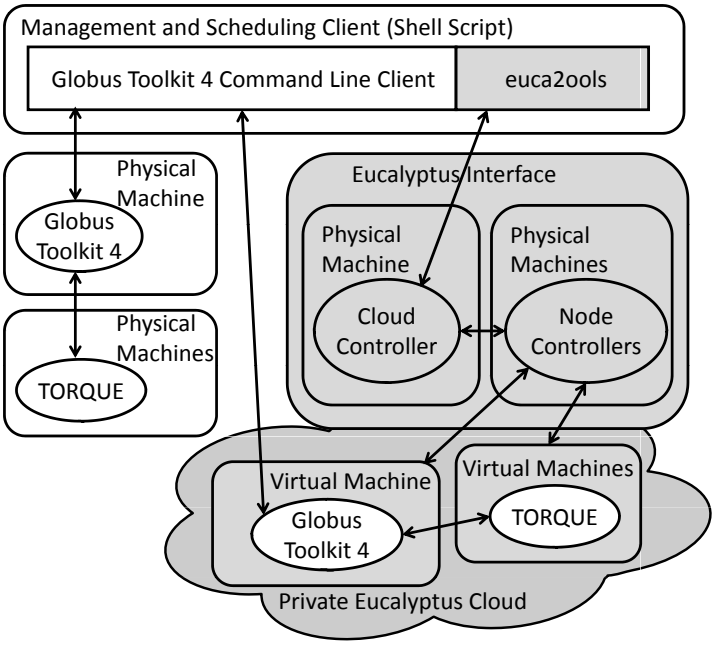


Figure 6.6: Deployment of GT4 grid middleware on Eucalyptus cloud resources

6.2.2 Integration of Globus Toolkit 4 and Eucalyptus

In the second integration scenario, we deployed GT4 (Section 2.3.3.2) and TORQUE cluster resources in a cloud configured with the open source cloud software Eucalyptus (Section 2.3.4.2). We choose GT4 since it is also a commonly used grid middleware implementing OGSA standards. In addition, Eucalyptus allowed us to have full control of our cloud environment.

For the integration of GT4 and Eucalyptus, Figure 6.6 shows the instantiation of the design depicted in Figure 6.3. We built the cloud system from scratch. Figure 6.6 shows the physical machines that are not directly visible in the cloud. The virtual machine image of the grid front end is configured with the GT4 middleware and a TORQUE server. Both are pre-configured with default values.

We developed a command line client based on shell scripts for the management of the cloud system and task management for the grid layer, since GT4 does not support HiLA. This client utilizes the command line client of GT4 for job submission and the command line tool euca2ools [45] to configure and start the Eucalyptus cloud. It also includes a scheduling module. The process to be performed and the deployment of the GT4 resources on Eucalyptus cloud resources are similar to the ones of UNICORE on AWS resources. The Eucalyptus cloud is deployed as a private cloud. The GT4 server and the TORQUE services are started and configured automatically. The GT4 resources in the Eucalyptus cloud can be used independent of the existing grid or in combination with it. The grid client is responsible to schedule tasks to both systems.

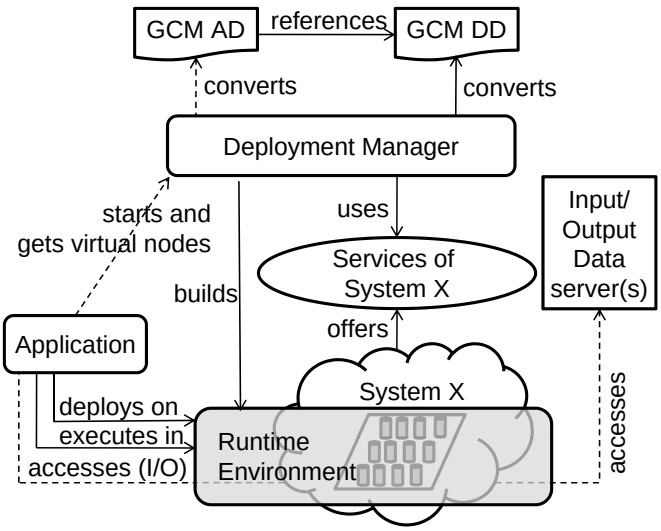


Figure 6.7: GCM Architecture

6.2.3 ETSI Grid Component Model

In the third scenario, we propose an integration of grid-cloud interoperability based on the ETSI GCM standards (Section 2.3.3.4). The GCM standards GCM DD and GCM AD provide formal specifications of resource information for the involved and possibly heterogeneous systems, e.g., grids and clouds. The content and concepts used in the GCM DD have been derived by abstracting different proprietary interfaces offered by commercial products in the grid, cloud, and cluster computing domains. The key aspect of the GCM specification is the mapping of this abstract interface to different proprietary interfaces of these systems as well as interfaces standardized for this purpose outside of ETSI, e.g., OGSA-BES. Figure 6.7 shows a generic GCM architecture, which focuses on the GCM AD and GCM DD. It introduces the deployment manager to illustrate the likely separation of GCM descriptor processing and provision of the actual resource by the involved systems.

The GCM deployment manager forms an interoperability gateway. The deployment manager itself is not standardized and an implementation of it does not exist, yet. Ideally, the deployment manager should provide scheduling and application management functionalities to administrate all connected systems and execute distributed applications based on the GCM AD and GCM DD. Even if a deployment manager is not implemented, the GCM AD and GCM DD standards provide a good baseline for grid and IaaS cloud interoperability. The GCM standards define the specification of deployment information and not an interface for the deployment.

The GCM DD describes the resources that can be requested for the deployment of an application on one or more systems. It is converted conceptually by the deployment manager

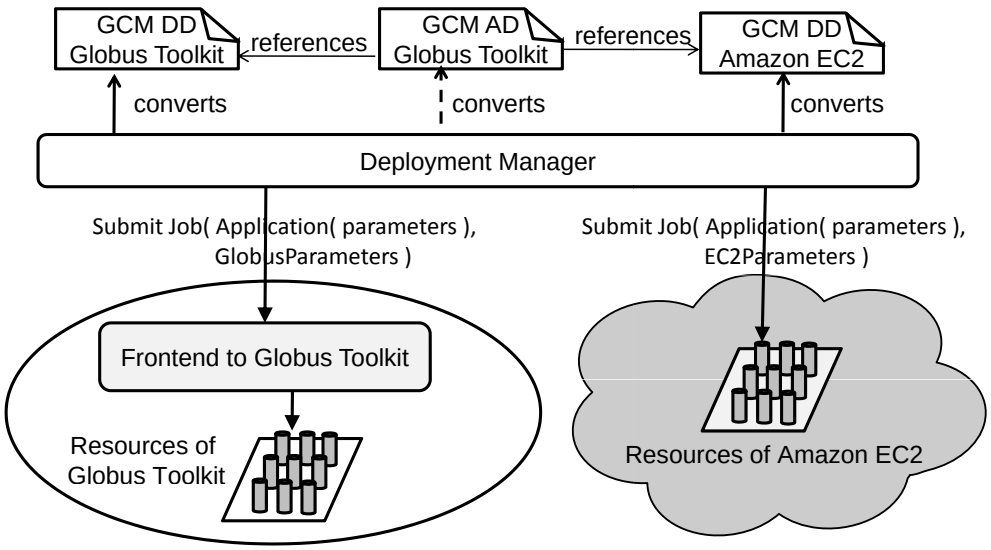


Figure 6.8: GCM deployment for Globus Toolkit and Amazon Elastic Compute Cloud

into the invocation of specific system services or commands to reserve resources from the specified system(s). The GCM differentiates between systems with direct access to their computing resource, as in the case of an IaaS cloud or a set of desktop computers, and indirect access by using a job scheduler, as in the case of a grid system.

For the application deployment, a GCM AD specifies the mapping of virtual nodes to real resources as well as the location(s) of input and output data server(s). If a GCM AD is provided, it is used to establish the runtime environment, which is required for executions of an application [34].

An example deployment scenario of GCM for the grid system GT4 and the IaaS cloud AWS EC2 is shown in Figure 6.8. The attributes for describing a GCM DD for GT4 have already been specified in the GCM standard [33]. An exemplified GCM DD is depicted in Listing 6.1. GT4 is contacted via a *Secure SHell* (SSH)-bridge to access the GT4 frontend. A UNIX-based operating system runs on each computing node, where each contains four processors, which are represented by the element `hostCapacity` of the attribute `host`. Since GT4 is a system with indirect resource access, the total number of available processors is not specified.

A GCM DD for the AWS EC2 is depicted in Listing 6.2. The scheme was not available and had to be developed for this setup. The successful deployment of this GCM DD builds the foundation for the extension of the GCM standards for AWS EC2. Since AWS EC2 is a system with direct resource access, the number of included computing nodes needs to be specified. In our example, the AWS EC2 contains ten computing nodes, which are based on a Windows operating system. The system is accessed via an SSH-bridge. Both presented DDs can be merged into one GCM DD file.

```
<?xml version="1.0" encoding="UTF-8"?>
<GCMDeployment xmlns="urn:gcm:deployment:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:gcm:deployment:1.0
    http://etsi.org/schemas/GCMDDSchemas/extensionSchemas.xsd ">
  <environment>
    <javaPropertyVariable name="user.home" />
  </environment>
  <resources>
    <bridge refid ="globusGateway" />
    <group refid="globusGrid">
      <host refid ="ComputeNodeUnix" />
    </group>
  </resources>
  <infrastructure >
    <hosts>
      <host id="ComputeNodeUnix" os="unix" hostCapacity="4">
        <homeDirectory base="root" relpath="${user.home}" />
      </host>
    </hosts>
    <groups>
      <globusGroup id="globusGrid"
        hostname="globus.grid.local"
        bookedNodesAccess="ssh"
        queue="free">
        <maxTime>5</maxTime>
        <stdout>./output</stdout>
        <stderr>./ error </stderr >
      </globusGroup>
    </groups>
    <bridges>
      <sshBridge id="globusGateway"
        hostname="grid.informatik.uni-goettingen.de"
        username="globususer" />
    </bridges>
  </infrastructure >
</GCMDeployment>
```

Listing 6.1: GCM Deployment Descriptor for Globus Toolkit

6.3 Phase III: Interoperability of the Grid Component Model

We only applied simple test frameworks to the integration scenarios for UNICORE and AWS as well as for GT4 and Eucalyptus. We checked if the configuration of the cloud environment and the deployment of the UNICORE resources in the AWS cloud were performed as expected by the cloud extension of the grid client. In addition, we submitted computational tasks into the UNICORE grid deployed in the cloud successfully. We validated the GT4-Eucalyptus system and client by submitting parallel grid tasks to resources of the existing GT4 grid system and on GT4 resources deployed in the Eucalyptus cloud successfully. This shows that it is feasible to integrate grid systems with IaaS clouds.

```
<GCMDeployment>
  <environment>
    <javaPropertyVariable name="user.home" />
  </environment>
  <resources>
    <bridge refid ="amazonCloudGateway" />
    <group refid="amazonCloud">
      <host refid ="ComputeNodeWindows" />
    </group>
  </resources>
  <infrastructure >
    <hosts>
      <host id="ComputeNodeWindows"
        os="windows" hostCapacity="1">
        <homeDirectory base="administrator" relpath="${user.home}" />
      </host>
    </hosts>
    <groups>
      <amazonCloudGroup id="amazonCloud"
        hostList ="node-[01-10]">
      </amazonCloudGroup>
    </groups>
    <bridges>
      <sshBridge id="amazonGateway"
        hostname="aws.amazon.com"
        username="amazonuser" />
    </bridges>
  </infrastructure >
</GCMDeployment>
```

Listing 6.2: GCM Deployment Descriptor for Amazon Elastic Compute Cloud

In Phase III of the IAI process, we focus on the interoperability assessment of the GCM AD and GCM DD standards, because they provide a good base for interoperable grid and cloud systems. The use of the GCM standards as a basis for interoperability test specification allows the development of a generic interoperability test system independent of the concrete system implementation of grid and cloud. Within our test executions, the deployment manager was simulated by manual activities of the interoperability tester. The advantage of a standard is that we can obtain testable requirements easier than from several specifications of proprietary system interfaces. Such an assessment also allows us to validate the GCM AD and GCM DD. We mainly focus on application deployment and application execution on grid and cloud systems based on the GCM AD and GCM DD. We will partly follow the DAITS process that is presented in Section 4.4 (as long as it is applicable for manual testing). We only specify the test descriptions but will not implement an ETS and ATS. Even if we utilize the GCM standards, a variety of proprietary interfaces implemented by the different IaaS clouds and grid systems lacking common standards are involved.

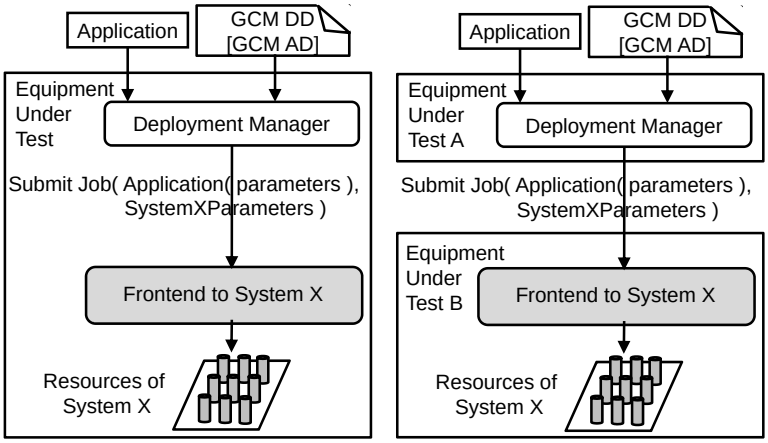
6.3.1 DAITS Process Prerequisites

As prerequisites of the DAITS process, test descriptions and test architectures are required. The library of automated interoperability tests and the identification of limitations are related to automation of the assessment and of EUT configurations and are, therefore, not needed. Instead, test applications that are executed on all involved systems are required to determine the real usage of resources and the behavior relevant to a test purpose covered by a test. The result of the test applications is evaluated manually.

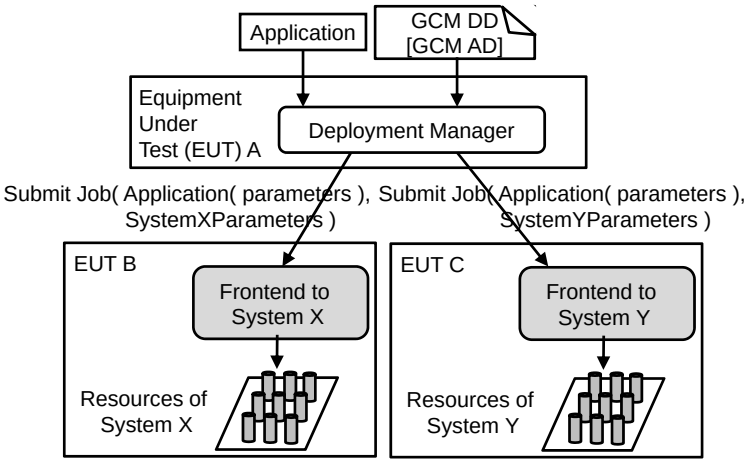
6.3.1.1 Test Architectures

For GCM interoperability testing, i.e., more specifically GCM-based application deployment, we determined four GCM test architectures. The test architectures specify structural aspects of a test and define EUTs that participate in a test as well as their communication. In all test architectures, the SUT consists of the deployment manager that is conceptually an EUT and at least one system that is an EUTs. The presented test architectures can also be used to assess other standards related to the deployment and execution of applications on grid or cloud systems, e.g., an OGSA-BES based interface between the deployment manager and the EUTs. The four test architectures are described in the following.

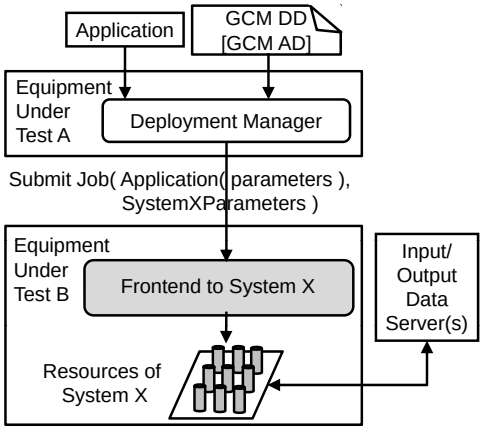
- **Single system**: In the test architecture "Single system", which is depicted in Figure 6.9 (a), the EUT contains a single system and the deployment manager. Access to the deployment manager, the system, the application, the GCM DD, and the GCM AD are available from one single physical machine. The purpose of this test architecture is to keep the complexity low to allow basic testing with minimal effort to establish the test architecture. The user uses the deployment manager to load the GCM DD and in case the test application is a GCM application, also the GCM AD as input. The user is logged locally into the system to establish the GCM runtime environment and submit jobs related to the application and the system. If a system provides indirect access to its resources, e.g., a grid system, a frontend is used to access its resources.
- **Single system with a bridge**: The test architecture "Single system with a bridge" depicted in Figure 6.9 (b) has two EUTs, where EUT A contains a deployment manager, which is connected via a bridge to EUT B, which contains a single system. In contrast to the test architecture "Single system", access to the deployment manager, the system, the test application to be executed, the GCM DD, and the GCM AD are distributed across two different physical machines. The user is connected remotely to the system in order to establish the GCM runtime environment and to submit jobs related to the application from the remote machine.
- **Two systems and bridges**: This test architecture is depicted in Figure 6.9 (c) and extends the test architecture "Single system with a bridge" with a second system. This test architecture has three EUTs, where EUT A contains the deployment manager,



(a) Single system (b) Single system with a bridge



(c) Two systems and bridges



(d) Single system with a bridge and I/O servers

Figure 6.9: GCM test architectures

EUT B contains the system X, and EUT C contains the system Y. Since the deployment manager controls both systems at the same time, it has to be connected to both systems via a bridge. This test architecture can be extended with several systems, which are then mapped to EUTs.

- **Single system with a bridge and I/O servers**: This test architecture is depicted in Figure 6.9 (d) and extends the test architecture "Single system with a bridge" with input and output data servers. The application can access the input/output data servers from the system.

6.3.1.2 Test Applications

For the assessment of the success and validity of each application deployment, a test application is executed on all involved systems. The purpose behind these applications is not to perform complex, real world, computational tasks but to produce output that allows determining the real usage of resources and the behavior relevant to a test purpose covered by a test. The test application is parameterizable to allow its reuse across multiple tests.

We determined four different kinds of test applications: single process batch job, parallel job, virtual node GCM application, and data manipulation GCM application [39].

The single process batch job starts a single process on a single processor and consumes the processor and memory for a given amount of time. The application's behavior including its execution time, the amount of memory to allocate, and the number of threads can be controlled by parameters. The application prints all information required to determine if a test execution has succeeded or failed either to the standard output or a file. This includes the application start time, the value of each parameter, and the identifier of the application. With this test application, resource deployment and resource usage can be evaluated.

The parallel job invokes an application that starts multiple processes. Each process is mapped to a single processor. The multiple processes application consists of one master process and multiple worker processes. The worker processes communicate with the master process so that the master process receives notifications from all worker processes. A notification should include the host name where the worker process runs and a timestamp. The number of worker processes to be created by the parallel application should be parameterizable. By default, the master process starts up as many worker processes as processors are available, i.e., one node less than specified in the GCM DD. That means that a parallel application requests all available resources. The parallel job prints all the information required to determine if a test execution has succeeded or failed either to the standard output or a file.

The virtual node GCM application starts a deployment as specified in the GCM AD and GCM DD. Once the deployment has been performed, it prints the information provided by each virtual node either to the standard output or a file. For each virtual node, the virtual node name, current number of nodes, and the information about each used node is required.

TP ID:	TP_GCM_DD_DA_PA_001
Clause Ref:	ETSI TS 102 827 V1.1.1 clause 7.1
Test Architecture:	Single system or single system with a bridge
Summary:	Ensure that a system with direct resource access provides a single processor as specified in the GCM DD

Figure 6.10: Test purpose "Single processor with direct resource access"

The data manipulation GCM application starts a deployment as specified in the GCM AD and GCM DD. It deploys a worker on each available node. Each worker reads the same input file from the remote or local input location as specified in the GCM AD. It creates a file with the same content as the input file into the remote or local output location as specified in the GCM AD. Workers should avoid file name conflicts and collisions in the output directory.

6.3.1.3 Test Purposes and Test Descriptions

Before we define the test descriptions, we develop test purposes that build the baseline of the test descriptions. We analyze the base standard and extract testable requirements that are used to specify test purposes. However, the GCM standard only defines the specification of deployment information and not an interface for the deployment. Therefore, the specification of test purposes for GCM descriptors is not a trivial task. In the case of GCM DD, the primary source for specifying test purposes is general information associated with resources, such as the number of offered processors or the number of threads per processor available for execution. The secondary source for specifying test purposes includes parameters that are common to a number of standardized GCM mappings to different systems, e.g., wall time and maximum memory. However, these might not be supported by each system. Therefore, a test purpose should not be specific to a single mapping. A third source for additional test purposes are variations of the sources for specifying test purposes mentioned above. The variations are based on resource access methods, i.e., direct vs. indirect and local vs. remote access. In this assessment, each test purpose is dedicated to one aspect of a specific requirement or concept defined in the GCM standard.

In Figure 6.10, an exemplified test purpose for GCM DD is depicted. In this case, the support of the direct resource access is a precondition and a GCM DD with a single processor reservation is the stimulus. The success of the application execution determines the success of the resource reservation.

A test purpose for GCM AD is exemplified in Figure 6.11. For this test, the support of the GCM AD is required. A GCM AD with a virtual node reservation is the stimulus. The test was successful if the test application is able to allocate the capacity of a virtual node as specified in the GCM AD.

TP ID:	TP_GCM_AD_VN_001
Clause Ref:	ETSI TS 102 828 V2.1.1 clause 5.2.2
Test Architecture:	Single system or single system with a bridge
Summary:	Ensure that a specific capacity of a virtual node (VN) is enforced as specified in the GCM AD

Figure 6.11: Test purpose "Specific capacity of a single virtual node"

In the development of GCM AD test purposes, (re)assessing of GCM DD information should be avoided. For example, the test purposes for GCM AD should be applicable independent of the method with which the resources of a system are accessed (directly or indirectly). This means that these test purposes focus on information and concepts specified in the GCM AD. Example sources for test purposes are the handling of virtual nodes and input/output data location.

Based on the test purposes, we develop the test descriptions that are a detailed but informal specification of the pre-conditions and test steps that are needed to cover one or potentially more given test purposes. The test description can also include a list of checks that should be performed when monitoring the EUT communication on standardized interfaces during the end-to-end test. In the case of GCM testing, this option is not directly relevant since the GCM standard does not intentionally define the interfaces between a deployment manager and a system. However, checks can be formulated if a system implements interfaces standardized for resource reservation and application execution by other standardization organizations, e.g., OGF and *Distributed Management Task Force, Inc.* (DMTF).

An exemplified test description for the GCM DD test purpose shown in Figure 6.10 is depicted in Figure 6.12. This test description details a test to check if a system with direct resource access provides a single processor as specified in the GCM DD. The pre-test conditions identify in addition to the features that are required to be supported by participating equipment to be able to execute this test, the requirements on GCM descriptors, as well as requirements on the parameterization of the test application. A complete list of the test descriptions can be found in [39].

6.3.2 Test System Design

From the test system design phase of the DAITS process, we can only conduct the definition of test configurations. For manual assessment, we also define compliance levels to classify up to which stage the functionalities are provided by an SUT.

Test Description		
Identifier:	TD_GCM_DD_DA_PA_001	
Summary:	Ensure that a system with direct resource access provides a single processor as specified in the GCM DD	
Test Architecture:	Single system or single system with a bridge	
Specification References:	ETSI TS 102 827 V1.1.1 clause 7.1	
Test Application:	Single process batch job	
Pre-test Conditions:	• System provides direct resource access • GCM DD contains a direct group description with `hostList` containing one host and host description with `hostCapacity=1` for the system • System has a processor available for use	
Test Sequence:	**Step**	**Description**
	1	User loads the GCM DD and starts the test application on the system using the deployment manager
	2	Verify that the system has created and executed the process
	3	Verify that returned application output is correct

Figure 6.12: Test description “Single processor with direct resource access”

6.3.2.1 Test Configuration

Test configurations are a refinement of the test architecture and are referenced during the specification of tests, which mainly specify behavioral aspects. Figure 6.13 shows a test configuration for GCM-based deployment based on the test architecture: “Single system with a bridge and I/O servers”. The SUT consists of at least two EUTs, which are the deployment manager and at least one system.

The different types of entities that compose the means of testing handle the provision of the GCM DD and GCM AD files to the deployment manager. These entities associated with the system to be tested evaluate responses from the deployment manager, and analyze the output produced by the application via their PCOs. In addition, the processes executed on each system, their interface(s) to the deployment manager, and the input/output server(s) are monitored during tests execution. The monitors are *Points of Observation* (PoOs).

6.3.2.2 Compliance Levels

The general test objective is to check that applications can be deployed and executed on a given system based on the information provided in GCM AD and GCM DD. A system can either provide direct or indirect resource access. To access a system, its protocol needs to

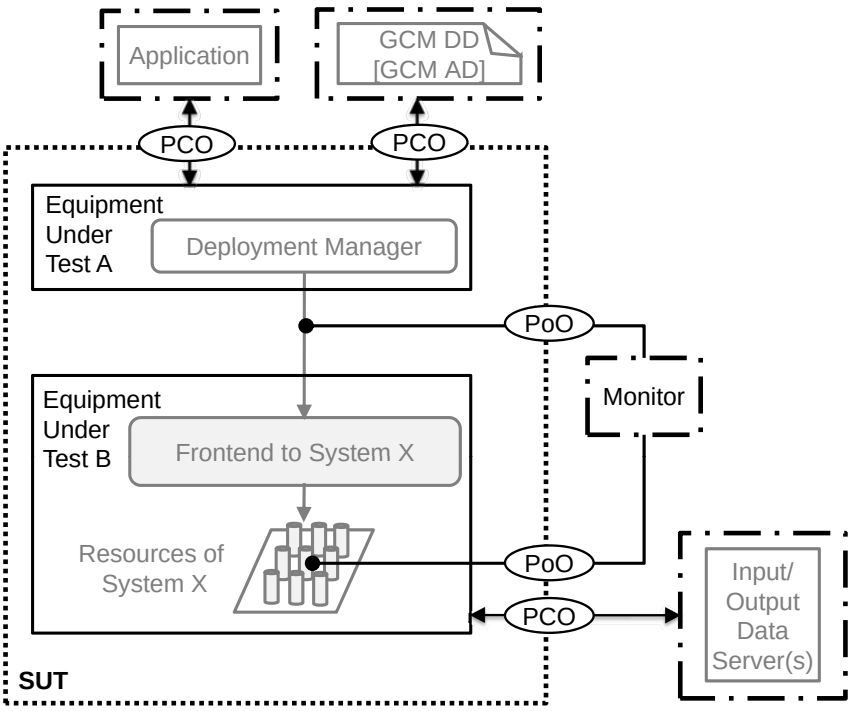


Figure 6.13: A test configuration for GCM-based deployment

be followed as specified in the GCM standard [33]. For a classification of functionalities that are provided by an SUT, we define compliance levels as follows:

Compliance by the system:

1. A system does not support properties described in GCM AD and GCM DD.
2. A system supports properties described in GCM AD and GCM DD, but these are converted in a manual manner.
3. A system supports properties described in GCM AD and GCM DD, and these are converted in an automated manner.

Compliance by the deployment manager:

1. Support of multiple systems fulfilling system compliance level 2.
2. Support of multiple systems where at least one of them fulfills system compliance level 3 and the others system compliance level 2 (at least one).
3. Support of multiple systems fulfilling system compliance level 3.

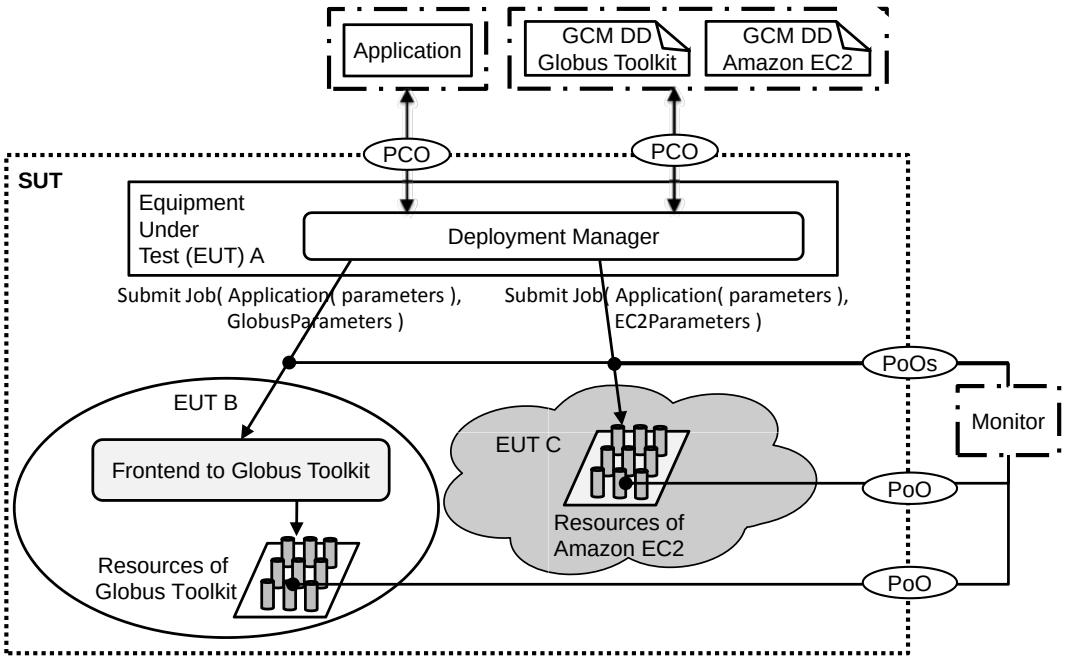


Figure 6.14: Test configuration for Globus Toolkit and Amazon Elastic Compute Cloud

6.3.3 Example Test Session

For an example test session and based on the test architecture “Two systems and bridges”, we developed a test configuration for the grid system GT4 and the IaaS cloud AWS EC2 derived from the scenario depicted in Figure 6.8. In the test configuration, which is depicted in Figure 6.14, EUT A is the deployment manager. EUT B contains the GT4 grid system and EUT C includes AWS *Elastic Compute Cloud* (EC2) cloud. For each system, a GCM DD is required.

We exemplify the test specified in the test description depicted in Figure 6.15. It will be checked if both systems provide multiple processors for a parallel application. Therefore, the parallel application allocates more than one processor in each system. The execution of the application will be logged by the monitor in order to evaluate the result of the test. The AWS EC2 system includes ten nodes as described in the GCM DD. The number of nodes in the GT4 system is not specified in the GCM DD. We configured the parallel test application to start ten processes on the AWS EC2 cloud and secondly four processes in the GT4 system, because the GT4 system needs to have at least one node with four processors. If all the processes have been started successfully and if the test application writes its output as expected, the test can be evaluated as successful.

Test Description		
Identifier:	TD_GCM_DD_DA_IA_PA_001	
Summary:	Ensure that a system with indirect resource access and a system with direct resource access provide multiple processors for a parallel application as specified in the GCM DD	
Test Architecture:	Two systems and bridges	
Specification References:	ETSI TS 102 827 V1.1.1 clause 7.1, 7.2	
Test Application:	Parallel job	
Pre-test conditions:	• One system provides indirect resource access • One system provides direct resource access • GCM DD contains one direct group description and one indirect group descriptions • Communication between the systems is supported • Systems have multiple processors available for use	
Test Sequence:	**Step**	**Description**
	1	User loads the GCM DD and starts the test application on both systems using the deployment manager
	2	Verify that the processes have been created and executed in both systems
	3	Verify that returned application output is correct

Figure 6.15: Test description "Multiple processors in systems with indirect and direct resource access"

6.3.4 Validation and Application of the Test System

The validation of interoperability test specifications usually takes place in testing events. For the GCM interoperability assessment, this validation took place in November 2009 as part of the ETSI Grids, Clouds, and Service Infrastructures event [29]. It provided a unique opportunity for standardization experts, operators, IT service providers, telecommunication equipment vendors to see available systems running. However, the tests need to be carefully selected according to the features of the involved systems. All pre-conditions of a test need to be evaluated to determine the applicability of the test. To speed up this process, an ICS should be established to allow system providers to specify supported features prior to a test execution and support automatic test selection. A test should be selected for execution if all of its pre-conditions have been ensured. Common types of pre-conditions in the GCM tests include constraints on:

- the GCM DD and/or GCM AD specifications,
- the system relating to the type of resource access, features that need to be supported by the EUTs, and available amount of resources,
- and the test application parameterization.

A test should be recorded as being not applicable if one or more pre-conditions are not met by one or more EUTs. A collection and specification of *Protocol Implementation Extra Information for Testing* (PIXIT) can be used to capture system specific aspects of a GCM DD such as the access details to a system and resource identifiers, and used to significantly speed up the execution of tests. The developer of an EUT states the PIXIT that includes information according to the
acEUT and its testing environment to enable runs of an appropriate test suite against the EUT [74].

Each grid and cloud system will be assessed under the same conditions based on the formalism of test descriptions. Applicable tests are executed by uploading a test specific application, providing system and deployment information, e.g., via GCM descriptors, and observing the execution of the application as specified in the test specification.

6.3.4.1 Application in an Interoperability Event

A requirement of the application of the GCM test specification is that a possible SUT needs to include an implementation of the GCM standard, which was not the case during the November 2009 event. However, the interoperability event included a variety of state-of-the-art systems that implement grid, cloud, cluster, or related technologies, which fit the idea of the GCM standard. Therefore, we compared and executed different state-of-the-art systems to determine and evaluate their similarities and differences. The goal was to feed the result of the demonstrations of the systems of the participated vendors back into the standard to make GCM applicable for these systems, which results in their interoperability.

In total, six exhibitors demonstrated their grid or cloud environments. One part of the demonstrations was the resource reservation and application deployment onto different systems. The basis for the evaluation was a questionnaire as well as use case scenarios defined in the ETSI GCM test specification. The questionnaire assessed interfaces for resource reservation and preparation of systems as well as standard support. The use cases included scenarios for systems offering direct and indirect access. Systems were not required to support ETSI GCM standards so that custom-made interfaces were used for application deployment.

The systems of the vendors who participated in the event mainly implemented a deployment manager for IaaS clouds. All the solutions provided a portal or *Graphical User Interface* (GUI) for resource reservation. For automated use, these have been realized either as a RESTful Web service, a *Command Line Interface* (CLI), or a Java/XML based API . In general, all the demonstrated systems shielded users from complex details of resource reservation. Resource provision and resource requests were handled separately. Handling of data transfer to and from the computing node was mostly done by the application, e.g., using the *Secure CoPy* (SCP) protocol or FTP.

6.3.4.2 Results from the Interoperability Event

We identified several issues of grid-cloud interoperability during the interoperability event. We categorized these issues into four main areas: resource request and access, standard support, identified needs in cloud standardization, and test automation.

Resource request and access were based on different requirements on resources such as computing, storage, or network resources and assessed by the test application referenced in the test description. The resources were selected based on requirements such as performance, *Service Level Agreement* (SLA), application types, and objectives. Fixed or common concepts between the systems could not be identified. The reason may be the application domain specific implementations of the systems. For example, while one system needed a detailed specification of resource requirements, another system only required a specification of a class defined for resource requirements. In addition, the transparency of the resource management differed. Therefore, there is a need for an appliance independent hypervisor that manages resources independent of the application.

Related to *standard support*, mainly ETSI GCM, OGF *Distributed Resource Management Application API* (DRMAA), and DMTF *Open Virtualization Format* (OVF) have been implemented for resource request and access. Most systems used a non-compliant default configuration, but also allowed adaptation to DMTF OVF. A few basic standards are supported by commercial cloud systems, since cloud computing is an emerging technology and standards are only slowly evolving. Most of the cloud systems provided proprietary RESTful Web Service and XML based interfaces to resources. This provides a good foundation for further standardization and extensions. Weak points of existing standards are that they allow too many options such as in the OGF *Job Submission Description Language* (JSDL) or that they require to fix the location of resources.

In the conducted interoperability event, we identified several *standardization needs* for cloud computing systems related to interoperability. A key area for standardization is an API for the provision of resources and for requests of resources. Open issues are the achievement of portable appliances of the hypervisor, i.e., the management of different virtual machine images and resources. Minor concerns include the lack of agreed terminology and the need for a strong common denominator.

For IaaS clouds, functionalities such as the management of resources, applications, and data but also common security (authentication and authorization), billing, and accounting interfaces need to be standardized. A cloud resource management standard should consider interfaces for the deployment of virtual machines including their start, stop, status requests, image format, monitoring, performance measurement, and access.

According to interoperable grid and cloud systems, an application should be able to use them simultaneously. For this, commonly agreed protocols are required to exchange information and to allow their management. A result would be a cloud/grid broker, which the user accesses to use functionalities of grid and cloud systems.

According to *test automation*, the presented test system has been mainly developed for the application in interoperability events. The tested systems can be seen as black boxes connected and accessible only by their interfaces. Automating test executions in such a configuration is challenging since agreed standards are not available for the usage of grid systems or clouds. Furthermore, in the setting of such events participants are usually known as late as two weeks before the event. However, test specification effort usually is concluded months before an event. Therefore, in this case study we conducted the test manually.

6.4 Related Work

We divide the work related to our contributions into the following areas: comparisons and integration as well as interoperability assessment of grid systems and clouds.

6.4.1 Comparisons and Integration of Grid Systems and Clouds

Foster et al. compare grid systems and clouds in a detailed analysis from different perspectives including architecture, security, and programming model [60]. Our contribution extends their architecture comparison by describing the direct relations of the architecture layers. Sadashiv and Kumar compare clusters, grids, and clouds by determining if the respective system has specific functionalities [124]. They do not describe the reasons for their determination. In contrast, we describe the comparison of grids and clouds based on the technical architecture of grids and clouds. Similarly, the comparison of Zhang et al. [156] does not provide a direct comparison as in our contribution.

Yamini et. al. present a method for scheduling jobs in an IaaS cloud extended grid environment [154]. Their approach is similar to our contribution. However, we determine connectivity options for IaaS cloud-grid. Jha et al. describe how clouds can be used as a semantical abstraction of grids [80]. It is not clear on which level of the cloud service model they operate. We give a technical description and implementation of the integration of IaaS clouds and grids. Ostermann et al. present an approach to extend the ASKALON grid with instances from three different IaaS clouds [111]. In contrast, we developed a generic model for the cloud and grid integration on different service levels, and implement two case studies for the infrastructure level integration for other grid system implementations than ASKALON. Nimbus Infrastructure [143] builds a compute grid only based on virtual machines. It mainly utilizes AWS and is deployed within a WSRF container. This approach is different to ours, because they create a computational cloud as a service inside the grid systems while we deploy the grid core services within a cloud.

6.4.2 Interoperability Assessment of Grid Systems and Clouds based on the Grid Component Model

Several interoperability and standard initiatives for grid and cloud computing systems exit. For cloud systems, these include the OGF OCCI Working Group [102], the IEEE Standards Association [72], the DMTF *Cloud Management Standards* [21], and the *Open Cloud Consortium* (OCC) [103]. The activities of major cloud standardization initiatives have been summarized in a report by the *International Telecommunication Union* (ITU) [76]. These standardization activities are diverse and each initiative chooses the flavors of cloud computing that fit their requirements best. This is one reason why the concepts of cloud computing are not fully agreed upon on.

Bernstein et. al. identified areas and technologies of protocols and formats that need to be standardized to allow cloud interoperability [9]. They call this set of protocols and formats Intercloud protocols because they should allow cloud computing interoperability. If this set of protocols will be commonly accepted, the GCM and the interoperability test system presented in Phase III of the IAI process could be adapted to improve cloud interoperability.

Merzky et. al. present application level interoperability between clouds and grids based on SAGA, a high-level interface for distributed application development [93]. The interoperability is achieved by cloud adapters. These adapters are specific to AWS EC2 and GT4.

Interoperability initiatives such as OGF *Grid Interoperability Now* (GIN) and standard bodies in grid computing are described in [118]. OGF interoperability test specifications for grids are often not defined and if so only for selected standards such as GridFTP. Also, they rather follow ETSI's notion of conformance testing than interoperability testing. To the best of our knowledge, an interoperability test system for such diverse domains of grid systems and clouds has not been published.

7 Interoperability of Grid and PaaS Cloud Systems

In this chapter, we analyze the interoperability between grid systems and PaaS clouds with our IAI process. Usually, PaaS clouds scale with the number of users, but not with computing intensive tasks. In our interoperability scenario, we use grids to scale PaaS clouds by submitting such tasks into the grid for processing. Therefore, no new instances for HPC need to be started in the cloud. The virtualization of the resources in the cloud weakens the computational performance and, therefore, lessens the efficiency of the computational resource utilization. We present a solution for grid-PaaS cloud interoperability, which is unique and has, to the best of our knowledge, never been implemented before. Therefore, we advance the state-of-the-art of grid-cloud interoperability. Because grid systems and PaaS clouds both provide access via Web services, we assess their interoperability with our methodology for automated interoperability testing. With this assessment, we contribute to the state-of-the-art of interoperability testing in grid and cloud systems. The results can be used to improve both systems and provide a basis for standardization.

In Section 7.1, we describe Phase I of the IAI process, which includes the comparison of grid systems with PaaS clouds. In Section 7.2, we develop a solution for interoperability of grid systems and PaaS clouds as part of Phase II of the IAI process. We exemplify the development with the UNICORE grid system and the GAE PaaS cloud. In Section 7.3, we apply Phase III of the IAI process and assess the interoperability solution for UNICORE and GAE. We conclude this chapter with related work in Section 7.4.

7.1 Phase I: Comparison of Grid Systems and PaaS Clouds

By direct comparision of grid systems and clouds, we map the abstracted functionalities of the conceptual grid model to the cloud layers. The mapping is depicted in Figure 7.1. The core services as well as low level computing and storage management services of the local resources of the grid reside in the PaaS cloud level, because they provide interfaces to use and manage resources similar to the functionality exposed by the control interfaces of PaaS clouds. While in grid systems, the interfaces are exposed directly to the developer, in clouds they are offered transparently via control interfaces. In grid systems, the grid core services provide delegated control over the local resources.

A PaaS cloud varies highly in its abstracted functionalities and usually hides the complexity of scheduling inquiries from users or developers. Hence, a grid scheduler providing

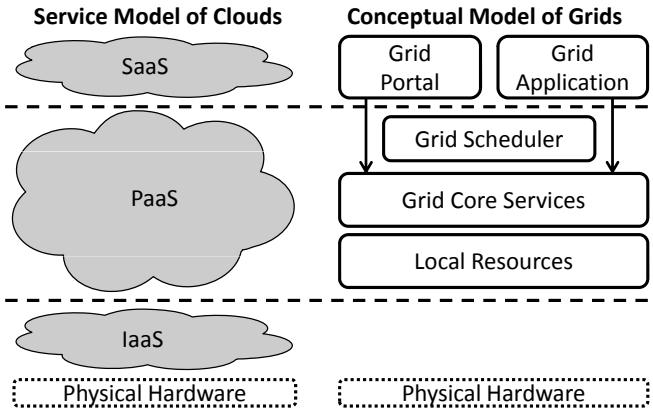


Figure 7.1: Comparison of the conceptual models of grid and cloud systems

similar functionality in the grid resides in the same layer as in a PaaS cloud. PaaS clouds scale with the number of users using applications deployed on the platform. In a PaaS cloud, the users are not able to manage the underlying server or operating systems. They have only limited possibilities to configure the application-hosting environment [92]. On top of the grid model are grid portals and applications, which are on the same level as SaaS clouds. They provide transparent access to resources via interfaces that are usually implemented using Web protocols. An application that is developed with a PaaS cloud is an SaaS application.

To our knowledge, no solutions for interoperability between grid systems and PaaS clouds exit. We can only refer to the standards organizations and interoperability initiative that we described in Section 6.1.2. None of them considers interoperability between grid and PaaS clouds.

Regarding PaaS cloud interoperability, an initiative of several companies that develop an open API for PaaS Application Management was started very recently [18]. In August 2012, they released the first version of the *Cloud Application Management for Platforms* (CAMP) specification [15]. The implementation of the CAMP specification facilitates the portation of a developed PaaS cloud application from one PaaS cloud provider to another.

7.2 Phase II: Interoperability of Grid Systems and PaaS Clouds

In the following, we describe how grid systems and PaaS clouds can interoperate. The application developer can utilize the control interface of a PaaS cloud to deploy a grid library within the PaaS cloud to access the grid core services. Alternatively, the grid library is integrated into the runtime environment by the PaaS cloud provider. The grid library can be imported by the cloud application to access grid resources. Figure 7.2 shows the design, where the grid core services are accessed via a grid library from the level of the PaaS cloud.

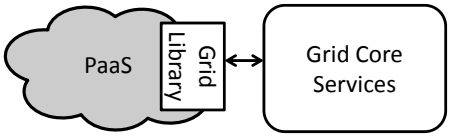


Figure 7.2: Grid-cloud integration on the platform level

The HiLA for grid applications and JavaGAT are examples for libraries to access grid core services of different grid middleware implementations.

We determined a generic workflow how to access a grid system from an SaaS cloud application that has been created with a PaaS cloud. First, the user selects the SaaS cloud application, which invokes computing intensive tasks specified in a job description. The cloud application is executed with the parameters in the PaaS environment that the user entered through the SaaS cloud interface. These parameters can include the specification of input data that is uploaded to the cloud storage and used by the cloud application. Before the job is sent from the PaaS cloud to the grid system, preambles need to be executed. These include the polling of the grid information service for available grid resources and the transfer of uploaded or required input data from the cloud storage to the grid system. After the management environment for the job is configured, the job is executed in the grid system. After completion of the grid job, the output is transferred to the cloud storage and, afterwards, presented to the user in the SaaS cloud interface. Depending on the length of the job run-time, the user could also be notified via email after termination of the job.

Related to the security, we identify major issues in this scenario. A user needs to be authenticated to be able to submit a job to a grid and authorized for the execution on a grid resource. The same applies for the input and output data that needs to be read as well as written in the respective other system. Ideally, both the grid system and the PaaS cloud use the same authentication service. Otherwise, a mapping between the user's credentials needs to be implemented.

Compared to the number of executions of the job in the grid system, the output of the grid application is usually accessed more often, because usually the results have a higher value than the input data. Our solution also provides a user SaaS interface within the PaaS cloud that allows to access data that has been calculated in the grid. This way the scalability with respect to the user access of the PaaS cloud is utilized to facilitate repeated and an increasing number of accessing the data produced in the grid. If the cloud system implements the five essential properties of cloud [92], it is ensured that the data is always accessible via the PaaS cloud. The data can be shared in the cloud with other users and be integrated using cloud interfaces.

We selected a UNICORE grid system and GAE as a PaaS cloud provided by Google for the implementation of an interoperability solution for grids and PaaS clouds. UNICORE implements already a set of standards and is also widely utilized by researchers. GAE

Figure 7.3: Schematic design of gateway-based GAE-UNICORE interoperability

does not implement a standard API similar to other PaaS cloud providers. The reason is that standards organizations focus on the standardization of the interfaces of the IaaS cloud layer. Only recently, CAMP was specified. However, within GAE, it is possible to use OpenID [108], which is an open standard that describes how users can be authenticated in a decentralized manner using a single digital identity for the authentication with multiple providers.

A deployment of libraries, e.g., HiLA, in the GAE PaaS cloud is not feasible, since GAE enforces the following restrictions [67]. GAE does not allow invoking threads. In addition, the communication takes only place over a specific port using HTTP. This means that socket connections, which are required for the HiLA library, are not allowed. Therefore, we implement an interoperability gateway for GAE-UNICORE interoperability. Another minor limitation is that GAE stops running Web applications in the cost-free version after a 30 second timeout automatically.

Figure 7.3 shows a schematic design of the integration of UNICORE and GAE. Since both systems are based on Web services, we implemented the interoperability gateway as a Web service as well. This offers many benefits as the interoperability gateway is on the same level of abstraction as both systems. Therefore, the protocols to communicate with the interoperability gateway resemble the protocols of the grid and cloud services, which simplifies the translation.

The deployment of the different services within the GAE and UNICORE systems as well as their interoperability gateway Web service is shown in Figure 7.4. GAE deploys different Web services, e.g., for authentication and for storing data on the GAE cloud storage named Blobstore. Using these services, service-based SaaS cloud applications can be developed. In our case, this cloud application uses computational resources of the UNICORE grid system. It sends a request to start a UNICORE computational job to the Web service interoperability gateway via HTTP to the Apache HTTP Server, which is deployed on a physical machine. The HTTP server deploys an application server (Apache Tomcat), which hosts the Web service for translating messages between GAE and UNICORE. The Web service interoperability gateway builds standardized JSDL descriptions that are used within HiLA calls that are sent to the already running UNICORE environment to start computational jobs.

A workflow of using a UNICORE grid system from a GAE PaaS cloud is depicted in Figure 7.5. After the user opened the related SaaS application, the Web server of GAE invokes the underlying GAE-UNICORE service that makes a request to the GAE-UNICORE

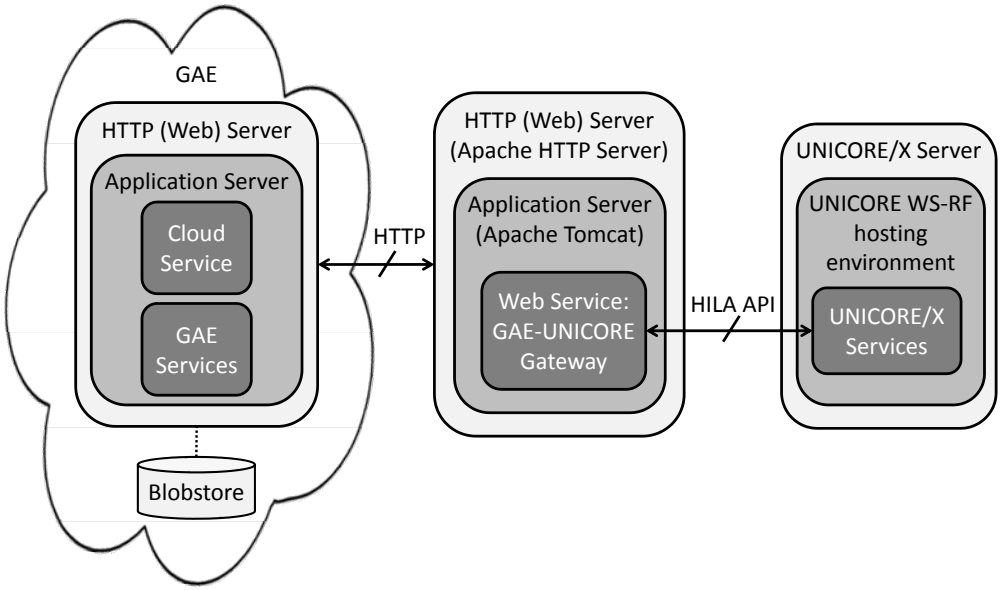


Figure 7.4: Architectural design of gateway-based GAE-UNICORE interoperability

gateway service to get data about available UNICORE resources and services. The GAE-UNICORE gateway service itself translates the request and sends it to the respective UNICORE service. For the response, the interoperability gateway translates the messages into a format understandable by the GAE-UNICORE service. Based on the response and the availability of UNICORE services, specific GAE-UNICORE services are presented to the user within the SaaS cloud application. The user selects one of these services, specifies parameters that will be transparently translated for the UNICORE service, and possibly uploads a file as an input for the UNICORE service. The file is stored via the GAE blobstore on GAE storage using the StoreSingleFile service. The user initiates the UNICORE grid job indirectly by sending a request with required parameters to the GAE-UNICORE service, which sends the job request to the interoperability gateway. The interoperability gateway accesses the GAE Blobstore via the BlobServe service and puts the input file on storage that is accessible by the UNICORE services. It creates the corresponding JSDL description from the user request and submits the job to the UNICORE system. Afterwards, the interoperability gateway polls for job completion and stores the output on GAE storage. The output is then presented within the SaaS cloud application. If the job execution time is relatively high, the user is notified about job completion. The output is accessible via the BlobServe service.

We used the application *Persistence of Vision Raytracer* (POV-Ray) [112] that renders scenes into images in the UNICORE grid systems. The size, the scene file, and other render related data required by POV-Ray is specified or uploaded respectively in the GAE cloud

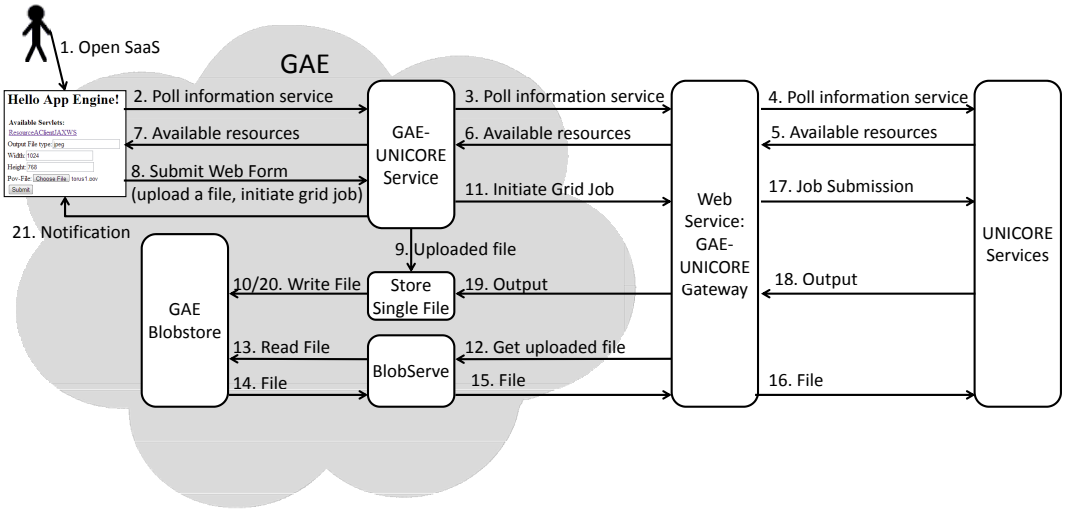


Figure 7.5: Message flow for a GAE-UNICORE interoperability scenario

system via the SaaS application. The render job is done within the UNICORE grid from which the output image is transferred via the GAE-UNICORE gateway to the GAE Blobstore.

7.3 Phase III: Automated Interoperability Testing of Grid Systems and PaaS Clouds

In this section, we assess the interoperability gateway for UNICORE grid systems and GAE PaaS clouds by utilizing the methodology for automated interoperability testing. Both systems provide Web service interfaces that can be assessed in an automated manner through protocols like HTTP and SOAP, which are open and whose basic structures are standardized. Following the test design guidelines and applying the generic environment for automated interoperability tests, we instantiate the DAITS process to develop an interoperability test system using TTCN-3. This test system builds the baseline for the assessment as well as improvement of grid–PaaS cloud interoperability exemplified with GAE-UNICORE systems.

7.3.1 DAITS Process Prerequisites

For the development of an automated interoperability test system for GAE-UNICORE interoperability, four prerequisites are required: the LibIot library, the test architecture, the test descriptions, and the limitations. The LibIot library is imported by the TTCN-3 ATS for GAE-UNICORE interoperability. A GAE-UNICORE interoperability test architecture

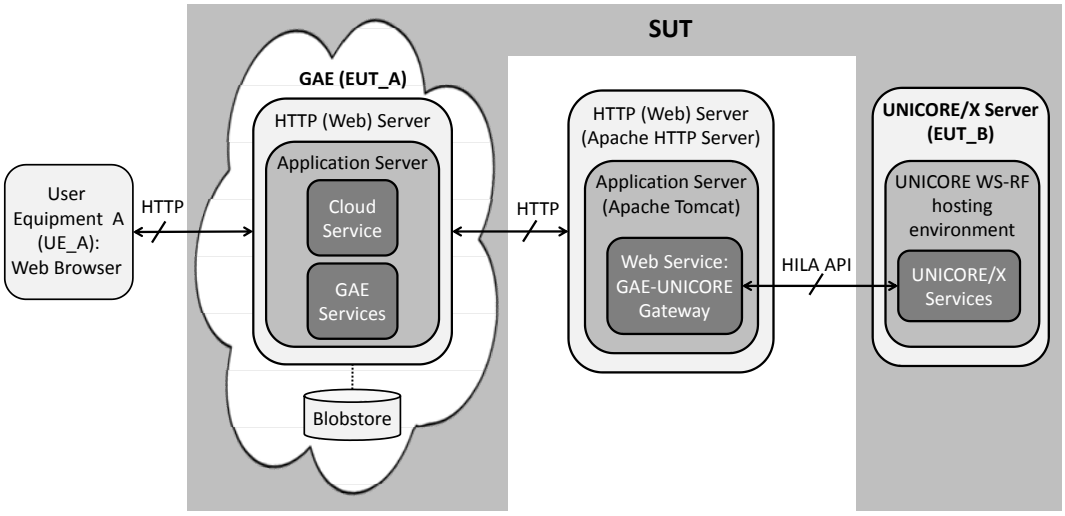


Figure 7.6: Test architecture: interworking grid–PaaS cloud system exemplified with GAE and UNICORE

is depicted in Figure 7.6. The test designer identified the GAE PaaS cloud as EUT_A and the UNICORE grid system as EUT_B. Both are connected to each other via networks as well as via the GAE-UNICORE gateway. The GAE PaaS cloud system is also connected to the UE_A, e.g., a Web browser.

Based on the scenario depicted in Figure 7.5, the test designer developed the test description for the interoperability test "GAE submits a grid job to the UNICORE system". Figure 7.7 shows this test description, which is used as a running example for the application of the DAITS process throughout this section. The interoperability test purpose is to check if the GAE cloud sends a computational grid job to a UNICORE system correctly as well as receives its output. The test description does not contain the referenced standard, since we apply the test to proprietary non-standardized interfaces.

The test sequence includes the verification of the interoperable functionality, which was specified in Phase II. UE_A invokes the cloud service of EUT_A, which itself submits a job to EUT_B. As a result, EUT_B executes the job and sends the produced output back to EUT_A. In order to verify the job submission, the job execution, as well as the transmission of the output, the verification steps that are described in the Steps 2, 3, 4, and 5 of the test description need to be done. Messages that are expected while executing the test sequence are specified in the conformance criteria. The criteria refers to one conformance test purpose, which checks the attributes of the HTTP message received by EUT_B and sent by UE_A. Therefore, we check the messages of the interface between EUT_A and EUT_B, as well as between UE_A and EUT_A.

Test Description		
Identifier	TD_GU_IOP_002	
Summary	Google App Engine sends a grid job to the UNICORE grid site and receive its output	
Test Architecture	Interworking Grid– PaaS Cloud system	
Specification Reference	Not applicable	
Pre-test conditions	• Systems and interoperability gateway need to be deployed, up and running • Registration of the user	
Test Sequence	**Step**	
	1	User sends request of a grid job submission from GAE to the UNICORE site
	2	Verify that the Web Service of the Grid-Cloud Gateway receive a HTTP message including the grid job submission parameters
	3	Verify that the UNICORE site receives the job submission
	4	Verify that the UNICORE job has been started
	5	Verify that the Output of the grid job has been transferred correctly to GAE
Conformance Criteria	**Check**	
	1	TP_GC_002 in message flow steps 11, 14: ensure that { when {User sends Grid Job Submission HTTP Request to UNICORE} then { UNICORE site receives a Job Description containing application = "povray" and width = 1024 and height = 768 and input = chess.pov}}

Figure 7.7: GAE-UNICORE test description

We identified one main limitation for GAE-UNICORE interoperability. We base our assessment on a custom-made interoperability solution with the involvement of proprietary interfaces that are utilized to access the EUTs. These proprietary interfaces limit the automation of the test execution for other grid and cloud systems. However, HTTP is an open protocol format, which is used as an input for the EUT_A and can be monitored if the messages are not encrypted. We specified the content of the HTTP messages, which is the basis for the interoperation and possibly a standardized API.

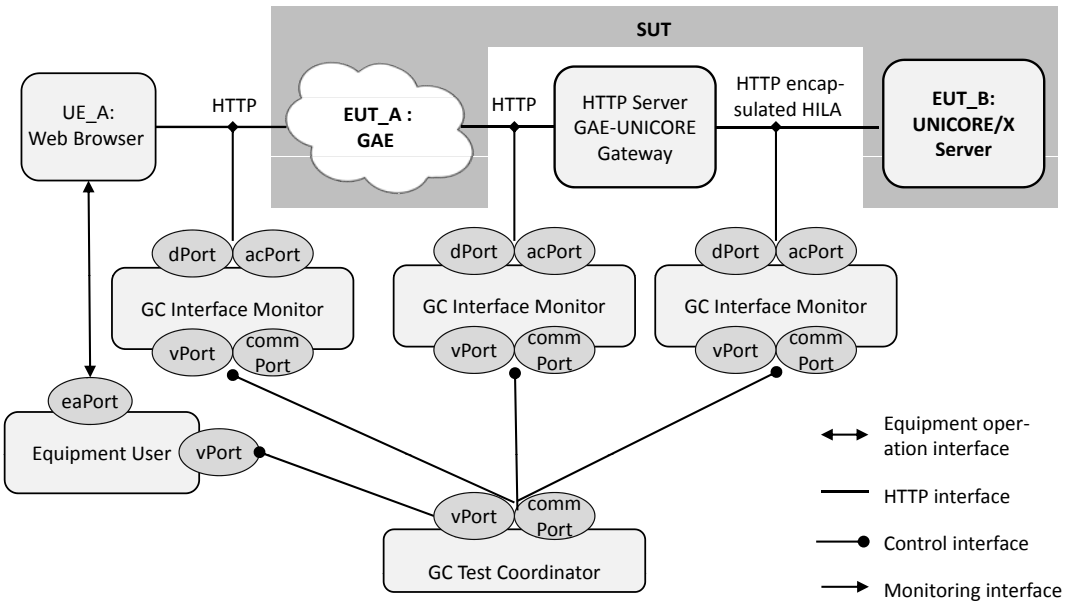


Figure 7.8: Test configuration for GAE-UNICORE interoperability tests

7.3.2 Interoperability Test Design

For the definition of a test configuration for GAE-UNICORE interoperability, the test system architect identifies required test entities. The test component types that are imported from LibIot are the same as for the IMS interoperability test suite as described in Section 5.2.2. They comprise the `EquipmentUser`, the `InterfaceMonitor`, and the `TestCoordinator`. The `EquipmentUser` is not instantiated for the EUTs, since the APIs of the EUTs are subject to frequent changes by the provider. In addition, the `EquipmentUser` should be able to handle EUTs in an abstract way independent of the provider. Implementing a handling of each involved system and its proprietary interfaces is hard to achieve and not the purpose of this test system. The `InterfaceMonitor` is used to monitor the HTTP interfaces between UE_A and EUT_A and between EUT_A and EUT_B.

For GAE-UNICORE interoperability, the test system architect extends the `InterfaceMonitor` component type to the `GCInterfaceMonitor` component type and the `TestCoordinator` to the `GCTestCoordinator` to add specific ports for cloud-grid interoperability. Figure 7.8 depicts the developed test configuration schematically. Each monitored HTTP interface is paired with a dedicated PTC of component type `GCInterfaceMonitor`, which receives all relevant message information from the system adapter. The UE is part of the system adapter and not assessed for interoperability. The `GCTestCoordinator` acts as the MTC.

```
type record Fields {
        charstring fieldName,
        charstring fieldValue,
        charstring contentType optional
}
type set of Fields FieldsList ;
type record RequestHeader {
          charstring method,
          charstring requestURI,
          charstring requestVersion,
          charstring pragma optional,
          charstring accept optional,
          charstring acceptEncoding optional,
          charstring acceptLanguage optional,
          charstring userAgent optional,
          charstring contentLength optional,
          charstring contentType,
          charstring referer optional,
          charstring connection optional,
          charstring host optional
}
type record HTTPRequest {
  RequestHeader header,
          FieldsList listOfFields optional
}
```

Listing 7.1: TTCN-3 types for an HTTPRequest

For message exchange between the test entities and the SUT, the same ports as for the IMS interoperability test suite are used as described in Section 5.2.2. They are the `acPort`, the `dPort`, and the `eaPort`. The test components are connected to each other via two ports: the `commPort` and the `vPort`. The `commPort` is used for communication between the test entities and `vPort` for sending local verdicts.

In addition, the test system architect defines the message structures. The message structures include the data types for HTTP: `HTTPRequest` and `HTTPResponse`. Both types define the structure for the header as well as for the body. The definition of `HTTPRequest` record type and its related type definitions are depicted in Listing 7.1. The HTTP type definitions are defined in a TTCN-3 library.

Based on the type definitions, the test system architect defines basic templates in the ATS. In our running example for the message check between the UE_A and EUT_A, the `method` field of the `RequestHeader` is set to the value `POST` and the `requestURI` field to `/resourceaclientjaxws`, which is the service to access the GAE-UNICORE gateway.

To finalize the interoperability test design, the test system architect needs to specify the module parameters that were defined by the test library implementer in LibIot. This includes especially the specification of the IP addresses of the involved EUTs as well as of the interfaces that need to be monitored.

```
1 f_mtc_userInvokeWebsite(v_ueA, v_accessInfo) // Step1
2 f_mtc_check_TP_GC_100_01_clientGae(v_clientGae, false); // Check1, Step2
3 f_mtc_check_TP_GC_100_01_GaeGateway(v_gaeGateway, false); // Check1, Step3
4 f_mtc_check_TP_GC_100_01_GatewayUnicore(v_gatewayUnicore, false); // Check1, Step4
5 f_mtc_checkOutput(v_ueA, v_accessInfo) // Step 5
```

Listing 7.2: Test body

7.3.3 Test Case Specification

Using our running example, we only highlight the main parts of the test case development, because it is very similar to the one described in Section 5.2.3 for IMS interoperability. A test case implements one test description and contains the three parts: test configuration setup, test body, and tear down. Within the test configuration setup, we instantiate the test configuration depicted in Figure 7.8, which relates to the test architecture specified in the test description. The preamble and the postamble are not specified within the test case, since they need to be executed manually. The tear down releases the test configuration.

The most important part is the test body, which is depicted in Listing 7.2. The `GCTestCoordinator` triggers the behavior by starting the test components passed as a parameter within each function call. The function `f_mtc_userInvokeWebsite` invokes the SaaS application that was developed with the GAE PaaS cloud to send jobs to UNICORE grid systems with application specific parameters. The rendering application POV-Ray is invoked. The variable is of component type `EquipmentUser`. The next three functions check the conformance criteria statement specified in the test description. The function `f_mtc_check_TP_GC_100_01_clientGae` checks the message sent by the SaaS application via HTTP between the UE_A and EUT_A. `v_clientGae` is instantiated as a component of type `GCInterfaceMonitor`. Within the monitored message, we expect the path to the POV-Ray, the UNICORE server, and the size of the output image. The other two functions check the correct translations of this message. The checks are implemented with a generic function, similar to the one we use for IMS interoperability testing, which is described in detail in Section 5.2.3. The last function depicted in Listing 7.2 checks if the output image has been transferred after job completion from EUT_B to EUT_A correctly.

In addition to the specification of the ATS, the system adapter and codec implementer specifies adaptation and codec functions to complete the ETS. This final step allows the automated execution of the interoperability test suite. The mapping between the test adapter and the test components is shown in Figure 5.10 and described in Section 5.2.3.

7.3.4 Validation and Application of the Test System

The test system is a prototype implementation and only contains three test cases. However, we validated the test system through reviews. Additionally, we applied the interoperability test system for an interoperating GAE-UNICORE system. We successfully executed the

test case represented by the test description depicted in Figure 7.7 in an automated manner. The test invokes the SaaS application that invokes the POV-Ray render application within a UNICORE grid. After the job is finished, the output of the POVRay render application is successfully written from the UNICORE grid into the GAE cloud storage. The messages were checked during interoperation in live mode automatically.

7.4 Related Work

To the best of our knowledge, no other approaches for achieving interoperability between grid systems and PaaS clouds exist. Hence, test systems to test the interoperability of grid systems and PaaS clouds do not exist either.

8 Conclusion

In this last chapter, we summarize and discuss the thesis and its contributions. Beyond that, we investigate possible research items, which extend or refine the results and methods presented in this thesis.

8.1 Summary

End-users expect services that are accessible at any time independent of their location. These services usually rely on services of other service providers and also on a variety of systems developed by different vendors. These services and systems constitute complex distributed systems and need to be able to interoperate.

In this thesis, we presented the IAI process, which is applicable to improve and to assess interoperability of homogeneous and heterogeneous distributed systems. The IAI process comprises three phases: investigation of the fulfillment of prerequisites for interoperability (Phase I), improvement of interoperability (Phase II), and assessment of interoperability (Phase III). For Phase III, we presented a methodology for automated interoperability testing. The methodology comprises four main parts: 1) We defined a generic environment for interoperability tests with message checks. This environment is the basis for automated interoperability tests. 2) We determined guidelines for interoperability test design and test automation. Both should be considered when specifying automated interoperability tests. 3) We developed a generic library for automated interoperability tests using TTCN-3 that implements the generic environment as well as the guidelines. 4) We created the DAITS process, which is a generic development process for a systematic specification of a complete and structured automated interoperability test system with message checks. The methodology for automated interoperability testing provides a first step towards a formalized and structured interoperability assessment of systems in an automated manner. An interoperability test system that is developed by applying our methodology lowers the costs for executing interoperability tests, because less human resources are required for their execution and evaluation.

We applied three case studies that instantiate the IAI process. In our first case study, we applied the IAI process successfully to IMS networks, which are implemented based on stable IMS standards by different vendors. The IMS standards specify open interfaces for interoperation. For the third phase of the IAI process, we applied the methodology for automated interoperability testing using TTCN-3 successfully to develop interoperability

tests with message checks for IMS network interoperability. The tests were successfully executed in the third ETSI IMS Plugtests™.

To evaluate the applicability of the IAI process for heterogeneous systems, we utilized the IAI process to assess and to improve the interoperability of grid and cloud systems. In our second case study, we focused on interoperability between grid and IaaS cloud systems. After the analysis and the comparison of both systems, we concluded that both can be integrated based on the communication protocol provided by the grid system through instantiating a grid system within an IaaS cloud system based on the interoperability gateway approach as part of Phase II of the IAI process. We integrated the AWS cloud with the UNICORE grid system and the Eucalyptus cloud with the GT4 system. In addition, we analyzed the ETSI GCM standards, which provide abstract descriptions of interfaces for the allocation of resources and for application deployment in different computing and storage systems. However, these interfaces still need an interoperability gateway to be able to communicate with the target system. We assessed the GCM for interoperability in Phase III of the IAI process. However, because of the different involved proprietary interfaces, we were only able to perform the interoperability tests manually based on test descriptions. Due to the proprietary interfaces and the high abstraction of the GCM standards, it was only partly possible to show interoperability. Therefore, another goal of the assessment was to identify commonalities of the different interface implementation of the target systems.

In our third case study, we applied the IAI process to a grid system and a PaaS cloud. In Phase I of the IAI process, we identified a common use case scenario applicable if both systems are able to interoperate. The responsibility of the PaaS cloud system is to scale with the number of requests and users while the grid environment executes computationally intensive tasks. In addition, the PaaS cloud provides means to publish the results of the computationally intensive tasks to cloud users. For Phase II, we selected the grid system UNICORE and the PaaS cloud GAE, for which we implemented an interoperability gateway. The cloud application that we developed with the GAE platform successfully submitted a computationally intensive task to the UNICORE system via the interoperability gateway. After termination of the task execution, the results were written to the GAE cloud storage. The interoperability gateway is implemented as a Web service that communicates via SOAP messages over HTTP. The structure of SOAP and HTTP is standardized and provided the basis for the assessment of the interoperability gateway with our methodology for automated interoperability testing. We developed an interoperability test system to assess GAE and UNICORE interoperability in an automated manner. Since their communication is based on HTTP, the test system can be reused for interoperability solutions based on Web services.

8.2 Discussion

While the case studies described in Chapters 5, 6, and 7 proved the applicability and the suitability of the IAI process as well as of the methodology for automated interoperability testing, they also raised a number of questions regarding the effort to implement interoperability solutions and interoperability test systems, but also regarding the extent to which the developed solutions are reusable. In the following, we discuss the results of the case studies regarding the application of the IAI process, the interoperability improvement between grids and clouds, and the automation of interoperability test execution.

8.2.1 Application of the IAI process

In our first case study, we applied the IAI process for interoperability assessment and improvement to homogeneous systems, i.e., IMS networks. Even though IMS networks implement common standards, a relatively high effort is required for analyzing the standards to identify common and complementary functionalities. However, compared to our second and third case study, where we applied the IAI process to grid and cloud systems, i.e., heterogeneous systems, the effort for performing Phase I of the IAI process was still low. The application of the IAI process was challenging for grids, IaaS clouds, and PaaS clouds, due to their implementation of proprietary interfaces. Even though standards for grids and IaaS clouds exist, they are not commonly adopted. In PaaS clouds only one standard initiative (CAMP) exists, which was released only recently.

In IMS as well as in the cloud domain, a variety of service providers exist. The difference is that IMS is clearly defined through standards. In the cloud domain, different vendors might have a different view on a cloud than other vendors. As a result, only broad definitions about clouds exist, which makes it difficult to identify commonalities and complementations between the clouds themselves as well as between grids and clouds. For grid and cloud interoperability, one needs to understand the communication protocols, interfaces, and architectures of both domains. After successful identification of common and complementary functionalities, their communication protocols and data formats need to be matched against each other. In addition, the common and complementary functionalities usually rely on other functionalities such as authentication and accounting in order to be interoperable. The interoperability of these dependent functionalities needs to be taken into account as well, e.g., with an interoperability gateway that maps user credentials between systems.

A limitation of the IAI process is that it does not take semantical and organizational interoperability into account. The reason is that both terms are only vaguely described in the literature [85]. In addition, if the meaning of information is preserved (semantical interoperability) and if business processes are linked together in a meaningful way (organizational interoperability) is difficult to assess. The technical and syntactical interoperability that we consider with the IAI process are both necessary conditions for semantical and organizational interoperability.

8.2.2 Interoperability of Grid Systems and Clouds

In two case studies, we analyzed and implemented interoperability solutions for grids with IaaS and PaaS clouds. For grid-IaaS cloud interoperability, we deployed the grid core services and the local resources management software stack in an IaaS cloud, which then becomes a PaaS cloud that offers computing services as well as an API provided by the grid middleware. This is especially interesting in application domains, where computing power is needed spontaneously and in unpredictable time intervals. It is also a step towards the migration of grid applications into IaaS cloud environments. Another benefit of achieving interoperability between grids and clouds is that existing grid systems are leveraged to secure previous and valuable investments of the development of well-engineered grid systems and applications.

We integrated the AWS cloud with the UNICORE grid system and the Eucalyptus cloud with the GT4 system. The integration effort for both grids with the respective cloud is similar, but high. The reason for the high effort is that the integration resulted in the management of two kinds of resources: the resources within the grid and within the IaaS cloud. We integrated the management of the IaaS cloud within the grid client. We developed a default configuration for the IaaS cloud that was instantiated automatically within the IaaS cloud. The configuration included virtual networks, instance selection, credentials, and user setup.

The IaaS cloud instantiation is based on a specific build of a virtual machine image that is configured with grid software. This virtual machine image was then instantiated in the IaaS cloud by the grid client. Ideally, during the instantiation, all configurations are triggered automatically, which we achieved with bash scripts. This allows the automatic connection of the grid-in-cloud services to the grid gateway of the existing grid. The effort for the building of the virtual machine images was high, since all required parts within the grid system needed to be configured and compiled into a virtual machine image. The configuration of a grid system has many complex constraints and includes the setup of users and their profiles, service and host certificates, and of specific grid containers and services. In addition, to change the grid configuration, we needed to build the virtual machine image again, which is very time intensive. GT4 was harder to configure than UNICORE. UNICORE provides a full installation and configuration documentation while GT4 was configured by trial and error. After the virtual machine images for both systems are configured and created, they can be reused in the same cloud, but not in different clouds. If the cloud provider decides to change the format and type of the virtual machine image, the virtual machine images need to be created with high effort again. This, as well as potential vendor lock-in, limits our approaches of integrating grid and IaaS cloud systems. The alternative is to deploy an own private cloud system, e.g., by installing Eucalyptus. However, the configuration and maintenance of an own private cloud is complex and time consuming as well. The issue of changing interfaces applies here as well as in public clouds, since the software is continuously developed and subject to potential interface changes as well.

In the third case study, we developed an interoperability solution for grid and PaaS systems based on UNICORE and GAE. We determined a specific and useful interoperability scenario for grid systems and PaaS clouds. Computational jobs are swapped out from the PaaS cloud into the grid system. As a result, the PaaS cloud provides a user friendly interface within SaaS, which hides the complexities of grid computing systems. The effort to implement the GAE-UNICORE gateway was very high. The infrastructure for the interoperability gateway needed to be deployed and configured. In addition, the interoperability gateway needed to be updated and adapted to changes of the interfaces of both systems, which makes this solution hard to maintain. Ideally, the PaaS cloud provider deploys the interfaces to the grid services natively into the PaaS cloud environment. This allows the PaaS cloud service to use grid services via a possibly standardized API. A limitation of our approach is that we only analyzed GAE. It needs to be further analyzed if the restrictions that the GAE PaaS cloud enforces are the same for other providers. In this case, our interoperability solution can be extended.

The presented grid-PaaS cloud usage scenario is limited and competes with similar cloud services. Cloud providers already offer virtualized HPC environments in the cloud. However, this would raise the cost for utilizing cloud resources considerably, since in case of heavy usage, it is cheaper to use grid or cluster resources as described by Carlyle et al. [16].

We identified the following common limitations of the grid-IaaS and the grid-PaaS interoperability solutions:

- **Portability**: If grid resources are connected with cloud resources, the portability of the application in the PaaS cloud or the virtual machine image in the IaaS cloud cannot be assured. One of the reasons is that grids and clouds are implemented and provided by different stakeholders usually implementing proprietary and often competing interfaces and data formats. The responsibility and the effort to change the cloud provider is on the side of the user. For porting the application from one PaaS cloud to another PaaS cloud, the user would need to implement the application again by utilizing the specific PaaS API of the new provider. However, if it comes to grid-cloud interoperability, it is unfeasible to change the grid system or the cloud provider. The main reason is that high effort needs to be invested to adapt the same solution for the new provider or system. However, if the cloud providers would adopt the OCCI, CDMI, or CAMP standards, the user would be able to port the application or virtual machine image with low effort to other cloud providers.
- **Reusability**: Resulting from the limited portability, interoperability solutions are also limited in their reusability. An interoperability solution can only be reused between different providers, if they implement the same standards. But, our solutions are based mainly on custom-made interfaces providing a feasibility study on their interoperability.
- **Performance**: We did not measure the performance of the grid-cloud interoperability solutions. Our approach lacks methods to assess the impact on the overall system, e.g., the determination of the latency when utilizing grid and cloud systems in an interoperable way.

- **Stress tests**: We did not apply stress tests on our solutions. However, since clouds scale up by definition, the bottleneck would be exposed by the interoperability gateway.
- **Costs**: The use of on-demand cloud resources is relatively expensive when using the resources in a 24/7 manner. The extension of a grid system by an IaaS cloud should, therefore, only be used in peak times with a limited duration. In contrast, the grid-PaaS cloud interoperability, can be used to submit jobs in the existing grid, which would save costly on-demand resources in the PaaS cloud.
- **Reliability**: Clouds do not provide a 100% reliability. Therefore, the interoperability solution can be extended with specific reliability measures so that the overall reliability is improved. For example, data should be stored on geographically distributed sites of the cloud system. Our current approaches do not consider reliability.
- **Data security**: Sensitive experiment data should not be sent into the cloud. Depending on the regulations and the openness of the cloud provider, the data might be read by a third party or never be deleted physically. Our interoperability gateways could take care of specific data security measures.
- **Deprecated API methods**: A common issue in the development of the interoperability solutions is that parts of the cloud API can become deprecated in subsequent versions. This happened frequently during the development of our interoperability solutions, since a cloud was not defined uniquely. This makes it hard to develop an up-to-date application for PaaS cloud and to implement an accurate cloud extension for a grid client when using IaaS clouds. However, the cloud definition became clearer recently [92]. Therefore, we expect that cloud interfaces become stable.

8.2.3 Interoperability Test Automation

We applied the methodology for automated interoperability testing in two case studies within Phase III of the IAI process. For the IMS interoperability case study, the tests were successfully executed in the third ETSI IMS Plugtests™. In contrast to the previous ETSI IMS Plugtests [26, 27], we were able to assess the conformance to the IMS standards during testing for interoperability automatically. In the first and second ETSI IMS Plugtests, the interoperability tests were driven manually and the conformance to the standards was assessed manually for each test case. The latter means that the traces of the communication between the EUTs was recorded during the executions of an interoperability test and analyzed after termination in a manual manner. The time required for manual conformance analysis is in the order of minutes and it can be performed only by an expert in the involved protocol. If the assessment of the conformance is done in an automated manner using our methodology, the time required for the automated analysis for one test case is in the order of milliseconds. Only if a conformance test fails, a manual analysis is required. The results of the automated assessment simplify the analysis. However, the test specification for auto-

mated interoperability tests with message checks needs to be developed, which can only be done by experts. The time required for such a development is rather high. But since the test specification is reusable, the development effort is amortized within several executions.

The reusability of an automated interoperability test system for systems that are not based on the same standards but providing common and complementary functionalities is limited depending on the interoperability solution, involved protocols, and data formats implemented by the systems. The test system usually needs to be updated to allow message checks and to trigger EUTs and UEs. We implemented an automated interoperability test system to assess the interoperability between grids and PaaS clouds exemplified by UNICORE and GAE. The developed test system can be reused for interoperability solutions based on Web services. The interoperability gateway needs to be updated regarding the utilized grid system and PaaS cloud to account for the standards and customized interfaces of the systems under consideration. In both case studies, we reused our generic LibIot library.

A main issue is the handling of proprietary interfaces, which is also related to the limitations of test automation. Limitations related to the automation of executing interoperability tests are discussed in detail in Section 4.2.2.

An interoperability test system that is developed by applying our methodology lowers the costs for executing interoperability tests, because less human resources are required for their execution and evaluation. More interoperability tests can be applied in interoperability test events, because an automatically executed test finishes faster than a manually executed test. This allows a more thorough testing. Furthermore, the scope of test scenarios can be extended, e.g., by the application of load tests. Compared to manual testing, the benefits of test automation are wider test coverage, consistency, and repeatability.

8.3 Outlook

Future research directions of the presented work are manifold and can be related to the following three topics: extension of the IAI process, extension of the methodology for automated interoperability testing, and the integration of grid and cloud systems.

Extension of the IAI process

The IAI process can be extended in the following directions. First of all, it needs to be investigated whether Phase I and Phase II can be automated to reduce the required effort for their application. The basis for the automation could be a formal specification of the interoperating systems. For Phase I, formal specifications can be used to map common and complementary functionalities of different systems, which would decrease the effort to apply Phase I. A prerequisite would be the use of the same or compatible formal specification techniques for all involved systems. For Phase II, an interoperability gateway may be generated automatically from the formal specifications and their mappings.

It also needs to be investigated to which degree the IAI process can be extended to semantical and organizational interoperability. Each phase of the process needs to be adapted and then applied to a specific level of interoperability. If the process is extended to organizational interoperability, business processes need to interoperate over organizational boundaries, which is currently not considered by the IAI process. Technical and syntactical interoperability is not sufficient if business processes require to interoperate over organizational boundaries. A challenge is that vendors with the market dominance pursue the objective of sustaining the dominance by rejecting measures to enable interoperability with their system. In such an environment, organizational interoperability is hard to achieve.

Additionally, we only show the substantiation of the IAI process for IMS, grid systems, and cloud systems. Further case studies need to be performed on additional systems to gain further evidence of the general applicability of the IAI process. Such case studies may include the application of the IAI process to IPv6 systems, smart grids, public services of eGovernment, medical devices, and emergency devices for public safety.

Extension of the Methodology for Automated Interoperability Testing

Regarding the methodology for automated interoperability testing, further possibilities for the automation of interoperability test executions need to be investigated. This includes the improvement of accessing proprietary interfaces and the reduction of manual interactions. If the interoperating systems are modeled using a formal specification technique, it might be possible to generate components that handle the interfaces during test execution in an abstract way. As a result, the components for handling equipment operations do not need to be implemented manually.

Another research direction is the derivation of automated interoperability tests from formal specifications. Ideally, the abstraction level of the formal specification is chosen appropriately to enable the generation of an executable test suite. The effort would then have to be invested into creating the formal specification of the systems and its transformation to an interoperability test system. This can speed up the creation of the interoperability test system.

The methodology for automated interoperability testing can be further extended to test the interoperability of *User Equipment*s (UEs). Thus, the UEs become a part of the SUT. However, the automated handling of the UE's interfaces is challenging due to differing implementations of user interfaces.

In addition, the methodology may be extended with support for testing non-functional properties. This will help to understand performance relations to interoperability. For testing the reliability of EUTs, automatic workloads can be generated and injected as background load into the SUT by the test system while the interoperability test is executed.

We implemented the test systems in TTCN-3. It needs to be investigated if other test languages are also suitable for the implementation of our methodology. This includes studies whether these languages are able to cope with complex and distributed interoperability test setups.

Further Investigations for the Integration of Grid and Cloud Systems

The interoperation of grid and cloud systems may impact non-functional properties such as security, performance, reliability, or usability. Investigations on these properties is an interesting research direction, which will help to determine the real benefit of an interoperability solution. For example, for the integration of a grid system with an IaaS cloud, a performance analysis can measure the impact on the integrated systems and determine whether the time to complete a task is affected. In addition, the effects of indirections that result from the grid-cloud integration need to be explored further.

According to our results from the grid-cloud integration, new standards need to be developed to support their interoperation. More formal specifications regarding interoperability interfaces between grid and cloud systems will enable common means for accessing different heterogeneous systems.

Bibliography

[1] 3GPP. TS 23.228 IP Multimedia Subsystem (IMS); Stage 2. 3rd Generation Partnership Project (3GPP). [Online; `http://www.3gpp.org/ftp/Specs/html-info/23228.htm` fetched on 12/04/2012].

[2] Adaptive Computing. TORQUE Resource Manager. [Online; `http://www.adaptivecomputing.com/products/torque.php` fetched on 12/04/2012].

[3] Amazon Web Services LLC. Amazon Web Services. [Online; `http://aws.amazon.com` fetched on 12/04/2012].

[4] Apache. Deltacloud. [Online; `http://deltacloud.apache.org/` fetched on 12/04/2012].

[5] Apache. Libcloud. [Online; `http://libcloud.apache.org/` fetched on 12/04/2012].

[6] Argonne National Laboratory. OpenPBS Public Home. [Online; `http://www.mcs.anl.gov/research/projects/openpbs/` fetched on 12/04/2012].

[7] T. Banks. Web Services Resource Framework (WSRF) – Primer v1.2. Organization for the Advancement of Structured Information Standards (OASIS), 2006.

[8] O. Bergengruen, F. Fischer, T. Namli, T. Rings, S. Schulz, L. Serazio, and T. Vassiliou-Gioles. Ensuring Interoperability with Automated Interoperability Testing. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[9] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow. Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability. In *Proceedings of the 4th International Conference on Internet and Web Applications and Services (ICIW)*, pages 328–336. IEEE, 2009.

[10] C. Besse, A. R. Cavalli, M. Kim, and F. Zaïdi. Automated generation of interoperability tests. In *Proceedings of the IFIP 14th International Conference on Testing Communicating Systems (TestCom)*, page 169. Kluwer, B.V., 2002.

[11] J. Bhuta and B. Boehm. A Framework for Identification and Resolution of Interoperability Mismatches in COTS-Based Systems. In *Proceedings of the 2nd International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS)*. IEEE, 2007.

[12] M. Bormann, D. Wermser, and R. Patz. Conformance Testing of Complex Services Exemplified with the IMS' Presence Service. In *Proceedings of the 3rd International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST)*, pages 21–26. IEEE, 2009.

[13] M. D. Bunn, G. T. Savage, and B. B. Holloway. Stakeholder analysis for multi-sector innovations. *Journal of Business & Industrial Marketing*, 17(2/3):181–203, 2002.

[14] Capgemini, Sogeti, HP. World Quality Report 2011-2012. [Online; `http://www.de.capgemini.com/insights/publikationen/world-quality-report-2011-2012/?d=699E6C62-9C61-C40C-520B-B8D5B1EDA0B6` fetched on 12/04/2012].

[15] M. Carlson, M. Chapman, A. Heneveld, S. Hinkelman, D. Johnston-Watt, A. Karmarkar, T. Kunze, A. Malhotra, J. Mischkinsky, A. Otto, V. Pandey, G. Pilz, Z. Song, and P. Yendluri. Cloud Application Management for Platforms (CAMP). CloudBees, Cloudsoft, Huawei, Oracle, Rackspace, Red Hat, Software AG, 2012. [Online; `http://cloudspecs.org/CAMP/CAMP_v1-0.pdf` fetched on 12/04/2012].

[16] A. Carlyle, S. Harrell, and P. Smith. Cost-Effective HPC: The Community or the Cloud? In *2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 169–176. IEEE, 2010.

[17] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. World Wide Web Consortium (W3C). [Online; `http://www.w3.org/TR/wsdl` fetched on 12/04/2012].

[18] CloudBees, Cloudsoft, Huawei, Oracle, Rackspace, Red Hat, Software AG. Cloud Application Management for Platforms (CAMP). [Online; `http://www.cloudspecs.org/paas/` fetched on 12/04/2012].

[19] A. Desmoulin and C. Viho. Automatic Interoperability Test Case Generation Based on Formal Definitions. In *Proceedings of the 12th International Conference on Formal Methods for Industrial Critical Systems (FMICS)*, pages 234–250. Springer, 2008.

[20] S. Dibuz and P. Kremer. Framework and Model for Automated Interoperability Test and Its Application to ROHC. In *Proceedings of the IFIP 15th International Conference on Testing Communicating Systems (TestCom)*, pages 243–257. Springer, 2003.

[21] DMTF. Cloud Management Standards. [Online; `http://www.dmtf.org/standards/cloud` fetched on 12/04/2012].

[22] E. Dustin, J. Rashka, and J. Paul. *Automated Software Testing - Introduction, Management and Performance*. Addison-Wesley, 1999.

[23] Eclipse Foundation. Eclipse Rich Client Platform. [Online; `http://www.eclipse.org/home/categories/rcp.php` fetched on 12/04/2012].

[24] Edinburgh Parallel Computing Centre (EPCC). OGSA-DAI. [Online; `http://www.epcc.ed.ac.uk/software-products/ogsa-dai` fetched on 12/04/2012].

[25] J. Ernits, M. Kaaramees, K. Raiend, and A. Kull. Requirements-driven model-based testing of the IP multimedia subsystem. In *Proceedings of the 11th International Biennial Baltic Electronics Conference (BEC)*, pages 203–206. IEEE, 2008.

[26] ETSI. 1st IMS Interoperability Event. [Online; `http://www.etsi.org/WebSite/OurServices/Plugtests/IMS2007.aspx` fetched on 12/04/2012].

[27] ETSI. 2nd IMS Plugtests^{TM}. [Online; `http://www.etsi.org/WebSite/OurServices/Plugtests/2008IMS2.aspx` fetched on 12/04/2012].

[28] ETSI. 3rd IMS Plugtests^{TM}. [Online; `http://www.etsi.com/WebSite/OurServices/plugtests/2009IMS3.aspx` fetched on 12/04/2012].

[29] ETSI. Grids, Clouds & Service Infrastructures: Plugtests^{TM} and Workshop. [Online; `http://www.etsi.com/plugtests/GRID09/GRID.htm` fetched on 12/04/2012].

[30] ETSI. Plugtests^{TM} Interop Events. [Online; `http://www.etsi.org/plugtests/` fetched on 12/04/2012].

[31] ETSI. ES 202 568: Methods for Testing and Specification (MTS);Internet Protocol Testing (IPT);Testing: Methodology and Framework. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2004.

[32] ETSI. 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.

[33] ETSI. TS 102 827: GRID;Grid Component Model (GCM);GCM Interoperability Deployment. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2008.

[34] ETSI. TR 102 766: GRID; ICT Grid Interoperability Testing Framework and survey of existing ICT Grid interoperability solutions. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2009.

[35] ETSI. EG 202 810: Methods for Testing and Specification (MTS);Automated Interoperability Testing;Methodology and Framework. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[36] ETSI. ES 202 237: Methods for Testing and Specification (MTS);Internet Protocol Testing (IPT);Generic approach to interoperability testing. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[37] ETSI. TR 102 788: Methods for Testing and Specification (MTS);Automated Interoperability Testing; Specific Architectures. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[38] ETSI. TR 102 789: Methods for Testing and Specification (MTS);Automated Interoperability Testing;Summary of ETSI experiences about using automated interoperability testing tools. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[39] ETSI. TS 102 811: GRID;Grid Component Model (GCM);Interoperability test specification. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[40] ETSI. TS 102 828: GRID;Grid Component Model (GCM);GCM Application Description. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[41] ETSI. TS 186 011-2 V2.3.1: IMS Network Testing (INT);IMS NNI Interoperability Test Specifications;Part 2: Test Description for IMS NNI Interoperability. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2010.

[42] ETSI. ES 201 873-10: Methods for Testing and Specification (MTS);The Testing and Test Control Notation version 3;Part 10: TTCN-3 Documentation Comment Specification. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2011.

[43] ETSI. ES 201 873: Methods for Testing and Specification (MTS);The Testing and Test Control Notation version 3. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2011.

[44] ETSI. TS 124 229: Digital cellular telecommunications system (Phase 2+);Universal Mobile Telecommunications System (UMTS);LTE;IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2011.

[45] Eucalyptus. Euca2ools User Guide. [Online; `http://open.eucalyptus.com/wiki/Euca2oolsGuide` fetched on 12/04/2012].

[46] Eucalyptus Systems. About Eucalyptus Systems. [Online; `http://www.eucalyptus.com/about` fetched on 12/04/2012].

[47] Eucalyptus Systems. Eucalyptus 1.6 Documentation – Administration Guide. [Online; `http://open.eucalyptus.com/wiki/EucalyptusAdministratorGuide_v1.6` fetched on 12/04/2012].

[48] European Commission. European Interoperability Framework (EIF) for European public services, Annex 2 to the Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of Regions 'Towards interoperability for European public services'. European Commission, Brussels, Belgium, 2010.

[49] J. Fang, S. Hu, and Y. Han. A service interoperability assessment model for service composition. In *Proceedings of the International Conference on Services Computing (SCC)*, pages 153–158. IEEE, 2004.

[50] FCC. 47 United State Codes (USC) § 153 - Definitions. Federal Communications Commission (FCC). [Online; `http://www.law.cornell.edu/uscode/text/47/153#43` fetched on 12/04/2012].

[51] L. Field and M. Schulz. Grid Interoperability: The Interoperations Cookbook. *Journal of Physics: Conference Series*, 119(1):012001. IOP Publishing, 2008.

[52] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Request for Comments 2616: Hypertext Transfer Protocol – HTTP/1.1. Internet Engineering Task Force (IETF), 1999.

[53] Forschungszentrum Jülich. Jülich Supercomputing Centre. [Online; `http://www2.fz-juelich.de/jsc/index.php?index=3` fetched on 12/04/2012].

[54] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA Basic Execution Service Version 1.0, GFD-R.108. Open Grid Forum, 2008.

[55] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, 2002.

[56] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The Open Grid Services Architecture, Version 1.5, GFD-I.080, 2006.

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

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

[59] 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 *Lecture Notes in Computer Science*. Springer, 2005.

[60] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Proceedings of the Grid Computing Environments Workshop*. IEEE, 2008.

[61] Fraunhofer FOKUS NGN. Welcome to Open IMS Core's Homepage. [Online; `http://www.openimscore.org/` fetched on 12/04/2012].

[62] gLite. [Online; `http://glite.web.cern.ch/glite/` fetched on 12/04/2012].

[63] Globus Alliance. [Online; `http://www.globus.org/alliance/` fetched on 12/04/2012].

[64] Globus Alliance. The WS-Resource Framework. [Online; `http://www.globus.org/wsrf/` fetched on 12/04/2012].

[65] Google. App Engine. [Online; `http://appengine.google.com/` fetched on 12/04/2012].

[66] Google. Google App Engine: Java Service APIs. [Online; `https://developers.google.com/appengine/docs/java/apis` fetched on 12/04/2012].

[67] Google. Google Apple Engine: The Java Servlet Environment. [Online; `https://developers.google.com/appengine/docs/java/runtime?hl=en` fetched on 12/04/2012].

[68] Google. What Is Google App Engine? [Online; `http://developers.google.com/appengine/docs/whatisgoogleappengine` fetched on 12/04/2012].

[69] D. Gruntz, S. Murer, and C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[70] Ibis Project. JavaGAT. [Online; `http://www.cs.vu.nl/ibis/javagat.html` fetched on 12/04/2012].

[71] IEEE. Standard Glossary of Software Engineering Terminology 610.12-1990. IEEE Press, 1999.

[72] IEEE Standards Association. P2301 - Guide for Cloud Portability and Interoperability Profiles (CPIP). [Online; `http://standards.ieee.org/develop/project/2301.html` fetched on 12/04/2012].

[73] ISO/IEC. Information Technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts. International ISO/IEC multipart standard No. 9646, 1991.

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

[75] ISTQB. Standard glossary of terms used in Software Testing. International Software Testing Qualifications Board (ISTQB), Glossary Working Party, 2010.

[76] ITU Telecommunication Standardization Bureau. Activities in Cloud Computing Standardization - Repository, 2010. [Online; `http://www.itu.int/dms_pub/itu-t/oth/49/01/T49010000020002PDFE.pdf` fetched on 12/04/2012].

[77] Ixia. IMS: Products. [Online; `http://www.ixiacom.com/solutions/ims/products/index.php` fetched on 12/04/2012].

[78] D. Jackson. Software Abstractions: Logic, Language, and Analysis – Appendix B: Alloy Language Reference. The MIT Press, 2006.

[79] K. Jeffery and B. Neidecker-Lutz, editors. *The Future Of Cloud Computing, Opportunities for European Cloud Computing Beyond 2010*. European Commission, 2010.

[80] S. Jha, A. Merzky, and G. Fox. Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes. *Concurrency and Computation: Practice & Experience*, 21(8):1087–1108. John Wiley & Sons, 2009.

[81] P. Johansson and H. Wallinder. A Test Tool Framework for an Integrated Test Environment in the Telecom Domain. D-level thesis, Karlstad University, 2005.

[82] Jülich Supercomputing Centre. UNICORE. [Online; `http://www.unicore.eu/` fetched on 12/04/2012].

[83] Jülich Supercomputing Centre. UNICORE Client Layer. [Online; `http://www.unicore.eu/unicore/architecture/client-layer.php` fetched on 12/04/2012].

[84] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software – Practice and Experience*, 32(2):135–164. Jobn Wiley & Sons, 2002.

[85] H. Kubicek and R. Cimander. Three dimensions of organizational interoperability. *The European Journal of ePractice*, 6:1–12. European Commission, 2009.

[86] E. Laure, S. M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. D. Meglio, and A. Edlund. Programming the Grid with gLite. In *Computational Methods in Science and Technology*, volume 12(1), pages 33–45. Scientific Publishers OWN, 2006.

[87] A. W. S. LLC. Products & Services. [Online; `http://aws.amazon.com/products/` fetched on 12/04/2012].

[88] J. Ludewig and H. Lichter. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag, 2007.

[89] W. Ma, L. Chung, and K. Cooper. Assessing Component's Behavioral Interoperability Concerning Goals. In *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*, volume 5333 of *Lecture Notes in Computer Science*, pages 452–462. Springer Berlin Heidelberg, 2008.

[90] M. E. Maarabani, A. Adala, I. Hwang, and A. Cavalli. Interoperability testing of presence service on IMS platform. In *Proceedings of the 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops (TridentCom)*, pages 1–6. IEEE, 2009.

[91] F. Magoulès, J. Pan, K.-A. Tan, and A. Kumar. *Introduction to Grid Computing*. CRC Press, Taylor & Francis Group, 2009.

[92] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Special Publication 800-145, National Institute of Standards and Technology (NIST), 2011. [Online; `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf` fetched on 12/04/2012].

[93] A. Merzky, K. Stamou, and S. Jha. Application Level Interoperability between Clouds and Grids. In *GPC Workshops*, pages 143–150. IEEE, 2009.

[94] Mircosoft. Windows HPC Server. [Online; `http://www.microsoft.com/hpc/en/us/product/cluster-computing.aspx` fetched on 12/04/2012].

[95] E. Mulyana, T. Juhana, D. Satriya, and D. Anggita. IP Multimedia Subsystem: A lab-scale test-bed: Implementation and case study for Click-to-Dial and VoD services. In *Proceedings of the 6th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, pages 181–185. IEEE, 2011.

[96] J. M. Myerson. Explore what customers should expect in cloudservice standards. IBM Corporation, developerWorks, 2012. [Online; `http://www.`

ibm.com/developerworks/cloud/library/cl-srvcstandardsgap/ fetched on 12/04/2012].

[97] H. Neukirchen. *Tools and Patterns for the Specification of Distributed Real-Time Tests*. PhD thesis, University of Göttingen, Germany, 2004.

[98] Niklas Brammer. Investigation of Automated Terminal Interoperability Test. Master's Thesis, Linköpings Universitet, Sweden, 2008.

[99] NTIA. FED-STD-1037C Telecommunications: Glossary of Telecomuunication Terms. National Telecommunications and Information Administration (NTIA), Institute of Telecommunications Sciences (ITS), Boulder, Colorado, USA, 1996. [Online; http://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm fetched on 12/04/2012].

[100] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *Proceedings of 9th IEEE International Symposium on Cluster Computing and the Grid*, pages 124–131. IEEE, 2009.

[101] OGF. OGSA Resource Usage Service WG (RUS-WG). [Online; http://www.ogf.org/gf/group_info/view.php?group=rus-wg fetched on 12/04/2012].

[102] OGF. Open Cloud Computing Interface Working Group. [Online; http://forge.ogf.org/sf/projects/occi-wg fetched on 12/04/2012].

[103] Open Cloud Consortium. Open Cloud Consortium. [Online; http://opencloudconsortium.org/ fetched on 12/04/2012].

[104] Open Grid Forum. Open Grid Forum. [Online; http://www.ogf.org/ fetched on 12/04/2012].

[105] Open Mobile Alliance. OMA Presence Simple V1.1. [Online; http://www.openmobilealliance.org/Technical/release_program/presence_simple_v1_1.aspx fetched on 12/04/2012].

[106] Open Mobile Alliance. OMA Test Events. [Online; http://www.openmobilealliance.org/TestFests/FutureEvents.aspx fetched on 12/04/2012].

[107] Open Science Grid. The Open Science Grid. [Online; http://www.opensciencegrid.org/ fetched on 12/04/2012].

[108] OpenID Foundation. OpenID. [Online; http://openid.net/ fetched on 12/04/2012].

[109] Oracle. Grid Engine. [Online; `http://www.oracle.com/technetwork/oem/grid-engine-166852.html` fetched on 12/04/2012].

[110] Organization for the Advancement of Structured Information Standards (OASIS. UDDI Version 3.0.2. [Online; `http://uddi.org/pubs/uddi_v3.htm` fetched on 12/04/2012].

[111] S. Ostermann, R. Prodan, and T. Fahringer. Extending Grids with Cloud Resource Management for Scientific Computing. In *Proceedings of the 10th International Conference on Grid Computing*. IEEE/ACM, 2009.

[112] Persistence of Vision Raytracer Pty. Ltd. Persistence of Vision Raytracer (POV-Ray). [Online; `http://www.povray.org/` fetched on 12/04/2012].

[113] M. Poikselka and G. Mayer. *The IMS: IP Multimedia Concepts and Services*. John Wiley & Sons, 2009.

[114] S. Pokraev, D. Quartel, M. W. A. Steen, and M. Reichert. Requirements and Method for Assessment of Service Interoperability. In *Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC)*, pages 1–14. Springer, 2006.

[115] D. Quartel and M. van Sinderen. On Interoperability and Conformance Assessment in Service Composition. In *Proceedings of the 11th International Enterprise Distributed Object Computing Conference (EDOC)*, page 229. IEEE, 2007.

[116] T. Rings and J. Grabowski. Pragmatic Integration of Cloud and Grid Computing Infrastructures. In *5th International Conference on Cloud Computing (CLOUD)*, pages 710–717. IEEE, 2012.

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

[118] T. Rings, G. Caryer, J. Gallop, J. Grabowski, T. Kovacikova, S. Schulz, and I. Stokes-Rees. Grid and Cloud Computing: Opportunities for Integration with the Next Generation Network. *Journal of Grid Computing: Special Issue on Grid Interoperability*, 7(3):375–393. Springer, 2009.

[119] T. Rings, J. Grabowski, and S. Schulz. A Testing Framework for Assessing Grid and Cloud Infrastructure Interoperability. *International Journal On Advances in Systems and Measurements*, 3(1&2):95–108. IARIA, 2011.

[120] T. Rings, J. Grabowski, and S. Schulz. On the Standardization of a Testing Framework for Application Deployment on Grid and Cloud Infrastructures. In *Proceedings of the 2nd International Conference on Advances in System Testing and Validation Lifecycle (VALID 2010)*, pages 99–107. IEEE, 2010.

[121] T. Rings, H. Neukirchen, and J. Grabowski. Testing Grid Application Workflows Using TTCN-3. In *Proceedings of the 1st International Conference on Software Testing Verification and Validation (ICST)*, pages 210–219. IEEE, 2008.

[122] T. Rings, P. Poglitsch, S. Schulz, L. Serazio, and T. Vassiliou-Gioles. A Generic Interoperability Testing Framework and a Systematic Development Process for Automated Interoperability Testing. *International Journal on Software Tools for Technology Transfer*. To appear, Springer, 2013.

[123] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Request for Comments 3261: SIP: Session Initiation Protocol. Internet Engineering Task Force (IETF), 2002.

[124] N. Sadashiv and D. Kumar. Cluster, Grid and Cloud Computing: A Detailed Comparison. In *Proceedings of the 6th International Conference on Computer Science & Education (ICCSE)*. IEEE, 2011.

[125] D. S. Sayogo, H. Jarman, A. Whitmore, G. K. Tayi, J. Zhang, J. Hrdinova, T. Pardo, D. F. Andersen, L. Luna-Reyes, X. Tan, and D. L. Andersen. A stakeholder analysis of interoperable data architecture: the case of I-Choose. In *Proceedings of the 13th Annual International Conference on Digital Government Research (dg.o)*, pages 145–154. ACM, 2012.

[126] 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 *Lecture Notes in Computer Science*. Springer, 2003.

[127] S. Schulz. Test suite development with TTCN-3 libraries. *International Journal on Software Tools for Technology Transfer*, 10(4):327–336. Springer, 2008.

[128] S. Seol, M. Kim, S. Kang, and J. Ryu. Fully automated interoperability test suite derivation for communication protocols. *Computer Networks*, 43(6):735–759. Elsevier, 2003.

[129] SNIA. Storage Network Industry Association, 2006. [Online; `http://www.snia.org/` fetched on 12/04/2012].

[130] B. Sotomayor. The Globus Toolkit 4 Programmer's Tutorial – 1.3. WSRF: The Web Services Resource Framework. University of Chicago, 2005. [Online; `http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s03.html` fetched on 12/04/2012].

[131] Sourceforge. Fura (grid middleware). [Online; `http://sourceforge.net/projects/fura/` fetched on 12/04/2012].

[132] A. Spillner, T. Linz, and H. Schaefer. *Software Testing Foundations*. Rocky Nook Inc., 2nd edition, 2007.

[133] W. Stallings. The session initiation protocol. *The Internet Protocol Journal*, 6(1):20–30. CISCO, 2003.

[134] T. Sterling. *Beowulf Cluster Computing with Windows*. The MIT Press, 2001.

[135] Storage Networking Industry Association. Cloud Data Management Interface. [Online; `http://www.snia.org/cdmi` fetched on 12/04/2012].

[136] A. Streit, P. Bala, A. Beck-Ratzka, K. Benedyczak, S. Bergmann, R. Breu, J. Daivandy, B. Demuth, A. Eifer, A. Giesler, B. Hagemeier, S. Holl, V. Huber, N. Lamla, D. Mallmann, A. Memon, M. Memon, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, T. Schlauch, A. Schreiber, T. Soddemann, and W. Ziegler. UNICORE 6 – Recent and Future Advancements. *Annals of Telecommunications*, 65:757–762. Springer, 2010.

[137] A. S. Tanenbaum and M. van Steen. *Distributed Systems Principles and Paradign*. Pearson Eduction, 2nd edition, 2007.

[138] J. Tang, C. Davids, and Y. Cheng. A study of an open source IP multimedia subsystem test bed. In *5th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, QShine '08, pages 45:1–45:7. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[139] TeraGrid Archives. [Online; `http://www.teragrid.org/` fetched on 12/04/2012].

[140] The ATM Forum Technical Committee. Introduction to ATM Forum Test Specifications. The ATM Forum, 1994.

[141] The Virtual Data Toolkit. [Online; `http://vdt.cs.wisc.edu/` fetched on 12/04/2012].

[142] UNICORE Team. UNICORE TSI: MANUAL, 2011. [Online; `http://unicore.eu/documentation/manuals/unicore6/files/tsi/tsi-manual.pdf` fetched on 12/04/2012].

[143] University of Chicago. Nimbus. [Online; `http://www.nimbusproject.org/` fetched on 12/04/2012].

[144] H. van der Veer and A. Wiles. Achieving Technical Interoperability - the ETSI Approach. White Paper, European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2008.

[145] D. E. Vega, I. Schieferdecker, and G. Din. Design of a Test Framework for Automated Interoperability Testing of Healthcare Information Systems. In *Proceedings of the 2nd International Conference on eHealth, Telemedicine, and Social Medicine*, pages 134–140. IEEE, 2010.

[146] C. Viho, S. Barbin, and L. Tanguy. Towards a Formal Framework for Interoperability Testing. In *Proceedings of the IFIP TC6/WG6.1 - 21st International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, pages 53–68. Kluwer, B.V., 2001.

[147] E. Wallmüller. *Software-Qualitätssicherung in der Praxis*. Hanser, 2001.

[148] T. Walter, I. Schieferdecker, and J. Grabowski. Test Architectures for Distributed Systems - State of the Art and Beyond. In *Proceedings of the IFIP TC6 11th International Workshop on Testing of Communicating Systems (IWTCS)*, pages 149–174. Kluwer, B.V., 1998.

[149] C. Willcock, T. Deiß, S. Tobies, S. Keil, F. Engler, and S. Schulz. *An Introduction to TTCN-3*. John Wiley & Sons, 2011.

[150] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Fifth Edition) – W3C Recommendation 26 November 2008. [Online; `http://www.w3.org/TR/REC-xml/` fetched on 12/04/2012].

[151] World Wide Web Consortium (W3C). Latest SOAP versions. [Online; `http://www.w3.org/TR/soap/` fetched on 12/04/2012].

[152] World Wide Web Consortium (W3C). QA Framework: Specification Guidelines – W3C Recommendation 17 August 2005. [Online; `http://www.w3.org/TR/2005/REC-qaframe-spec-20050817/#isoguide24` fetched on 12/04/2012].

[153] World Wide Web Consortium (W3C). Web Services Glossary – W3C Working Group Note 11 February 2004. [Online; `http://www.w3.org/TR/ws-gloss/` fetched on 12/04/2012].

[154] L. Yamini, G. LathaSelvi, and S. Mukherjee. Efficient metascheduling in a cloud extended grid environment. In *Proceedings of the International Conference on Recent Trends in Information Technology (ICRTIT)*. IEEE, 2011.

[155] L. Zha, W. Li, H. Yu, X. Xie, N. Xiao, and Z. Xu. System Software for China National Grid. In H. Jin, D. A. Reed, and W. Jiang, editors, *NPC*, volume 3779 of *Lecture Notes in Computer Science*, pages 14–21. Springer, 2005.

[156] S. Zhang, X. Chen, S. Zhang, and X. Huo. The Comparison Between Cloud Computing and Grid Computing. In *Proceedings of the International Conference on Computer Application and System Modeling (ICCASM)*. IEEE, 2010.

List of Acronyms

3G *3rd Generation Mobile Telecommunications*

3GPP *3rd Generation Partnership Project*

AD *Application Descriptor*

API *Application Programming Interface*

ASN.1 *Abstract Syntax Notation One*

ATS *Abstract Test Suite*

AWS *Amazon Web Services*

CAMP *Cloud Application Management for Platforms*

CD *Coding/Decoding*

CDMI *Cloud Data Management Interface*

CLI *Command Line Interface*

COTS *commercial-off-the-shelf*

CSCF *Call Session Control Function*

DAITS *Development of an Automated Interoperability Test System*

DD *Deployment Descriptor*

DMTF *Distributed Management Task Force, Inc.*

DRMAA *Distributed Resource Management Application API*

DNS *Domain Name System*

EC2 *Elastic Compute Cloud*

EPR *Endpoint Reference*

ETSI *European Telecommunications Standards Institute*

EUT *Equipment Under Test*

ETS *Executable Test Suite*

GAE *Google App Engine*

GCM *Grid Component Model*

GIN *Grid Interoperability Now*

gLite *lightweight middleware for grid computing*

GLUE *Grid Laboratory for a Uniform Environment*

GOS *Grid Operation System*

GT4 *Globus Toolkit 4*

GUI *Graphical User Interface*

HiLA *High Level Application Programming Interface*

HPC *High Performance Computing*

HSS *Home Subscriber Server*

HTTP *Hypertext Transfer Protocol*

I-CSCF *Interrogating-CSCF*

IaaS *Infrastructure as a Service*

IAI *Interoperability Assessment and Improvement*

IBCF *Interconnection Border Control Function*

ICS *Implementation Conformance Statement*

IDE *Integrated Development Environment*

IMS *IP Multimedia Subsystem*

IP *Internet Protocol*

ITU *International Telecommunication Union*

IUT *Implementation Under Test*

JavaGAT *Java Grid Application Toolkit*

JSDL *Job Submission Description Language*

LibIot *Interoperability Test Library*

LTE *Long Term Evolution*

MoC *Means of Communication*

MTC *Main Test Component*

NAT *Network Address Translation*

NNI *Network-to-Network Interface*

OCC *Open Cloud Consortium*

OCCI *Open Cloud Computing Interface*

OGSA *Open Grid Services Architecture*

OGSA-BES *OGSA-Basic Execution Service*

OGSA-DAI *OGSA-Data Access and Integration*

OGSA-RUS *OGSA-Resource Usage Service*

OGSA-UR *OGSA-Usage Records*

OGF *Open Grid Forum*

OMA *Open Mobile Alliance*

OSI *Open Systems Interconnection*

OSG *Open Science Grid*

OVF *Open Virtualization Format*

PaaS *Platform as a Service*

PBS *Portable Batch System*

PCO *Point of Control and Observation*

P-CSCF *Proxy-CSCF*

PIXIT *Protocol Implementation Extra Information for Testing*

PKI *Public-Key-Infrastructure*

POV-Ray *Persistence of Vision Raytracer*

PTC *Parallel Test Component*

QE *Qualified Equipment*

S3 *Simple Storage Service*

SaaS *Software as a Service*

SA *System Adapter*

SCP *Secure CoPy*

SDP *Session Description Protocol*

S-CSCF *Serving-CSCF*

SIP *Session Initiation Protocol*

SLA *Service Level Agreement*

SOA *Service Oriented Architecture*

SNIA *Storage Network Industry Association*

SQA *Software Quality Assurance*

SQS *Simple Queue Service*

SSH *Secure SHell*

SUT *System Under Test*

TC MTS *Technical Committee for Methods for Testing and Specification*

TCI *TTCN-3 Control Interface*

TRI *TTCN-3 Runtime Interface*

TSI *Test System Interface*

TSS *Test Suite Structure*

TTCN-3 *Testing and Test Control Notation Version 3*

UDDI *Universal Description, Discovery and Integration*

UMTS *Universal Mobile Telecommunications System*

UNICORE *Uniform Interface to Computing Resources*

UE *User Equipment*

USA *United States of America*

VDT *Virtual Data Toolkit*

VM *Virtual Machine*

VoIP *Voice over Internet Protocol*

VPN *Virtual Private Network*

W3C *World Wide Web Consortium*

WS-Resource *Web Service Resource*

WSDL *Web Services Description Language*

WSRF *Web Services Resource Framework*

XML *eXtensible Markup Language*

XNJS *UNICORE's internal execution management engine*

XUUDB *UNICORE User Database*

List of Figures

List of Listings

List of Tables