Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3280

Interoperability Standard for Remote Laboratory Systems

Andreas Gerhardt

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Math. Michael Reiter
begonnen am:	05. Dezember 2011
beendet am:	17. Juni 2012
CR-Klassifikation:	C.2.4, D.2, D.2.13, D.4.7, H.5.2

Abstract

Experimenters are the shock troops of science. - Max Planck [1]

Experimente dienen Wissenschaftlern zur Erforschung neuer Technologien und zur Verifikation erhobener Thesen. Sie bilden die Grundlage des Fortschritts und sind aus der Wissenschaft nicht wegzudenken. In einer globalisierten Welt arbeiten Wissenschaftler und Forscherteams an gemeinsamen Projekten und müssen in der Lage sein, Ressourcen zu teilen. Um dieses Ziel zu erreichen findet eine sukzessive Virtualisierung von Ressourcen statt. Diese Entwicklung beinhaltet auch die Bereitstellung von virtuellen Laboren, um Forschern einen entfernten Zugriff auf Experimente zu erlauben. Systeme zur Bereitstellung von Experimenten über virtuelle Labore werden Remote Laboratory Systems (RLS) genannt. Weltweit existieren Forschungsprojekte zur Entwicklung einer Infrastruktur für RLS. Um den Zugriff auf global verteilte Ressourcen zu ermöglichen, müssen die Systeme in der Lage sein zu interagieren.

Ziel dieser Diplomarbeit ist die Entwicklung eines Standards für das Global Online Laboratory Consortium (GOLC), welcher Services zur Kommunikation zwischen RLS definiert. Als Grundlage wird zunächst eine wissenschaftliche Einordnung von RLS diskutiert und ein Vergleich zu Grid-Computing gezogen. Es wird die wissenschaftliche Notwendigkeit betrachtet, sowie die Konzepte und die Architektur des Grid-Computing mit denen eines RLS verglichen. Anschließend werden die Konzepte und das Design des Standards erläutert, welcher auf einem Set an Profilen basiert. Es werden die Operationen der Profile definiert und deren Implementierung als SOAP-basierte Webservices beschrieben. Die Services ermöglichen es Nutzern anderer Systeme, alle relevanten Operationen durchzuführen um die Funktionalität eines RLS zu verwenden. Nutzer aus verschiedenen Domänen werden in der Lage sein, Experimente zu verwalten und auszuführen, Ressourcen zu buchen und wichtige Abfragen, z.B. über die angebotenen Dienste, zu tätigen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Verwendete Abkürzungen

GOLC - Global Online Laboratory Consortium

LiLa - Library of Labs, RLS Forschungsprojekt der Universität Stuttgart

MTOM - Messaging Transmission Optimization Mechanism

RLS - Remote Laboratory System

SOAP - Smart Object Access Protocol

WS - Webservice

WS-* - WS-* Spezifikationen

WSDL - Web Service Description Language

XML - Extensible Markup Language

XSD - XML Schema Definition

1. Einleitung

Dieses Kapitel stellt die Einleitung der Diplomarbeit dar. Zunächst wird der Hintergrund von Remote Laboratory Systems beschrieben und der wissenschaftliche Rahmen besprochen. Aus diesem Kontext heraus wird die Aufgabenstellung und somit das Ziel der Diplomarbeit spezifiziert, sowie eine Übersicht über Arbeiten genannt, die entweder für die Umsetzung der Aufgabenstellung von Bedeutung waren oder sich mit einer ähnlichen Thematik befassen. Abschluss der Einleitung bildet eine Übersicht über die Struktur der Arbeit und Beschreibungen der Kapitel.

1.1 Motivation

Weltweit existieren Forschungsprojekte zur Entwicklung einer Infrastruktur von Online-Laboren. Unter Verwendung dieser Infrastruktur soll es externen Nutzern und insbesondere Studenten ermöglicht werden, zeit- und ortsunabhängig auf entfernte Experimente und Labore zuzugreifen. Entfernte Labore werden in reale Labore und virtuelle Simulationen eingeteilt. Reale Labore stellen eine Hardwareinstallation bereit, auf der ein Nutzer ein definiertes Experiment durchführen kann. Die Steuerung und Programmierung der Maschinen, Roboter und Instrumente erfolgt ü ber das Internet, beispielsweise mittels Browser über Applets oder über separate Desktopanwendungen. Der Ablauf des Experimentes lässt sich oft per Kamera live beobachten. Die Ergebnisse und Messdaten, die während der Durchführung entstehen, werden dem Nutzer interaktiv präsentiert und können gespeichert werden. In virtuellen Laboren hingegen werden die Eigenschaften und Ergebnisse eines Experiments durch eine Simulation berechnet und auf dem Monitor des Nutzers dargestellt.

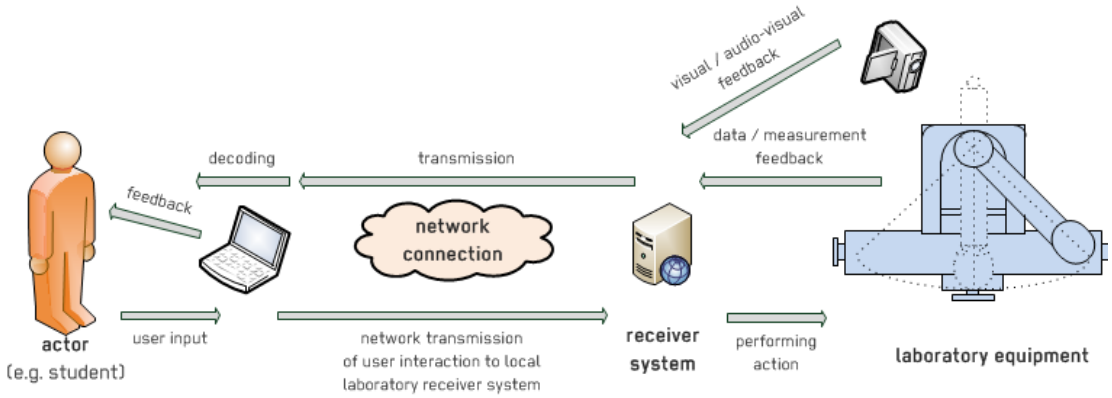


Abbildung 1 Virtuelles Labor. Entnommen aus [RL39]

Durch Online-Labore können Versuche, die aus Kosten-, Platz-, Zeit- oder Sicherheitsgrü nden bisher nicht möglich waren, fü r alle Interessenten zugänglich gemacht werden. So existieren Online-Labore für Experimente mit Radioaktivität oder Experimente, bei denen ein Roboter durch ein Labyrinth gesteuert werden kann.

Um den Zugriff auf weltweite Ressourcen und Experimente zu ermöglichen, müssen die Systeme der einzelnen Universitäten und Forschungszentren in der Lage sein zu interagieren. Langfristiges Ziel ist eine globale Zusammenarbeit der einzelnen Remote Laboratories, um den Zugriff auf weltweite Ressourcen und Experimente zu ermöglichen. Um dieses Ziel zu erreichen, mü ssen die unterschiedlichen Systeme in der Lage sein zu interagieren und gemeinsame Funktionen zu teilen. Bisher sind die Ressourcen eines Labors oder Experimentes nur innerhalb eines Verbundes buchbar, wie etwa dem Projekt Lila der Universität Stuttgart oder den iLabs des MIT [2]. Um Studenten und andere Nutzer in die Lage zu versetzen auch Ressourcen in einem anderen Verbund zu buchen, fehlt ein globaler Standard zur Kommunikation zwischen den einzelnen Systemen. Die Entwicklung eines solchen Standards ist Gegenstand der Diplomarbeit.

1.2 Aufgabenstellung

Im Rahmen der Diplomarbeit soll fü r das Global Online Laboratory Consortium ein Standard entwickelt werden, welcher Interaktionen zwischen bestehenden und zukünftigen Remote Laboratory Systems und das laborü bergreifende Buchen von Ressourcen definiert. Die zu entwickelnden Operationen sollen in Profile gruppiert werden und eine spätere Erweiterung leicht ermöglichen. Jedes dieser Profile soll eine bestimmte Funktionalität definieren, die sich an den Bedürfnissen eines Nutzers orientiert. Dazu zählen Abfragen zur Funktionalität selbst, die Verwaltung von Experimenten und Mechanismen zur Verwendung von Hardwareinstallationen. Im Rahmen der Operationen kann es notwendig sein, weitere Spezifikationen und Definitionen zum Nachrichtenaustausch festzulegen. Grundlage der Operationen bildet ein vorhandenes und standardisiertes Datenmodell, welches bereits von den beteiligten Systemen implementiert wird. Während der Spezifikation soll eine Komponente entstehen, die sich leicht in bereits vorhandene Architekturen einordnet. Diese Komponente soll auf Webservices basieren, wobei eine Vorabwahl des verwendeten Protokolls erfolgen soll.

Während der Entwicklung soll besondere Rü cksicht auf eine einfache Migration, die Einhaltung der Anforderungen und die Integration in bestehende Remote Laboratory Systeme genommen werden. Die Operationen müssen einen ausreichenden Datenschutz und Sicherheit beim Übertragen der Nachrichten gewährleisten. Die Implementierung der Webservices und Interfaces soll für das

System der Universität Stuttgart efolgen und mittels Java und frei wählbarer Werkzeuge erfolgen. Um Zugriff auf die Datenhaltung und das Buchungssystem zu erhalten, stehen die vorhandenen Komponenten zur Verfügung, welche ebenfalls in Java implementiert wurden. Die Implementierung der Services soll um einen Client ergänzt werden, welcher von anderen Systemen benutzt werden kann um die Services in Stuttgart zu nutzen.

1.3 Verwandte Arbeiten

Dieses Unterkapitel stellt Arbeiten vor, die entweder als Grundlage für die durchgeführte Diplomarbeit dienen oder sich mit der Thematik der Diplomarbeit befassen.

[3] behandelt das Konzept und die Architektur des Buchungssystems für das Projekt LiLa [4]. Die Arbeit beschreibt den Prozess einer Buchung und erläutert die beteiligten Elemente und Ressourcen. Im weiteren Verlauf wird das auf SCORM und LiLa Learning Objects (LLOs) basierende Konzept besprochen, sowie die Architektur und Implementierung beschrieben. Die in dieser Diplomarbeit entwickelten Services wurden für die Ausführung innerhalb des Projekts LiLa implementiert und bauen somit direkt auf dem Buchungssystem auf.

[5] definiert das Metadaten-Modell des Global Online Laboratory Consortiums. Die Arbeit definiert wichtige Begriffe, die für das Verständnis der Thematik und für das Verständnis von RLS, im Kontext des GOLC von Bedeutung sind. Es wird das Datenmodell von RLS festgelegt und die Entitäten beschrieben. Wichtige Bestandteile und Elemente des Daten-Modells werden in Aktivitäten eingebunden und so in Relation zu den Systemen gebracht. Sowohl das verwendete Buchungssystem, als auch die verwendete API für die LiLa Ontologie, basieren auf diesem Datenmodell. In Kapitel 5 dieser Diplomarbeit werden Änderungen am Datenmodell beschrieben, die zur Umsetzung der Services nötig sind.

[6] beschäftigt sich mit dem aktuellen Stand und der Evolution von RLS. Es werden Gründe beschrieben, wieso die aktuelle Entwicklung von RLS ins Stocken geraten ist und wie diese behoben werden können. Es werden vier Hauptgründe genannt, Wiederverwendbarkeit, Interoperabilität, Zusammenarbeit und die Annäherung an Learning Management Systeme. Das Paper geht näher auf die Interoperabilität ein und beschreibt die Zusammenführung von global verteilten Experimenten zu einem größeren Arbeitspaket. Um diese zu erreichen müssen Dienste entwickelt werden, welche diese Zusammenarbeit ermöglichen und eine Möglichkeit geboten werden, die Dienste zu finden.

[7] befasst sich mit den Anforderungen an einen Client zur Nutzung von Remote Laboratory Systems. Diese verwenden unterschiedliche Interfaces und können so keine Ergebnisse von Experimenten austauschen. Das Paper stellt einen möglichen

Ansatz zum Austausch von Ergebnissen vor. Dieser basiert auf XML und definiert ein Format zur Beschreibung von Ergebnissen. Ergänzend wird der Einsatz von Webservices im Kontext von RLS diskutiert und ein möglicher Nutzen zur Förderung der Interoperabilität beschrieben.

[8] beschreibt die historische Entwicklung von RLS und die dabei verwendeten, unterschiedlichen Forschungsansätze, welche zu inkompatiblen Systemen geführt hat. Als Resultat sind Interaktionen nur eingeschränkt möglich. Es wird an Hand von bestehenden RLS Architekturen versucht, einen Lösungsansatz zur Interoperabilität von RLS zu beschreiben und die dabei entstandenen Probleme aufzuzeigen. Der Lösungsansatz behandelt die Unterschiede der Architekturen, das Mappen gegebener Funktionalitäten und Datenmodelle, sowie die Entwicklung einer Zusammenarbeit unterschiedlicher Protokolle.

1.4 Aufbau des Dokuments

Kapitel 2 - Grundlagen

In diesem Kapitel werden Grundlagen behandelt, die für das Verständnis dieser Diplomarbeit von Bedeutung sind. Zunächst wird der Ablauf und Aufbau eines entfernten Experiments und eines Remote Laboratory Systems als Ganzes besprochen. Dies soll dem Leser als Grundlage für das Verständnis dienen. Ergänzend werden wichtige Begriffe im Kontext von RLS erläutert. Anschließend werden Technologien beschrieben, die für die Implementierung der Interfaces verwendet wurden. Dazu zählen die Serviceorientierte Architektur, Webservices und daraus resultierende Protokolle und Technologien, wie SOAP, WSDL und WS-Policies.

Kapitel 3 – Wissenschaftliche Einordnung von RLS

In diesem Kapitel findet eine wissenschaftliche Einordnung von RLS statt. Wir werden die wissenschaftliche Notwendigkeit diskutieren und die Architektur von RLS mit den Konzepten des Grid-Computing vergleichen. Als Beispiel hierfür dient die Architektur des RLS in Sydney und die Open Grid Service Architecture. Als Abschluss des Kapitels folgt eine Einordnung in wissenschaftliche Paradigmen unter Verwendung des Buches „The Fourth Paradigm“.

Kapitel 4 – Konzepte und Design

Das Kapitel beschreibt die grundlegenden Konzepte des entworfenen Standards. Es wird zunächst der Begriff Interoperabilität untersucht, welcher als Grundlage der erstellten Diplomarbeit einen hohen Grad an Bedeutung hat. Anschließend werden die funktionalen und nicht-funktionalen Anforderungen an die zu entwickelnde

Komponente definiert. Es werden die Bedürfnisse eines Nutzers an Hand von Anwendungsfällen beschrieben und die geplante Umsetzung diskutiert. Abschluss des Kapitels bildet eine Auswahl des Protokolls, durch das die Services implementiert werden sollen.

Kapitel 5 – Spezifikation der Profile

In diesem Kapitel werden die einzelnen Profile behandelt, welche Operationen zur Umsetzung der Anwendungsfälle definieren. Jede Operation wird beschrieben und die Ein- und Ausgabeparameter definiert. Für jedes Profil wird anschließend ein Szenario beschrieben, welches die Anwendung der Operationen verdeutlichen soll.

Kapitel 6 – Implementierung und Evaluation

In diesem Kapitel behandeln wir die Implementierung der Webservices, welche im Wesentlichen aus zwei Teilen besteht. Den eigentlichen Webservices auf der Serverseite und einem Client, welcher die Services aufruft und von anderen Systemen eingebunden werden kann. Zur Umsetzung der Operationen waren Erweiterungen am Datenmodell erforderlich. Diese werden ebenso beschrieben, wie die Schnittstellenbeschreibungen in WSDL. Abschluss des Kapitels bildet eine Beschreibung der Konfiguration der Services und eine Evaluation mittels SoapUI.

Als letzten Teil der Diplomarbeit fassen wir die erreichten Ziele zusammen und geben einen Ausblick auf zukünftige Möglichkeiten der globalen Zusammenarbeit von Remote Laboratory Systems.

2. Grundlagen

Dieses Kapitel soll dem Leser Grundlagen vermitteln, die zum Verständnis der Diplomarbeit erforderlich sind. Drei Bereiche sind hierzu von besonderer Bedeutung. Im ersten Unterkapitel werden zunächst wichtige Begriffe aus dem Kontext von Remote Laboratory Systemen erläutert und eine Erklärung zur grundlegenden Architektur von RLS gegeben. Unterkapitel 2.2 behandelt Grid Computing, welches als Basis für die wissenschaftliche Einordnung in Kapitel 3 von Bedeutung ist. Das letzte Unterkapitel behandelt Technologien, die für die Umsetzung des entwickelten Standards erforderlich waren.

2.1 Remote Laboratory Systems

Eine kurze Abhandlung zum Thema Remote Laboratory Systems befindet sich bereits in der Motivation in Kapitel 1. Dieses Unterkapitel soll die Terminologie erläutern um das Verständnis der Diplomarbeit zu erleichtern. Der zweite Teil widmet sich der grundlegenden Architektur von Remote Laboratory Systems, welche am Beispiel der Library of Labs der Universität Stuttgart erläutert wird.

2.1.1 Terminologie

Für das Verständnis von RLS im Kontext des GOLC und für das Verständnis der Diplomarbeit ist es wichtig, spezifische Begriffe zu definieren und abzugrenzen. Falls nicht anders ausgewiesen, entsprechen die hier vorgestellten Definitionen den Erläuterungen der LiLa Ontology [3] und des Metadaten-Profils des GOLC [5].

Remote Laboratory / System

Als Remote Laboratory wird eine Installation bezeichnet, bei der ein Nutzer, mittels einer Software, auf die Ressourcen einer physisch entfernten Hardwareinstallation oder Simulation zugreift, um ein Experiment durchzuführen. Ein System welches für die Verwaltung von Remote Laboratories zuständig ist, wird als Remote Laboratory System bezeichnet. In der Einleitung dieser Diplomarbeit wird auch der Begriff Online-Labor oder entferntes Labor als Synonym verwendet.

Experiment

Ein Experiment definiert den Prozess, bei dem ein Nutzer einen Client verwendet, um die Ressourcen eines Rigs oder einer Simulation für einen festgelegten Zeitrahmen zu kontrollieren. Dazu zählt das Konfigurieren der Ressourcen, die Ergebnisse der Berechnungen, die eingegebenen Daten und Daten die während des Vorgangs erzeugt

und gemessen werden. Sofern das Experiment nicht auf einer Simulation basiert, müssen die Ressourcen erst für einen definierten Zeitraum reserviert werden.

Interactive Experiment

Als interaktives Experiment wird ein Experiment bezeichnet, bei dem die Aktivitäten der Ressourcen in Echtzeit beobachtet und die Parameter und Einstellungen der verwendeten Hardware mittels eines User Interfaces, während der Ausführung modifiziert werden können. Im verwendeten Datenmodell speichert die Entität InteractionPackage alle nötigen Daten zur Durchführung eines interaktiven Experiments.

Batch Experiment

In Kontrast zu einem interaktiven Experiment finden bei einem Batch-Experiment keine Interaktionen mit dem Nutzer während der Ausführung statt. Batch-Experimente können komplexe Berechnungen sein, bei denen Rechenzeit auf einer bestimmten Hardware gemietet wird. Batch-Experimente werden vor der eigentlichen Ausführung konfiguriert und zur festgelegten Zeit selbstständig auf einem Rig ausgeführt. Resultate können entweder teilweise während der Ausführung oder vollständig nach der Ausführung abgefragt werden.

Rig / RigSet

Ein Rig bezeichnet einen Versuchsaufbau, auf dem ein definiertes Experiment ausgeführt werden kann, welches per Fernzugriff über ein Softwareinterface durch einen Nutzer gesteuert wird. Rigs können hierarchisch angeordnet sein, wobei mehrere Rigs mit gleichem Versuchsaufbau zu einem RigSet zusammengefasst werden. Im Datenmodell der RL-Systeme sind Rigs ohne Kinderknoten als einelementige RigSets definiert. Zur Vereinfachung werden im weiteren Verlauf nur Rigs mit einem oder mehreren Kinderknoten als RigSets bezeichnet.

Provider

Als Provider wird eine Person oder Organisation bezeichnet, welche einer definierten Zielgruppe Zugriff auf Remote Laboratories ermöglicht. Als Provider System wird das Computersystem und die Interfaces bezeichnet, mittels denen die angebotenen Ressourcen verwendet werden können.

Consumer

Ein Consumer ist definiert als eine Person oder Organisation, welche die von einem Provider angebotenen Dienste und Ressourcen nutzt. Das Computersystem, welches die Nutzung des Provider Systems ermöglicht, wird als Consumer System bezeichnet.

User

Der Begriff User beschreibt eine Person, welche über ein Consumer System auf die angebotenen Dienste und Ressourcen eines Providers zugreift. Im weiteren Verlauf der Diplomarbeit wird der Begriff Nutzer als Synonym für den Begriff User verwendet.

Booking

Ein Booking bezeichnet den Prozess der Reservierung von knappen Ressourcen, für einen limitierten Zeitraum und einen bestimmten Nutzer, oder eine begrenzte Gruppe an Nutzern. Eine Reservierung kann entweder automatisch durch das System des Providers freigegeben werden, oder durch die für das Rig zuständige Person oder Institution. In dieser Diplomarbeit werden auch Buchung oder Reservierung als Synonyme verwendet.

Booking System

Ein Datenbanksystem, als Bestandteil des Provider Systems, welches für die Verwaltung von Bookings zuständig ist. Consumer können über dieses System Reservierungen anfordern und abfragen, sowie die Verfügbarkeit eines spezifizierten Rigs für einen gegebenen Nutzer erfragen.

2.1.2 Systemübersicht

Ein Remote Laboratory System besteht im Wesentlichen aus drei Bereichen, den Hardwareinstallationen, dem Client des Nutzers und einer Middleware zur Koordination. Die Hardwareinstallationen stellen die eigentlichen Labore, also die Versuchsaufbauten der Experimente dar. Sie müssen über eine Schnittstelle verfügen, über die ein Nutzer die Hardware bedienen und Experimente steuern kann. Der Client stellt die Software dar, die ein Nutzer verwendet um sowohl auf die Middleware zuzugreifen, als auch die Hardwareinstallationen zu steuern. Dies ist in vielen Fällen ein Standardbrowser, es kann sich aber auch um eine separate Desktopanwendung handeln.

Die Middleware erfüllt mehrere Anforderungen, sowohl funktionale, als auch nicht-funktionale. Sie ist für die Verwaltung der Datenhaltung verantwortlich und koordiniert die Bedürfnisse der Nutzer und die angebotenen Ressourcen. Da die Ressourcen beschränkt sind, muss sich die Middleware um eine effiziente Verwaltung der Hardwareinstallationen bemühen und einem Nutzer ermöglichen, Labore für einen festgelegten Zeitraum zu reservieren. Diese Aufgabe übernimmt eine Komponente namens Buchungssystem, welche üblicherweise durch on-demand Dienste umgesetzt ist. Ergänzt werden kann die Middleware um ein Learning Management System, welches als Broker für Experimente angesehen werden kann.

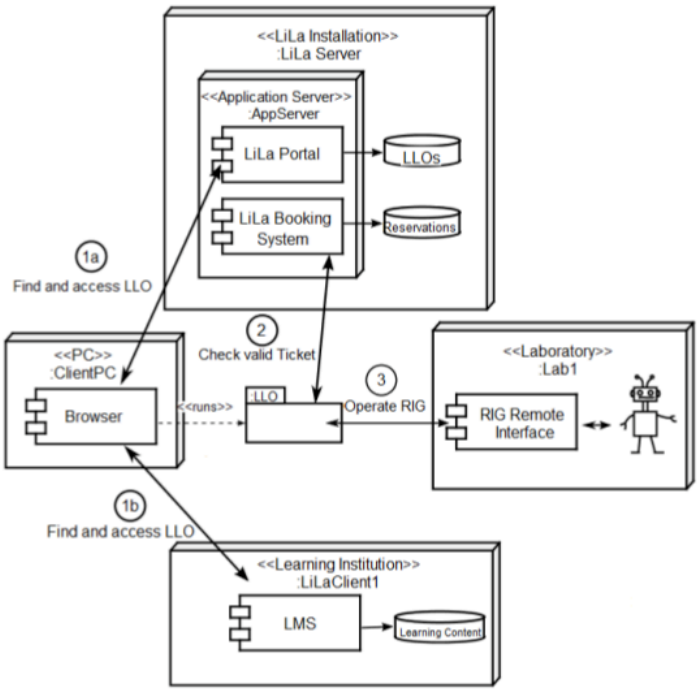


Abbildung 2 Architektur Library of Lalbs [3]

Ein Nutzer hat so die Möglichkeit, sich durch ein Portal auf Seite des Providers, oder durch das LMS über Experimente zu informieren und diese zu finden. Über das Buchungssystem lässt sich ein fester Zeitrahmen für die Durchführung eines Experiments reservieren. Der Provider stellt dem Nutzer daraufhin ein Objekt bereit, hier als Lila Learning Object bezeichnet, welches alle Informationen zur Durchführung des Experiments beinhaltet. Dieses Objekt beinhaltet Beschreibungen über das Experiment, Software zur Ausführung und Informationen zur Kommunikation mit der Hardwareinstallation, beispielsweise die Adresse der Interfaces der Hardwareinstallation.

2.2 Grid Computing

Grid Computing ist ein Konzept aus den 80er Jahren des vergangenen Jahrhunderts, bei dem verteilte Ressourcen Rechenleistung und Speicherkapazitäten von anderen Ressourcen nach Bedarf abrufen können [GR30]. Es findet eine Vernetzung von vorhandenen Infrastrukturen statt, um durch das Bündeln von Ressourcen eine höhere Effizienz zu erreichen. Die Koordination übernimmt dabei eine besondere Middleware.

Definition nach Foster [9]

The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization.

Foster beschreibt zwei wichtige Aspekte des Grid-Computing. Das Nutzen von gemeinsamen Ressourcen, welche meist geografisch verteilt sind und die Herkunft der Nutzer aus verschiedenen Domänen. Ressourcen sind dabei nicht als Dateien zu verstehen, die zwischen Organisationen und Systemen ausgetauscht werden, sondern als Computer, Software, Daten und Wissen, auf welche direkt zugegriffen werden kann. Nutzer werden als Personen oder Institutionen aus Industrie und Wissenschaft beschrieben, welche ein gemeinsames Ziel verfolgen oder ein gemeinsames Problem lösen wollen. Um dieses Ziel zu erreichen werden die Ressourcen der beteiligten Nutzer gemeinsam verwendet. Die Nutzer bilden somit eine virtuelle Organisation und legen fest, wer welche Ressourcen zu welchem Zeitpunkt und auf welche Art und Weise benutzen darf.

2.2.1 Architektur

Die abstrakte Architektur eines Grid wird als Sanduhr-Architektur bezeichnet und ist in Abbildung 3 dargestellt. Die Architektur ist in fünf Schichten eingeteilt, wobei Connectivity-, Resource und Collective-Layer die Middleware eines Grid und somit die Schnittstelle zwischen den Anwendungen im Application-Layer und den Ressourcen im Fabric-Layer definieren. Die Middleware vermittelt verfügbare Ressourcen und Arbeitsaufträge und erfüllt nichtfunktionale Anforderungen wie Sicherheit, Ressourcen- und Usermanagement. Höhere Schichten verwenden direkt die Protokolle und Schnittstellen tiefer gelegener Schichten. Somit steigt der Grad der Abstraktion der Funktionalität in höheren Schichten.

Die Darstellung der Architektur als Sanduhr nimmt direkt Bezug auf die Menge der Elemente einer Ebene. Die Zahl an möglichen Anwendungen im oberen Teil und möglicher Ressourcen im unteren Teil, welche durch die Mitglieder der virtuellen Organisation bereitgestellt werden, unterliegt theoretisch keiner Begrenzung. In der Resource-Layer-Ebene, in der Mitte der Sanduhr, werden sicherheitskritische Protokolle und Dienste verwendet, welche sich an einem Minimum an standardisierten Protokollen orientieren sollten.

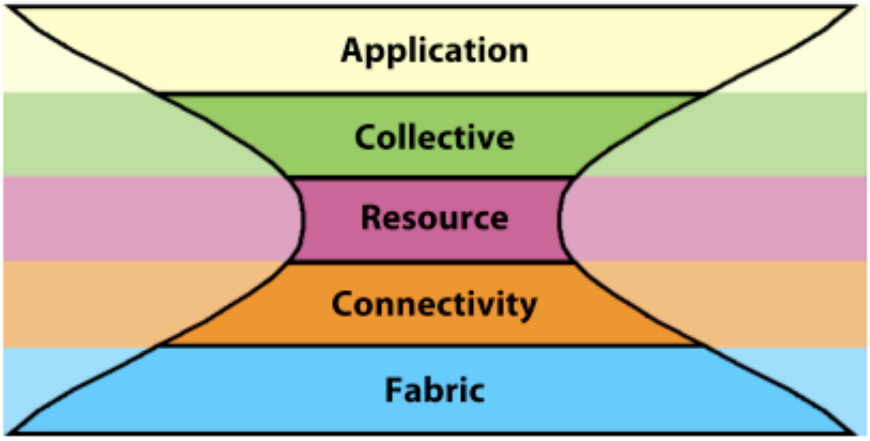


Abbildung 3 Sanduhr-Architektur [40]

Elemente einer Ebene. Die Zahl an möglichen Anwendungen im oberen Teil und möglicher Ressourcen im unteren Teil, welche durch die Mitglieder der virtuellen Organisation bereitgestellt werden, unterliegt theoretisch keiner Begrenzung. In der Resource-Layer-Ebene, in der Mitte der Sanduhr, werden sicherheitskritische Protokolle und Dienste verwendet, welche sich an einem Minimum an standardisierten Protokollen orientieren sollten.

Die unterste Ebene wird Fabric-Layer genannt und steht stellvertretend für die Ressourcen eines Grids, welche durch die Nutzer bereitgestellt werden. Als Ressourcen zählen Hard- und Software, Rechenleistung, Speicherplatz, Daten und Informationen, welche die Basis eines Grids bilden. Die nächste Ebene wird Connectivity-Layer genannt und bezeichnet Protokolle und Dienste zur Kommunikation und Authentifizierung von Grid-spezifischen Transaktionen in Netzwerken. Diese ermöglichen den Austausch von Daten zwischen Ressourcen des Fabric-Layers. Die dritte Ebene beinhaltet Protokolle um mit entfernten Ressourcen und Diensten zu interagieren und wird Resource-Layer genannt. Es wird zwischen zwei hauptsächlichen Klassen an Protokollen unterschieden. Die erste Klasse definiert Protokolle zur Gewinnung von Informationen über die Struktur und den Status einer Ressource. Die zweite Klasse beschreibt Protokolle um den Zugriff auf Ressourcen zu verwalten. Während die Resource-Layer-Ebene zur Verwaltung einzelner Ressourcen dient, befasst sich die Collective-Layer-Ebene, als vierte Ebene, mit der Koordination und Überwachung von Ressourcen-Kollektiven. Dabei werden direkt die Protokolle der Resource- und Connectivity-Layer-Ebenen benutzt. Des Weiteren müssen sich die Protokolle des Collective-Layers mit Sicherheitsaspekten, Richtlinien und Kontenführung befassen. Die oberste Schicht wird Application-Layer genannt und steht zusammenfassend für ausführbare Anwendungen im Grid.

2.3 Serviceorientierte Architektur

OASIS definiert eine Serviceorientierte Architektur (SOA) als Paradigma für Organisation und Nutzung verteilter Ressourcen, welche unter der Aufsicht verschiedener Besitzer sein können. [10] Eine SOA kapselt Funktionen und Komponenten in Dienste und unterstützt somit die Modularisierung von Systemen. Dienste können wiederum gruppiert und zu höheren Diensten zusammengefasst werden. Man spricht von einer Dienstekomposition, welche in zwei Arten unterscheiden werden kann. Bei einer Orchestrierung werden Dienste zusammengefasst, um eine definierte Aufgabe zu erfüllen und einen wieder verwendbaren Geschäftsprozess zu bilden. Die Dienste werden dabei von einer Kontrollinstanz gesteuert. Bei einer Choreographie findet die Koordination durch einen Nachrichtenaustausch zwischen den Diensten statt.

OASIS definiert drei wichtige Schlüsselkonzepte zur Beschreibung einer SOA, Sichtbarkeit, Interaktion und Auswirkung. Sichtbarkeit bedeutet, dass Nutzer und Anbieter von Diensten gegenseitig sichtbar sind und eine Möglichkeit zur Suche nach Diensten gegeben ist. SOA definiert zu diesem Zweck einen Verzeichnisdienst, Universal Description, Discovery and Integration (UDDI) [11], welcher ein Verzeichnis von Diensten verwaltet und Anbietern die Möglichkeit bietet, Dienste zu veröffentlichen. Ein Nutzer kann über diesen Verzeichnisdienst nach Diensten suchen und deren Beschreibungen erlangen. Die Beschreibungen beinhalten einen Verweis auf den angebotenen Dienst und werden in den folgenden Unterkapiteln näher behandelt. Sichtbarkeit bietet folglich die Möglichkeit, Bedarf und Funktionalität abzustimmen. Interaktion, als zweiter Schritt, beschreibt die Interaktion zwischen Konsument und Anbieter und somit die Aktivität der Nutzung eines Dienstes. Weiterer Bestandteil der Interaktion ist die Verhandlung über Verträge zur Nutzung der Dienste, welche in den Richtlinien einer Dienstbeschreibung enthalten sind. Das Ziel der Interaktion wird als Auswirkung bezeichnet. Konsumenten interagieren mit Anbietern um Ergebnisse zu erzielen. Dies kann das Gewinnen von Informationen oder die Änderung einer Entität in einer Datenbank sein. Die Auswirkungen bilden einen wichtigen Bestandteil der Zusammenführung von Bedürfnis eines Nutzers und Dienst eines Anbieters. In der Phase der Interaktion werden die Erwartungen an die Funktionalität ausgebaut. Eine Veranschaulichung der Zusammenarbeit zwischen Konsument, Anbieter und Verzeichnisdient ist in Abbildung 4 dargestellt.

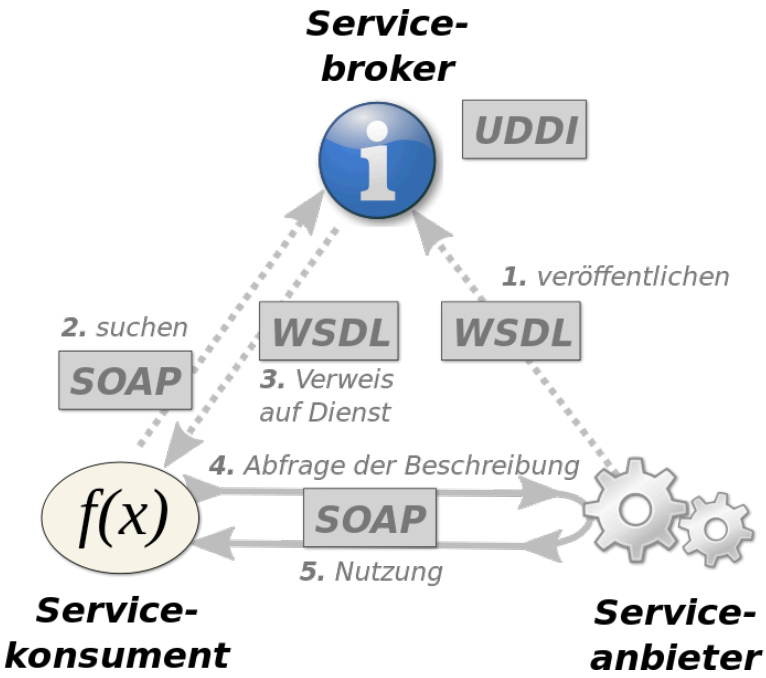


Abbildung 4 Zusammenarbeit in einer SOA [12]

2.3.1 Webservices

Die Dienste einer SOA werden häufig als Webservices implementiert. Als Webservice wird eine, in einem Netzwerk bereitgestellte, Komponente bezeichnet, die eine definierte Funktionalität als Teil einer Applikation erfüllt. Das World Wide Web Consortium definiert einen Webservice als ein Softwaresystem, welches die Interoperabilität einer Interaktion zwischen Maschinen über ein Netzwerk unterstützen soll [13].

Webservices fungieren als Schnittstelle zur Interaktion zwischen Systemen und definieren eine feste Funktionalität. Um eine größtmögliche Interoperabilität zu gewährleisten, basieren Webservices auf offenen und anerkannten Standards, wie etwa XML. Die Plattform eines Systems oder die eingesetzten Frameworks schränken die Nutzung von Webservices ebenso wenig ein, wie die verwendeten Protokolle. Oft verwendete Protokolle sind SOAP über HTTP, XML-RPC oder REST. Die Operationen und Parameter eines Webservices werden in einer Schnittstellenbeschreibung definiert, für die eine Standardsprache, die Web Services Description Language (WSDL), entwickelt wurde. WS-Policies beinhalten Verträge, welche bei der Nutzung eines Webservice eingehalten werden müssen.

2.3.2 WSDL

Die Web Service Description Language (WSDL) [WS65] ist eine standardisierte Beschreibungssprache für Webservices, basierend auf XML. WSDL definiert Dienste als Sammlungen von Netzwerkendpunkten oder Ports. Ein Port ist ein einzelner Endpunkt, bestehend aus einem Binding und einer Netzwerkadresse. Das Binding definiert ein konkretes Protokoll und die Datenformatierung für einen bestimmten Port Type. WSDL ist somit protokollunabhängig, wobei SOAP eines der häufigsten Protokolle zur Umsetzung ist. Der Port Type beinhaltet eine abstrakte Gruppe an Operationen, welche durch den Dienst unterstützt werden. Operationen bestehen abstrakten Ein- und Ausgabenachrichten. Die in Nachrichten verwendeten Parameter werden im Element Types einer Nachricht als XSD-Elemente definiert.

2.3.3 WS-* Spezifikationen

Die WS-* Spezifikationen [14] des W3C sind eine Sammlung an Spezifikationen im Kontext von Webservices unter Verwendung von SOAP und WSDL. Sie definieren Standards für Anwendungsgebiete, die nicht in SOAP oder WSDL enthalten sind und können unabhängig voneinander verwendet werden. WS-Policy[15] ist Bestandteil dieser Spezifikationen und stellt einen Mechanismus zur Beschreibung von nicht-funktionalen Anforderungen an Webservices, basierend auf XML dar. Sowohl Konsument, als auch Anbieter können Policies nutzen, um ihre Mindestrichtlinien zu spezifizieren. Dazu zählen Sicherheit, Qualität, aber auch die Version des Dienstes oder Details zur Nachrichtenübertragung. Konsument und Anbieter verhandeln vor der eigentlichen Nutzung des Dienstes über die spezifizierten Policies und einigen sich auf die höchstmöglichen Richtlinien.

Ebenfalls Bestandteil der Spezifikationen ist der WS-Security Standard [16], welcher sich mit der Sicherheit bei der Übertragung von Nachrichten befasst. Es lassen sich sowohl Teile einer SOAP-Nachricht in Header und Body, als auch komplette Nachrichten verschlüsseln und signieren. Zur Auswahl stehen dabei verschiedene Profile zur Erstellung der Sicherheitsmerkmale, wie etwa X.509-Zertifikate, die bei Client und Server hinterlegt werden oder SAML-Assertions. Als weiterer Mechanismus dient die Authentifizierung bei der Nutzung eines Dienstes über UsernameToken. Diese drei Hauptmechanismen decken die alle relevanten Sicherheitsaspekte ab. Es kann werden wer autorisiert wird, ob beim Routing Teile der Nachricht geändert wurden, ob die Nachricht von der angegebenen Quelle handelt und wie bestimmte Teile der Nachricht verborgen werden können.

2.3.4 SOAP

SOAP ist ein Netzwerkprotokoll, welches sich im TCP/IP-Referenzmodell in der Anwendungsschicht befindet. Das W3C definiert SOAP als ein leichtgewichtiges Protokoll zum Austauch von strukturierten Informationen in einer dezentralen,

verteilten Umgebung[13]. SOAP definiert Regeln für den Aufbau von Nachrichten, die Darstellung und Interpretation der Daten und spezifiziert eine Vorschrift für entfernte Prozeduraufrufe. Nachrichten werden über Internetprotokolle wie HTTP oder TCP gesendet und basieren auf XML zur Repräsentation der Daten. Eine Nachricht besteht aus einem Envelope, welches einen Header und einen Body umgibt und ist in Abbildung 5 dargestellt. Der Header beinhaltet optionale Metadaten, beispielsweise zur Verschlüsselung, zur Identifizierung der Transaktion oder andere, durch den Nutzer spezifizierte Informationen. Der Body enthält Informationseinheiten für den Empfänger der Nachricht. Dies können Anweisungen für den Aufruf einer entfernten Prozedur sein, Informationen zum Datenaustausch oder sonstige Parameter. Optionale können binäre oder nicht-binäre Dateien als Attachment an eine Nachricht angehängt werden.

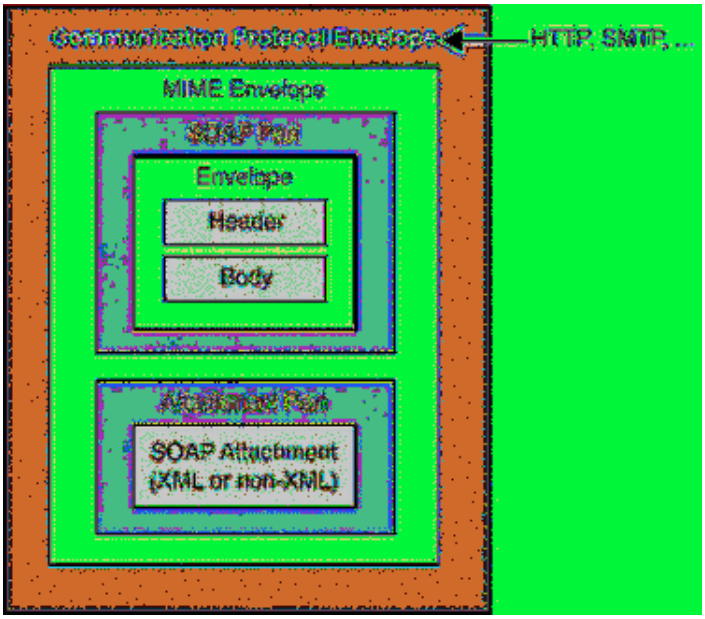


Abbildung 5 Aufbau einer SOAP-Nachricht [41]

3. Wissenschaftliche Einordnung von RLS

Dieses Kapitel beschäftigt sich mit der wissenschaftlichen Einordnung von RLS. Zunächst wird die wissenschaftliche und industrielle Notwendigkeit analysiert. Anschließend wird eine Einordnung in das Grid Computing unter Zuhilfenahme einer drei Punkte umfassenden Checkliste und Rahmenbedingungen des Grid Computing vorgenommen. Abschluss des Kapitels bildet eine Einordnung von RLS in wissenschaftliche Paradigmen nach dem Werk „The Fourth Paradigm“ [17].

3.1 Wissenschaftliche und industrielle Notwendigkeit

In [18] wird die Notwendigkeit einer Infrastruktur für Rechen- und Datenmanagement als Schlüssel zur heutigen Wissenschaft beschrieben. Die Wissenschaft produziert enorme Datenmengen, welche verarbeitet werden müssen. Der Large Hadron Collider des Cern zum Beispiel erzeugt jährlich eine Datenmenge von ungefähr 15 Petabyte [19]. Um solche Datenmengen auswerten zu können, wird eine erhebliche Rechenleistung benötigt. Um dieses Ziel zu erreichen ist nicht nur die Bildung von High-End Supercomputern erforderlich, sondern auch die Zusammenarbeit von bestehenden Infrastrukturen. RLS können an dieser Zusammenarbeit mitwirken, indem Installationen mit entsprechender Hardware zur Verfügung gestellt werden, auf denen Rechenzeit reserviert oder gemietet werden kann. Die Nutzer haben so die Möglichkeit, batchbasierte Experimente zur Auswertung von Daten von Berechnungen und Simulationen zu definieren und auszuführen.

Kapitel 2 des Buches beschreibt die Evolution einer Technologie, bestehend aus einer Entwicklungs- und einer Anwendungsphase. Zu Beginn steht die Technologie selbst im Vordergrund und wird durch die Akzeptanz der Beteiligten sukzessive durch die Post-Technologie abgelöst in welcher die Anwendung der Technologie das zentrale Element ist. Nutzer wollen ohne Kenntnisse der Technologie die Applikationen ausführen. Im Falle von RLS kann die Grundidee des Grid-Computing als Basistechnologie verstanden werden, wodurch die Ausführung von Experimenten als Post-Technologie gedeutet wird.

Der steigende Bedarf an simulations- und datengesteuerter Wissenschaft bedeutet jedoch nicht, dass experimentelle Wissenschaft an Bedeutung verliert. Durch die Weiterentwicklung von Technologien entstehen neue, revolutionäre Experimente und die Entstehung von Hochgeschwindigkeitsnetzen ermöglicht eine Integration der Experimente in wissenschaftliche Prozesse. Durch diese Integration können Wissenschaftler weltweit den Zugriff auf Experimente teilen und somit gemeinsam an einem Problem arbeiten. Wladawsky-Berger beschreibt die Tendenz zur Virtualisierung von Ressourcen mit dem Ziel der Vereinfachung des Zugriffs und

somit der Steigerung der Effizienz. Hauptbestandteil des Grid-Computing ist die Verwendung des Internets zur Kommunikation. Nutzer haben das Bedürfnis, unabhängig vom Aufenthaltsort auf alle Ressourcen zuzugreifen und diese mit anderen Nutzern zu teilen. Durch diese Entwicklung steigt die Zahl der virtualisierten Ressourcen stetig an, Ressourcen werden on-demand verfügbar. Diese Entwicklung zeigt sich auch durch das Aufkommen von RLS. Diese ermöglichen es Nutzern, unabhängig von Zeit und Ort, auf Ressourcen und Experimente zuzugreifen. Ein wissenschaftlicher Nutzen von RLS ist direkt durch die Notwendigkeit von Experimenten gegeben.

3.2 Grid Checkliste

Um die Frage der Einordnung von Remote Laboratory Systems in Grid Computing erläutern zu können, muss zunächst bestimmt werden, was einen Grid auszeichnet.

Foster definiert hierzu eine Checkliste mit drei Punkten[9]:

1.) Ein Grid koordiniert und integriert Ressourcen und Nutzer, die nicht von einer zentralen Kontrollinstanz verwaltet werden und aus unterschiedlichen Domänen, zum Beispiel aus unterschiedlichen Firmen oder unterschiedlichen Abteilungen einer Firma stammen und gewährleistet dabei Sicherheits-, Vertrags-, Abrechungs- und Mitgliedschaftsaspekte.

Dieser Punkt wird durch RLS erfüllt. Ressourcen stellen die Experimente und Labore dar, auf die Nutzer zugreifen können. Die Hardwareinstallationen sind unabhängig voneinander und unterliegen keiner zentralen Kontrollinstanz. Die Verwaltung der Installationen übernehmen die jeweiligen Institute und Abteilungen. Die Koordination der Ressourcen erfolgt durch die Softwarekomponenten des RLS selbst, welche als Middleware zwischen Laboren und Nutzeranwendungen gesehen werden kann. Die beschriebenen Aspekte werden ebenfalls durch das RLS gewährleistet. Diese verfügen über eine Authentifizierung der Nutzer und sind für die Sicherheit der Daten zuständig.

2.) Ein Grid verwendet standardisierte, offene und vielseitige Protokolle und Interfaces und beschreibt grundlegende Aspekte wie Authentifizierung, Ressourcenfindung die Verwaltung des Zugriffs auf Ressourcen. Es darf sich dabei nicht um ein Anwendungsspezifisches System handeln.

Die hier beschriebenen Anforderungen sind nicht zwangsweise durch ein RLS abgedeckt. Es existieren RLS von Organisationen und Forschungseinrichtungen, die nach Außen hin abgeschlossen sind und nicht auf die Vorteile offener Protokolle und Interfaces angewiesen sind. Der Rahmen der Diplomarbeit beschränkt sich jedoch auf RLS des GOLC. Diese verwenden ausschließlich standardisierte und offene Protokolle und Programmiersprachen. Die Interfaces sind sorgfältig spezifiziert und dokumentiert. Dadurch ist eine leichte Integration neuer Teilnehmer gewährleistet.

3.) Ein Grid stellt nicht-triviale Servicequalitäten bereit und erlaubt die koordinierte Nutzung der Ressourcen unter Rücksichtnahme auf Servicequalitäten wie Verfügbarkeit, Antwortzeit und Durchsatz.

Auch diese Anforderungen werden durch RLS erfüllt und sind im folgenden Unterkapitel beschrieben.

Betrachtet man die in Kapitel 2 beschriebene Architektur, wird schnell klar, dass RLS eine Grid Architektur beschreiben. Die Hardwareinstallationen eines RLS stellen die Ressource-Schicht der Architektur dar und bilden die Ressourcen als Gegenstand des Interesses durch andere Teilnehmer. Sie werden ergänzt durch die Simulationen, welche ebenfalls als Ressourcen betrachtet werden können. Die Connectivity-Schicht beschreibt die Schnittstelle zwischen den Hardwareinstallationen eines Rigs und den Verwaltungssystemen des RLS. Komponenten wie das Buchungssystem sind für die Koordination der Ressourcen verantwortlich. Hierbei spielt insbesondere die Reservierung von Rigs eine Rolle, welche die Nutzung gewissermaßen einschränkt. Die Anwendungsschicht wird durch das bereitgestellte Portal und die Interfaces für Nutzer beschrieben, welche zum Definieren und Ausführen von Experimenten eingesetzt werden. Ergänzend zur Übereinstimmung der Architektur erfüllen RLS alle Anforderungen der Checkliste von Foster und können somit als Grid-Architektur aufgefasst werden.

3.3 Anforderungen im Grid Computing

Nachdem gezeigt wurde, dass RLS eine Grid-Architektur darstellen, befasst sich dieses Unterkapitel mit wichtigen Rahmenbedingungen der Architektur.

Standards

Der Einsatz von Standards ist im Grid Computing sehr wichtig, da viele heterogene Systeme und Komponenten zusammenarbeiten müssen. Durch das Open Grid Forum definierte Standards basieren auf Webservices als Schnittstelle zwischen Systemen und greifen so auf anerkannte Standards wie WSDL und SOAP zurück. Diese Tendenz lässt sich auch bei RLS beobachten, wo ebenfalls Webservices mittels SOAP oder Rest eingesetzt werden.

Virtualisierung

Virtualisierung ist ein zentraler Punkt des Grid Computing. Diese ermöglicht es Ressourcen über Netzwerke verfügbar zu sein und somit eine Entkopplung von der physischen Präsenz zu erreichen. Dadurch erhalten Personen aus entfernten Domänen die Möglichkeit, die verfügbaren Ressourcen mit zu benutzen. Zur Koordination verwenden viele Grids einen Ressource Broker. Dieser kann wie eine Warteschlange

betrachtet werden, der man eine Aufgabe übermittelt, welche zum passenden Zeitpunkt weiter delegiert wird. Virtualisierung von Ressourcen ist der Kerngedanke von RLS. Hierzu werden Hardwareaufbauten für Experimente über Schnittstellen zugänglich gemacht. Eine Koordination der Ressourcen findet ebenfalls statt. Diese besteht entweder aus der Reservierung von Ressourcen oder der Nutzung einer Warteschlange.

Verfügbarkeit

Die Verfügbarkeit von Ressourcen ist ein wichtiger Punkt. Im Idealfall sollten Ressourcen on-demand zu jeder Zeit zugänglich sein. Für Informationen und Daten ist dies wesentlich einfacher als für Hardware, da diese immaterielle Güter sind und jederzeit repliziert werden können. Ressourcen die aus Hardware bestehen, wie in RLS, können nicht on-demand verfügbar sein. Dies kann allenfalls für Simulationen erreicht werden, vorausgesetzt es ist keine hohe Rechenleistung nötig.

Sicherheit

Die Konzepte des Grid Computing führen verschiedene Institutionen und Organisationen zusammen. Unter diesen Gegebenheiten ist ein Sicherheitskonzept von Nöten, welches auf Authentifizierung, Verschlüsselung und Signierung setzt. Es muss sichergestellt werden, dass nur Mitglieder der virtuellen Organisation Zugriff auf die Ressourcen haben. Dies wird über eine Zertifikat-basierte Authentifizierung und eine asymmetrische Verschlüsselung erreicht. Die selben Mechanismen werden für die Interaktion zwischen RLS in dem entwickelten Standard umgesetzt.

3.4 Einordnung in Architekturmodelle

Die Architekturmodelle eines Grid unterscheiden sich nach dem Problem das adressiert werden soll. In einem Wissensgrid steht der Austausch von Informationen und Wissen im Vordergrund. Dabei soll sich das vorhandene Wissen ergänzen um ein größeres Ziel zu erreichen. Ein Datengrid dient der Koordination von Daten aus verschiedenen Domänen. Teilnehmer erhalten hierbei Zugriff auf die Datenbanken anderer Teilnehmer. Ein Rechengrid vereint die Rechenleistung mehrerer Teilnehmer um durch den Zuwachs an Leistung komplexe Probleme adressieren zu können. Es existieren weitere Grid-Modelle wie Ressourcen- oder Servicegrids. Alle Modelle orientieren sich dabei an einem speziellen Aspekt eines Systems. RLS sind nach dieser Einteilung insbesondere Daten- und Rechengrids, da es Teilnehmern ermöglicht wird, auf die Hardwareinstallationen eines anderen Teilnehmers zuzugreifen und die vorhandenen Daten mit zu nutzen. Über batchbasierte Experimente lässt sich Rechenzeit nutzen, um Daten zu erzeugen, die später ausgewertet werden kann. RLS bieten dabei keine Steigerung der Rechenleistung

durch Zusammenarbeit, können jedoch Cluster für komplexe Berechnungen bereitstellen.

[20] beschreibt einen Ansatz basierend auf einer Topologie und stellt den Umfang der virtuellen Organisation in den Fordergrund. Die kleinste Einheit beschränkt sich auf eine einzelne Organisation, die intern Dienste eines Grid zur Verfügung stellt und nach Außen hin abgeschlossen ist. Diese wird Intragrid genannt und besteht aus Komponenten und Systemen zur Nutzung gemeinsamer Daten in einem privaten Netzwerk. In einem Intragrid ist es auf Grund von Sicherheitsaspekten einfacher, Rechen- und Datengrids zu reaslisieren. Ein Extragrid erweitert dieses Konzept und führt zwei oder mehr Intragrids zusammen. Dabei müssen die vorhandenen Systeme zur Kommunikation mit anderen Systemen in der Lage sein. Das dritte Konzept heißt Intergrid und ermöglicht eine offene Teilnahme von virtuellen Organisationen.

Ein wichtiger Aspekt der Einteilung ist die Nennung der geografischen Lage einer Organisation. Betrachtet man ein RLS als Verbund von Instituten, welche Hardwareinstallationen und Experimente bereitstellen, so bildet jedes Institut einen Intragrid und die Teilnahme an einem RLS erweitert dies zu einem Extragrid. Das Konzept der Interoperabilität, welches Gegenstand der Diplomarbeit ist, ermöglicht somit die Bildung eines Intergrid.

3.5 OGSA und OGSI als Standards

Dieses Unterkapitel widmet sich bereits existierenden Standards der Grid Community. Ziel dieser Standards ist der Zusammenschluss von dezentralen und heterogenen Systemen, unabhängig der Plattformen und Strukturen durch die Förderung von Kompatibilität, Erweiterbarkeit und Portabilität.

Die Open Grid Service Architecture [21] definiert eine Serviceorientierte Architektur des Grid Computing für den Einsatz in Industrie und Wissenschaft. Sie definiert fundamentales Verhalten von Grid Services, wodurch ein Framework zur Interoperabilität entsteht. Hauptziel der OGSA ist die Virtualisierung von Ressourcen. Dabei orientiert sich die OGSA an offenen und anerkannten Standards und Protokollen, wie Webservices unter Verwendung von WSDL und SOAP. Da die Funktionaltität von Webservices für Anwendungen nicht ausreicht [22], wurde diese zu Grid Services erweitert und in einem weiteren Standard namens Open Grid Services Infrastructure (OGSI) [23] spezifiziert. Grid Services erweitern das Konzept von Webservices und spezifizieren Konzepte zu Servicedaten, Servicegruppen, Benachrichtigungen und eine Verwaltung des Lebenszyklus. Des Weiteren wird adressieren Sie das Problem der Zustandslosigkeit von Webservices und erweitern das Single-Port-Type Konzept.

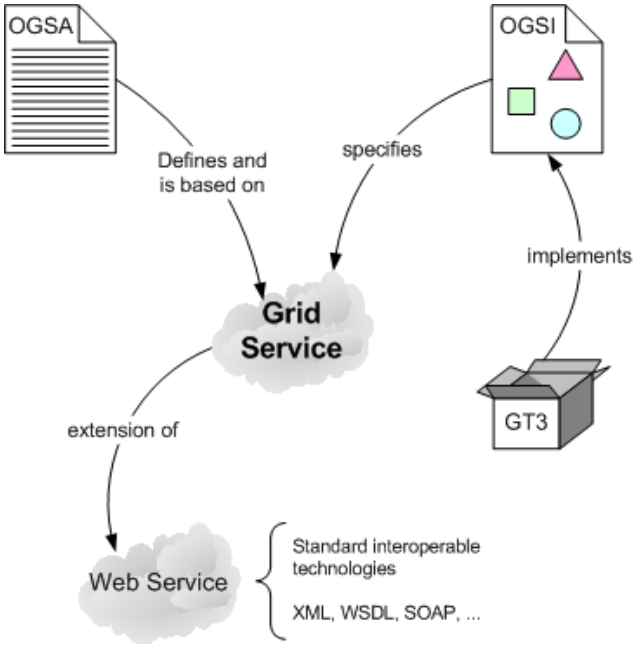


Abbildung 6 Grid Service durch OGSA und OGSA [42]

Remote Laboratory Systems im Kontext des GOLC verfügen bisher über keine derartige Infrastruktur, könnten die hier beschriebene OGSA jedoch ohne Einschränkungen verwenden und ihre Ressourcen über Grid Services verfügbar machen. Sowohl RLS, als auch die OGSA setzt auf anerkannte und offene Standards zur Realisierung der Architektur. Der in dieser Diplomarbeit entwickelte Standard setzt erste Akzente in Richtung der Struktur einer OGSA, indem Webservices definiert werden, welche eine Teilung der Ressourcen unter Mitgliedern des GOLC ermöglichen.

3.6 Einordnung in wissenschaftliche Paradigmen

[17] befasst sich mit dem Gedanken von Jim Gray, dass datenbasierte Wissenschaft ein viertes Paradigma der Wissenschaft darstellen kann. Das Wort Paradigma steht als Synonym für Muster oder Herangehensweise und wird häufig im Kontext von Problemlösungen verwendet. Als wissenschaftliches Paradigma wird folglich die Herangehensweise beschrieben, die zur Erlangung wissenschaftlicher Erkenntnisse nötig ist. Um ein viertes Paradigma zu beschreiben, müssen zunächst die vorangegangenen drei Paradigmen erläutert werden. Laut Gray ist das erste wissenschaftliche Paradigma die empirische Wissenschaft. Dabei werden Vorgänge und Verhaltensweisen in der freien Natur beobachtet und durch Experimente, Beobachtung und Befragung Rückschlüsse auf die dadurch gegebenen Gesetze gezogen. So wurde die Forschung der Schwerkraft durch einen fallenden Apfel

angetrieben und der Zusammenhang zwischen dem Umlauf des Mondes und den Gezeiten des Meeres beobachtet.

Das zweite Paradigma definiert die theoretische Wissenschaft. Hier werden theoretische Überlegungen zu definierten Problemen angestellt und versucht eine Problemlösung auf Basis von Modellen und Beweisen zu erarbeiten. Das Paradigma beschreibt also vier Schritte, das Definieren von Beobachtungen die untersucht werden sollen, das Aufstellen eines Modells, das die Beobachtungen beschreibt, der Erarbeitung einer Lösung für das Modell und den Beweis, dass die Lösung korrekt ist. Viele Problemstellungen sind jedoch sehr komplex und können nicht ohne weiteres modelliert oder gelöst werden. Die Lösung des Modells muss dabei oft durch eine aufwendige Simulation oder komplexe Rechnungen ergänzt werden. Diese können nicht von einem Menschen ausgeführt werden und bedürfen des Einsatzes von Rechenleistung durch Computer. Dadurch ist das dritte Paradigma definiert, welches als computerbasierte Wissenschaft bezeichnet wird. Dieses Paradigma ist erst im vergangenen Jahrhundert entstanden und wurde durch die Weiterentwicklung von Computern und Software und die damit verbundene Steigerung der Rechenleistung erreicht. Angetrieben wurde diese Entwicklung auch durch die Verwendung neuer Technologien und Ideen, wie beispielsweise dem behandelten Grid Computing und verbesserten Verfahren zur Berechnung der Daten. Dieses Paradigma ist noch nicht ausgeschöpft und wird mit der Erfindung des Buchdrucks vergleichen, welcher mehr als tausend Jahre gebraucht hat um den heutigen Stand zu erreichen.

Der Einsatz von Computersystemen zur Simulation und Unterstützung der Modelle in der theoretischen Wissenschaft hat zu einem enormen Wachstum der Daten und einer Bildung von komplexeren Datensets geführt. Die Menge der erzeugten Daten überschreitet die Menge, die durch vorhandene Computersysteme abgearbeitet werden kann. Oftmals müssen Datensets zusammengefasst oder archiviert werden und können nicht vollständig analysiert und dokumentiert werden. Wie bereits erwähnt, erzeugt der LHC des Cern eine jährliche Datenmenge von 15 Petabyte. Gray definiert nun ein viertes Paradigma, welches als datenbasierte Wissenschaft bezeichnet wird. Dieses Paradigma beschreibt ein Rahmenwerk zur Integrationen und Zusammenarbeit der ersten drei Paradigmen. Ein Beispiel beschreibt die Wettervorhersage als Zusammenarbeit der Paradigmen. Der Grundstein wurde durch Beobachtungen der Natur gelegt und durch Modelle der theoretischen Wissenschaft beschrieben und analysiert. Um eine Vorhersage zu treffen werden computerbasierte Simulationen verwendet, welche auf den definierten Modellen basieren und an Hand gegebener Parameter eine Vorhersage treffen. Die Auswertung der Simulationsdaten schließt das vierte Paradigma mit ein. Das Buch beinhaltet weiterhin Beschreibungen für den Nutzen in verschiedenen wissenschaftlichen Disziplinen, wie Lehre, Gesundheit und Energiemanagement und hebt einen Paradigmenwechsel weg von Hypothesen und hin zu Mustern hervor.

Die Idee von RLS lässt sich keinem festen Paradigma zuordnen. Viel eher ist es so, dass RLS alle vorgestellten Paradigmen unterstützen oder abbilden können. Virtuelle

Experimente und Labore können Beobachtungen der realen Welt abbilden und simulieren, jedoch nicht ersetzen. Forschern kann es so ermöglicht werden, Beobachtungen zu reproduzieren und unter verschiedenen Bedingungen und Aspekten auszuführen um verifizierbare Ergebnisse zu erhalten. RLS üben durch diese Möglichkeit eine unterstützende Wirkung auf die empirische Wissenschaft aus. Das Definieren der Beobachtungen und das Erstellen von Modellen in der theoretischen Wissenschaft ist nicht Bestandteil von RLS, wodurch eine Zuordnung in das zweite Paradigma ebenfalls entfällt. Die Hardwareinstallationen von RLS bieten die Möglichkeit Modelle auszuwerten und können so zur Analyse und Problemlösung sowie dem Überprüfen der Lösung beitragen. Es werden folglich die letzten zwei Schritte der theoretischen Wissenschaft unterstützt. Die computerbasierte Wissenschaft aus dem dritten Paradigma ist Gegenstand der Idee von RLS. Sowohl interaktive, als auch batchbasierte Experimente können Simulationen von Modellen darstellen und somit das Hauptmerkmal des dritten Paradigmas abbilden. Die datenbasierte Wissenschaft wird ebenfalls durch RLS unterstützt. Hardwareinstallationen können Programme zur Auswertung entstandener Daten bearbeiten und somit zum Konzept der Datenanalyse beitragen. Der Kern von RLS liegt nach wie vor auf dem Bereitstellen von Experimenten und nicht dem Bereitstellen von Rechenleistung und Datenspeicher. Wie gezeigt wurde ist eine Eingliederung in ein festes Paradigma schwer. Die naheliegende Einordnung wäre in das Paradigma der computerbasierten Wissenschaft. Jedoch beschreibt dies nur einen Teil der Möglichkeiten von RLS. Das Fazit lautet somit, dass alle Paradigmen entweder Gegenstand von RLS sind oder durch diese unterstützt werden können.

3.7 Zusammenfassung und Einordnung

Es wurde zunächst die wissenschaftliche Notwendigkeit von RLS gezeigt. Experimente sind ein fundamentaler Bestandteil der Wissenschaft und die Virtualisierung von Experimenten hat einen hohen Stellenwert in der heutigen, computergetriebenen Wissenschaft. Wir haben gezeigt, dass RLS über die Eigenschaften einer Architektur des Grid Computing verfügen und die von Foster aufgestellte Checkliste für Grid Computing mit RLS verglichen. Des Weiteren wurden wichtige Rahmenbedingungen vergleichen. Die Motivation von Grid Computing und RLS ist identisch, nämlich die Virtualisierung von Ressourcen. RLS wurden von Anfang an als Szenario für Grid Computing dargestellt. Grid Computing abstrahiert das Konzept von RLS und somit stimmen beide in fast allen Punkten überein. Ein einzelnes RLS stellt bereits einen Grid dar. Durch Interaktionen zwischen mehreren RLS lässt sich dies bis zu einem Intergrid auf globaler Ebene erweitern.

Eine feste Einordnung in ein von Gray beschriebenes Paradigma konnte nicht getroffen werden. RLS können alle Paradigmen entweder unterstützen oder bilden eine direkte Anwendung des Paradigmas.

4. Konzepte und Design

In diesem Kapitel werden die Konzepte des Standards beschrieben. Um einen Standard zu Interoperabilität zu schreiben ist es zunächst wichtig, den Begriff zu definieren und in den Kontext von RLS zu bringen. Anschließend wird die zu entwickelnde Komponente, deren Eingliederung in ein bestehendes System sowie funktionale und nicht-funktionale Anforderungen erläutert. Abschluss des Kapitels bildet eine Auswahl des verwendeten Interfaceprotokolls. Es standen drei Protokolle zur Auswahl die an Hand ihrer Vor- und Nachteile erläutert und gegeneinander abgegrenzt werden.

4.1 Interoperabilität

Bevor wir Interoperabilität in den Kontext von Remote Laboratory Systems bringen, ist es zunächst wichtig, den Begriff Interoperabilität im Allgemeinen zu definieren und gegen den Begriff Kompatibilität abzugrenzen.

Definition Kompatibilität

Der Duden definiert Kompatibilität von Hard- und Softwaresystemen als zusammenpassend, beziehungsweise die Möglichkeit zweier Systeme, sich zusammen zu setzen lassen.

[24] definiert Kompatibilität als die Verträglichkeit, Zusammenschließbarkeit und Austauschbarkeit von verschiedenen Geräten, Systemen und Programmen unterschiedlicher Hersteller. Sie ist eine der Hauptaufgaben der nationalen bzw. internationalen Standardisierungsbemühungen. Zur Kompatibilität tragen Standards bei, insbesondere die international anerkannten Schnittstellen und Regelungen für offene Netze und Systeme.

Definition Interoperabilität:

[25] definiert Interoperabilität als die Fähigkeit unabhängiger, heterogener Systeme, möglichst nahtlos zusammenzuarbeiten, um Informationen auf effiziente und verwertbare Art und Weise auszutauschen bzw. dem Benutzer zur Verfügung zu stellen, ohne dass dazu gesonderte Absprachen zwischen den Systemen notwendig sind.

Eine weitere Definition nach [26] beschreibt Interoperabilität als die Fähigkeit eines Programms oder Systems (dessen Schnittstellen vollständig offengelegt sind) mit

anderen gegenwärtigen oder zukünftigen Produkten oder Systemen ohne Einschränkungen hinsichtlich Zugriff oder Implementierung zusammenzuarbeiten bzw. zu interagieren.

Kompatibilität definiert die Zusammensetzbarkeit zweier bestehender Systeme und kann als Bestandteil der Interoperabilität angesehen werden. Interoperabilität erweitert diese Definition, in dem eine Zusammenführung der Systeme nicht nötigt ist, sondern beide Systeme autark bleiben und miteinander über eine Komponente interagieren. In diesem Zusammenhang bezieht sich die Kompatibilität auf die Schnittstellen zwischen den Systemen. In beiden Fällen sind die Systeme in der Lage, Funktionalitäten und Ressourcen zu teilen. Der Zusammenhang zwischen Kompatibilität und Interoperabilität ist in Abbildung 7 dargestellt.

4.1.1 Interoperabilität von RLS

Beide Definitionen von Interoperabilität stimmen in wesentlichen Punkten überein. Zunächst werden Systeme als voneinander unabhängig beschrieben. Die erste Definition geht von existierenden, unabhängigen und heterogenen Systemen aus. Als heterogene Softwaresysteme werden Systeme bezeichnet, die auf inkompatiblen Systemplattformen laufen, unterschiedliche Programmiersprachen und Techniken oder unterschiedliche Standards verwenden [25]. Heterogene Systeme unterscheiden sich in wesentlichen Teilen oder Elementen und verfügen oftmals nicht über identische Schnittstellen, was eine Zusammenarbeit nur über die Anpassung bestehender Systeme durchführbar macht. Die zweite Definition nennt explizit auch zukünftige Systeme, über die noch keine Informationen verfügbar sind. Bestehende RLS, deren Betreiber Mitglieder des GOLC sind, verfügen über ein standardisiertes Datenmodell, welches die Definition der zu entwickelnden Interfaces vereinfacht. Es kann davon ausgegangen werden, dass sich auch zukünftige Mitglieder an Spezifikationen des Datenmodells halten werden. Änderungen am Datenmodell

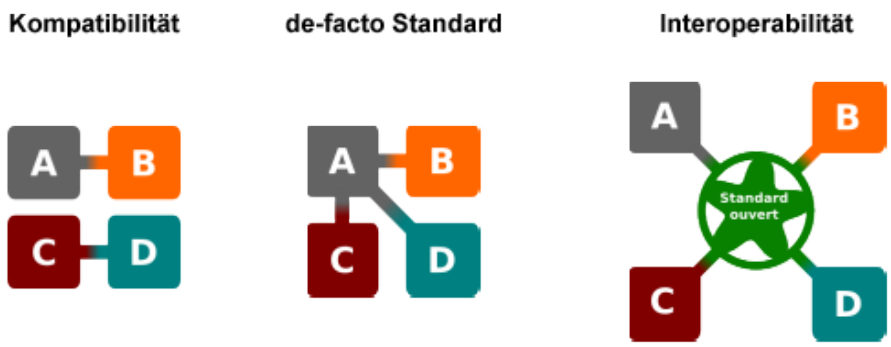


Abbildung 7 Kompatibilität und Interoperabilität [26]

implizieren deshalb direkt eine Änderung an den Interfaces des zu entwickelnden Standards.

Eine weitere Anforderung an den Standard ist die Verwendung von offenen und anerkannten Protokollen zur Kommunikation zwischen den Systemen. Die zweite Definition gibt an, dass keine Einschränkungen bezüglich der Implementierung und dem Zugriff vorhanden sein dürfen. Der Austausch von Nachrichten muss somit über wohl definierte Standardprotokolle erfolgen. Dies fördert den Einsatz von Webservices, da diese ein anerkannter und offener Standard sind und keine Einschränkungen bezüglich der Interfaces oder Programmiersprachen haben.

Als Interoperabilität von RLS kann somit die Zusammenarbeit der einzelnen, physisch getrennten, RLS mit dem Ziel verstanden werden, einem Nutzer eines bestimmten Systems den Zugriff auf andere beteiligte Systeme zu ermöglichen. In Abgrenzung zur Definition sind die Interaktionen der RLS dabei nicht nur auf den Austausch von Informationen beschränkt. Im Vordergrund der Interaktionen steht das Teilen der Funktionalität und der Zugriff auf Ressourcen von beteiligten Systemen, wodurch Ressourcen gebündelt und größere Ziele erreicht werden können. Die Schnittstellen der Systeme müssen zu diesem Zwecke kompatibel sein.

4.2 Systemübersicht

Aufgabe der Diplomarbeit ist die Entwicklung einer Komponente zur Interoperabilität von Remote Laboratory Systems für das Global Online Laboratory Consortium. Die Komponente hat die Aufgabe, als föderative Schnittstelle zwischen bestehenden RLS zu fungieren und somit eine Zusammenarbeit der einzelnen Systeme zu ermöglichen. Die zu entwickelnde Komponente wird in einem separaten Standard beschrieben und definiert, welcher als Richtlinie für Teilnehmer der Zusammenarbeit verstanden wird. Da die Komponente für alle teilnehmenden RLS spezifiziert wird, muss die Implementierung dezentral erfolgen. Auch wenn das Datenmodell standardisiert ist, verfügen die Systeme bisher über keine Komponente für den Zugriff von externen Systemen. Des Weiteren unterscheiden sich die Verwaltungssysteme, was eine einheitliche Implementierung der Services nicht möglich macht. Durch eine dezentrale Implementierung ist bereits ein Teil des Datenschutzes gewährleistet. Die Möglichkeit einer Zuordnung von Personen und Daten darf unter keinen Umständen für externe Systeme möglich sein. Eine zentrale Komponente müsste Zugriff auf den gesamten Datenbestand aller RLS haben und würde eine solche Zuordnung erlauben.

4.2.1 Integration

Abbildung 8 zeigt die Integration der Komponente in bestehende Systeme. Hierzu sind zwei Teile erforderlich, die Services welche auf Seite des Providers eingesetzt werden und ein Client, der auf Seite des Consumers benutzt wird, um die Services aufzurufen. Die Anbindung der Services erfolgt dabei über zwei bestehende Komponenten. Das Buchungssystem ist verantwortlich für die Reservierung von Hardwareinstallationen und ist somit notwendig um externen Systemen den Zugriff auf Hardwareressourcen zu ermöglichen. Die zweite Komponente beschreibt die Ontology API zur Verwaltung der Entitäten in der Datenhaltung.

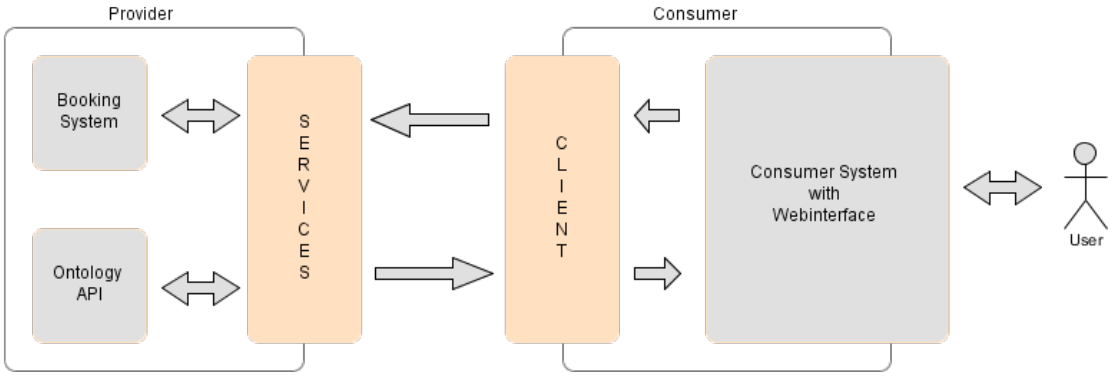


Abbildung 8 Systemübersicht

4.3 Funktionale Anforderungen

Dieses Unterkapitel beschreibt die funktionalen Anforderungen an die zu entwickelnde Komponente. Diese orientieren sich an der Funktionalität eines RLS. Die ersten beiden Anwendungsfälle beschreiben Abfragen über die angebotenen Dienste. Dazu zählen die Servicelevel eines Providers, welche in Kapitel 5 definiert sind. Der zweite Anwendungsfall betrifft die Komponente selbst und eine Abfragemöglichkeit, welche Profile und Operationen unterstützt werden.

Als wichtigster Bestandteil ist eine Verwaltung der Experimente nötig. Nutzern soll die Möglichkeit geboten werden, Experimente zu erzeugen, zu lesen, zu ändern und zu entfernen. Experimente bezeichnen hierbei aus Sicht der Datenhaltung nicht die Experimente im herkömmlichen Sinn sondern die Durchführung eines bereits in einem InteractionPackage definierten Experiments. Eine verantwortliche Person, beispielsweise der Professor einer Übungsgruppe, kann ein Experiment definieren. Die Studenten dieser Übungsgruppe können dann das definierte Experiment ausführen, wozu ein Container gebraucht wird um die Informationen und Ergebnisse zu speichern. Dieser Container wird in der Datenhaltung als ExperimentDefinition bezeichnet. Für ein einfacheres Verständnis wird im Folgenden der Begriff Experiment mit dem Begriff ExperimentDefinition gleichgesetzt. Hat ein Student ein

Experiment erzeugt, muss ihm die Möglichkeit gegeben werden, ein Experiment durchzuführen und gegebenenfalls wieder abzubrechen.

Zur Durchführung eines Experiments können mehrere Funktionen wichtig sein. Für interaktive Experimente wird ein Interface zur Anzeige und für batchbasierte Experimente zur Spezifikation der Parameter benötigt. Dies schließt sowohl das Starten als auch das Verwenden eines Interfaces ein. Sowohl interaktive, als auch batchbasierte Experimente können eine Reservierung eines Rig erfordern. Somit müssen dem Nutzer Operationen zur Abfrage von freien Zeitslots, dem Tätigen einer Reservierung, dem Abbrechen einer Reservierung und einer Statusabfrage geboten werden. Alternativ können batchbasierte Experimente das Einfügen in eine Warteschlange erfordern. Die benötigten Operationen betreffen das Abfragen des Status eines Experiments in einer Warteschlange sowie das Einfügen und Entfernen. Um die Verwaltung der Experimente abzuschließen werden Abfragen bezüglich der eigenen erstellten Experimente und deren Status benötigt sowie eine Abfrage der Ergebnisse eines abgeschlossenen Experiments.

Zur Suche nach Rigs muss die Komponente drei wichtige Operationen unterstützen. Als erstes müssen Informationen über ein Rig eingeholt werden können. Diese Informationen beinhalten Details zur Hardwareinstallation selbst sowie zu den untergeordneten Rigs. Hierzu ist eine Spezifikationsdatei erforderlich, welche die Details beschreibt. Als zweite Operation muss eine Abfrage zum Status eines Rigs spezifiziert werden. Diese teilt dem Nutzer mit ob ein Rig online und verfügbar ist. Die dritte Operation überprüft, ob der Nutzer die nötigen Rechte besitzt um mit dem Rig zu interagieren.

Abbildung 9 Anwendungsfälle

4.4 Nicht-funktionale Anforderungen

Datenschutz:

Die zu entwickelnde Komponente ist für den Einsatz in einem universitären Umfeld gedacht und muss deshalb besondere Rücksicht auf den Datenschutz nehmen. Bei einer Kommunikation mit externen Systemen ist darauf zu achten, dass die übertragenen Daten nicht mit realen Personen in Verbindung gebracht werden können.

Konsistenz:

Bei der Verarbeitung von Operationen muss auf die Konsistenz der Daten geachtet werden. Operationen müssen entweder vollständig oder gar nicht ausgeführt werden. Geänderte Daten müssen dauerhaft in den Datenbanksystemen gespeichert sein.

Sicherheit:

Zur Einschränkung der Nutzung ist eine Authentifizierung der Nutzer erforderliche. Diese soll so effizient wie möglich gestaltet werden. Ergänzend sollen die Inhalte der Nachrichten verschlüsselt werden. Drittparteien sollen nicht die Möglichkeit haben auf die Daten der Nachrichten zugreifen zu können. Dies schließt insbesondere die Ergebnisse von Experimenten ein. Um Manipulationen vorzubeugen, soll sichergestellt werden, dass keine Inhalte einer Nachricht verändert wurden.

Verfügbarkeit:

Die Komponente soll einen hohen Grad an Verfügbarkeit haben. Angebotene Funktionalitäten müssen rund um die Uhr auf Abruf verfügbar sein.

Wartbarkeit:

Die zu entwickelnde Komponente soll einfach zu warten sein und eine einfache Umsetzung von Änderungen ermöglichen. Da die Buchungssysteme der einzelnen RLS nicht identisch sind, soll ein Prototyp implementiert werden, welcher die Services ohne Aufrufe von Verwaltungssystemen implementiert. Bestandteil der Wartbarkeit ist eine saubere und vollständige Dokumentation der zu entwickelnden Klassen.

Erweiterbarkeit:

Die zu entwickelnde Komponente stellt eine Basisversion der Implementierung der Interfaces dar. In der Zukunft ist eine Anpassung und Erweitertung sehr wahrscheinlich. Deshalb soll die Komponente leicht erweiter- und anpassbar sein.

Migration:

Die Komponente soll einfach auf bestehende Systeme migriert werden können und deshalb eine einfache Konfiguration und Bereitstellung bieten. Dabei ist zu berücksichtigen, dass die verwendeten Technologien und Anwendungsserver von einander abweichen können.

4.5 Interfaceprotokoll

Dieses Unterkapitel soll die Frage des verwendeten Protokolls zur Implementierung der Services klären. Zur Auswahl stehen dabei Remote Procedure Calls (RPC), SOAP

und REST. Alle Protokolle haben Stärken und Schwächen welche nacheinander beschrieben und anschließend verglichen werden.

4.5.1 Vorstellung der Optionen

Die Idee für Remote Procedure Calls geht in das Jahr 1976 zurück[27]. RPCs sollten das Teilen von Ressourcen in Netzwerken und den Aufruf von Funktionen in unterschiedlichen Namensräumen ermöglichen. Der Ablauf ist simpel und basiert auf einem Modell aus Client und Server. Zunächst sendet der Client eine Anfrage mit der gewünschten Funktion und den Eingabeparametern an den Server. Der Server ruft die Funktion auf und bearbeitet die Anfrage. Anschließend wird das Ergebnis zurück gesendet. Die Dauer der Bearbeitung ist unbekannt, weshalb eine asynchrone Kommunikation, damals über UDP, benötigt wird. Heute existieren Frameworks und Weiterentwicklungen zu diesem Protokoll. XML-RPC stellt eine solche Weiterentwicklung dar und basiert auf XML zur Spezifikation der Funktion und Parameter. Für die Parameter ist ein festes Set an Datentypen definiert, über welches der Aufbau der Nachricht erfolgt. JSON-RPC [28] ist eine zweite Weiterentwicklung und basiert auf der JavaScript Object Notation.

SOAP wurde bereits in den Grundlagen dieser Arbeit besprochen, weswegen auf eine erneute Beschreibung verzichtet wird. SOAP baut auf dem Ansatz von RPC auf. Jedoch wird RPC separat behandelt um auf JSON-RPC einzugehen. REST [29] steht für Representational State Transfer und entspricht eher einem Programmierparadigma als einem Protokoll. REST abstrahiert die Architekturelemente und stellt Ressourcen in den Mittelpunkt des Modells. Rest fordert von Ressourcen, dass sie über eine eindeutige Adresse verfügbar sind und geht dabei nicht auf die Details der Implementierung einzelner Komponenten oder die Verwendung bestimmter Protokolle ein. Jede Ressource adressiert eine Entität in einem Datenmodell und beschreibt eine zugehörige Funktionalität. Die vorhandenen Ressourcen teilen sich ein Interface, welches zur Kommunikation zwischen Client und Server dient. Üblicherweise werden die Interaktionen durch HTTP realisiert, wobei die Operationen von HTTP wie Get, Post, Put und Delete zur Manipulation von Daten verwendet werden.

4.5.2 Diskussion der Vor- und Nachteile

Dieses Unterkapitel erläutert die Vor- und Nachteile der Optionen. Dabei wird im Besonderen die Komplexität der Implementierung, der Einsatz von Standards, die Existenz von Frameworks und Sicherheitsaspekte berücksichtigt.

RPC bietet eine einfache Möglichkeit zur Implementierung von verteilten Anwendungen und unterstützt dabei einen modularen Aufbau. Die Nutzung der Funktionalität und die Kommunikation sind effizient und leichtgewichtig. RPC definiert keinen Standard sondern beschreibt eine Methode zur Kommunikation zwischen Server und Client. Anwendungen sind somit nicht standardisiert und können

sich in wesentlichen Teilen unterscheiden. Es existieren keine Möglichkeiten zur Beschreibung und Definition von Schnittstellen, wodurch eine Wiederverwendbarkeit nur schwer zu erreichen ist. Die Akzeptanz in Forschung und Industrie ist gegenüber SOAP und REST relativ gering. JSON-RPC bietet eine höhere Verfügbarkeit von Bibliotheken und Werkzeugen, jedoch nicht vom selben Umfang wie SOAP oder REST. Die Technologie gilt als veraltet und als durch SOAP abgelöst. Die Sicherheit in RPC-Anwendungen ist, ebenso wie bei REST, nicht spezifiziert. Es müssen externe Technologien wie SSL verwendet werden.

SOAP bietet einen guten Ansatz, basierend auf standardisierten und anerkannten Technologien. Die Struktur von SOAP und die Definition der Services über Schnittstellenbeschreibungen erlauben einen hohen Grad an Wiederverwendbarkeit. SOAP bietet umfangreiche Möglichkeiten zur Definition von verwendeten Elementen in XSD und WSDL. Die definierten Interfaces lassen sich leicht anpassen. Diese Definitionen führen jedoch zu einer komplexeren Struktur als beispielsweise in einer REST-Anwendung. SOAP ist seit langem ein anerkanntes Protokoll in Forschung und Industrie. Dies hat zur Entwicklung von hochgradigen Werkzeugen wie SoapUI und Frameworks geführt. SOAP wird um die WS-* Spezifikationen erweitert, welche zur Definition von nicht-funktionalen Anforderungen genutzt werden. Es können Verträge zwischen Konsument und Anbieter ausgehandelt, Transaktionen beschrieben und Nachrichten verschlüsselt oder eine Authentifizierung spezifiziert werden. In JSON oder REST Anwendungen liegen diese Aufgaben bei den Entwicklern und verfügen nicht über Standards. Die Implementierung lässt sich dank Schnittstellenbeschreibungen und Tools zur Generierung sehr einfach gestalten. Das Senden von Dateien als Anhang stellt einen großen Vorteil dar. Dateien müssen nicht mehr als Binärdaten in die Nachricht kodiert werden und müssen so nicht mit der Nachricht selbst übertragen werden. Dennoch erzeugen SOAP-Nachrichten einen großen Overhead. Möchte man zum Beispiel einen einzelnen Zahlenwert übertragen, können die Nachrichten um mehr als das zehnfache größer sein als bei einer REST-Nachricht.

Representational State Transfer stellt eine einfache Umsetzungsmöglichkeit für Dienste dar und kann in jeder Programmiersprache verwendet werden. Die Interaktionen sind leichtgewichtig und bieten eine geringe Nutzung von Bandbreite und Ressourcen. Die Nachrichten sind visuell lesbar, wodurch Ergebnisse direkt im Browser angezeigt werden können. Diese Möglichkeit ist für die zu entwickelnden Dienste jedoch nicht von Bedeutung. Zentrales Element von REST sind die Ressourcen, welche durch den Entwickler spezifiziert werden müssen. Eine Implementierung durch REST bedarf keiner Toolkits oder umfangreicher Frameworks. REST kann mittels HTTP als Protokoll und URI als Identifizierungsmechanismus auf weit verbreitete und allgemein anerkannte Technologien zurückgreifen. Die Zahl an hochwertigen Frameworks steigt zunehmend parallel zur Akzeptanz des Paradigmas in Forschung und Industrie. Gute Frameworks für nicht-funktionale Anforderungen wie Sicherheit, sind noch nicht auf

dem selben, ausgereiften Stand von SOAP. Die Sicherheit in REST Anwendungen ist nicht spezifiziert. Das Paradigma verlangt, dass alle Informationen in der Anfrage enthalten sind. Jedoch sollten sicherheitskritische Parameter nie als Parameter in URIs gesendet werden. Es liegt in der Verantwortung des Entwicklers ein Konzept zur Lösung dieses Problems zu erarbeiten. Des Weiteren definiert das Paradigma keinen Mechanismus zur Authentifizierung von Nutzern. Dieser muss ebenfalls durch den Entwickler spezifiziert werden. Oftmals verwendet werden SSL zur Sicherung der Kommunikation und HTTP- oder IP-Authentifizierung. Diese sind jedoch weniger flexibel als beispielsweise die Spezifikationen in WS-Security.

4.5.3 Fazit

Es wurden die drei Optionen RPC, SOAP und REST vorgestellt sowie die Vor- und Nachteile wichtiger Konzepte erläutert. Die Interfaces des Standards basieren auf Operationen, was eine Verwendung von RPC oder SOAP impliziert. Rest als Ressourcen-basiertes Framework würde die Umsetzung der Profile erschweren. In Kapitel 4.1 wurde bereits die Bedeutung von offenen und anerkannten Standards hervorgehoben. Die Bestandteile einer SOAP-Nachricht sind klar und deutlich spezifiziert. Sowohl die Beschreibungen der Parameter, als auch die der Operationen verwenden standardisierte Sprachen, welche eine Wiederverwendbarkeit erleichtern. Sowohl RPC als auch REST liefern keine klare Beschreibung der Interfaces. Die Struktur der Nachrichten ist in beiden Fällen abhängig von den verwendeten Protokollen und Nachrichtenformaten. Auch die Standardisierung spricht folglich für den Einsatz von SOAP als Protokoll. Allerdings erzeugen SOAP-Nachrichten einen großen Overhead bei der Kommunikation. Sowohl REST als auch RPC sind sehr leichtgewichtige Protokolle. Sollten die Dienste im späteren Betrieb sehr häufig genutzt werden, könnte dies ein Vorteil und somit ein Argument gegen SOAP sein. Die hohe Akzeptanz und die daraus resultierende Breite an Bibliotheken und Frameworks spricht jedoch wieder für den Einsatz von SOAP, ebenso wie die Erweiterungen durch die WS-* Spezifikationen. Diese bieten uns die Möglichkeit nicht-funktionale Anforderungen zu definieren. Insbesondere Aspekte der Sicherheit und Authentifizierung können so einfach und effizient gelöst werden. Bei der Verwendung von RPC und REST müssten diese Aspekte separat behandelt und durch weitere Technologien ergänzt werden. Letzter wichtiger Punkt des Vergleichs ist die Implementierung. SOAP ist hier wesentlich komplexer als RPC oder REST. Diese müssen lediglich HTTP-Anfragen verarbeiten, während bei SOAP die Definition der Datenelemente und Operationen, sowie eine Implementierung von Endpunkten und Ports nötig ist. Einziger Pluspunkt ist das Senden von Dateien, welche als Anhang an eine Nachricht angefügt werden können. Dies ist wichtig, da mehrere Operationen des Standards Dateien senden müssen. In RPC oder REST müssten diese Dateien binär kodiert in die Nachrichten eingefügt werden.

Nachdem die Optionen erläutert und die Vor- und Nachteile diskutiert wurden, fiel die Entscheidung zu Gunsten von SOAP. Manche Punkte sprachen gegen den Einsatz

von SOAP, jedoch waren diese von der Priorität her eher untergeordnet. Zwei der Hauptpunkte für die Entscheidung für SOAP war der hohe Grad an Standardisierung, welcher bei Systemen zur Interoperabilität notwendig ist und die Erweiterung durch die WS-* Spezifikationen.

5. Spezifikation der Profile

Das vorangegangene Kapitel erläutert die Einordnung der neuen Komponente in bestehende RLS, sowie deren funktionale und nicht-funktionale Anforderungen. Dieses Kapitel befasst sich mit der Spezifikation der Profile, welche die Interfaces des Standards definieren und die Funktionalität der spezifizierten Anwendungsfälle abdeckt. Die aktuelle Version des Standards beschreibt sechs Profile, um die grundlegende Funktionalität von beteiligten Remote Laboratory Systemen (RLS) abzudecken. Die Profile, sowie ein Teil der Operationen, waren bereits durch ein erstes Working Draft des GOLC gegeben. Diese umfassten jedoch keine Interfacedefinitionen und waren nicht vollständig. Für ein besseres Verständnis und da die Interfaces mit Ein- und Ausgabeparametern nicht gegeben waren, werden auch die vorhandenen Operationen erläutert. Die Gruppierung der Operationen in Profile orientiert sich an der bereitgestellten Funktionalität und dem Hintergrund der einzelnen Operationen. Die ersten drei Profile definieren Operationen für Abfragen bezüglich des Systems des Providers, der vorhandenen Rigs und RigSets, sowie von Experimenten. Die letzten drei Profile umfassen Operationen für den Zugriff auf Ressourcen. Das erste dieser Profile ist für den Zugriff auf Rigs zuständig und bietet die Möglichkeit von nutzerspezifischen Abfragen, der Verwaltung von Buchungen für ein Rig und der Verwaltung von Experimenten in der Warteschlange eines Rigs. Die übrigen zwei Profile stehen für Operationen im Zusammenhang mit Batch- und interaktiven Experimenten und erlauben das Verwalten und Durchführen solcher Experimente.

In den folgenden Unterkapiteln werden die einzelnen Profile besprochen und deren Interfaces definiert. Für jede Operation wird eine Beschreibung gegeben und die Ein- und Ausgabeparameter mit Kardinalitäten und Datentypen spezifiziert. Ergänzend wird für jedes Profil ein Szenariodiagramm mit Beschreibung geliefert, um den Nutzen abzugrenzen und ein allgemeines Verständnis zu erlangen. Für jede Operation wird vorausgesetzt, dass der Nutzer über die nötigen Rechte verfügt. Dies wird somit bei den Bedingungen nicht gesondert erwähnt. Die Verwendung der Dienste durch andere Systeme erfordert eine umfangreiche Möglichkeit zur Bestimmung von Fehlerursachen. Hierzu wurde eine Liste an Antwortcodes definiert, welche in Unterkapitel 4.7 erläutert wird. Jede Operation hat einen zwingend erforderlichen Antwortcode als Ausgabeparameter und eine optionale Fehlermeldung. Im Rahmen der Profile war es außerdem nötig die Servicelevel sowie eine Spezifikation für individuelle Benutzeroberflächen, zu definieren. Diese sind in den letzten beiden Unterkapiteln beschrieben.

5.1 System Query Profile

Das System Query Profile dient für Abfragen, welche die bereitgestellten Dienstleistungen zwischen einem Consumer und einem Provider beschreiben. Dazu zählen zunächst die unterstützten Profile und deren Versionen, die innerhalb der selben Hauptnummer, das heißt beginnend mit der selben Startnummer, abwärtskompatibel sein müssen. Sollte sich ein Profil grundlegend ändern, oder Operationen von Vorgängerversionen nicht mehr verfügbar sein, muss die Hauptnummer erhöht werden. In diesem Fall werden Vorgängerversionen nicht mehr unterstützt. Als zweiten wichtigen Bestandteil definiert das Profil Operationen zur Abfrage von Serviceleveln. Diese erlauben einem Provider nicht-funktionale Eigenschaften der Angebotenen Dienste zu spezifizieren. Eine Übersicht der definierten Attribute befindet sich in Sektion 4.8 dieses Kapitels. Servicelevel können entweder für ein spezielles RigSet oder für alle verfügbaren Rigs eines Providers abgefragt werden. Die Antworten des Providers müssen visuell lesbar sein, um auf dem Bildschirm des Nutzers angezeigt werden zu können.

5.1.1 Operationen

GetAllProfiles

Beschreibung: Liefert eine Liste aller Profile des Standards und die höchste unterstützte Version

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Output	profileNames	Liste der Profile, bestehend aus Name und unterstützter Version	String	[0..*]
Output	responseCode	Der zurückgelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

GetProfileVersion

Beschreibung: Dient zur Abfrage der unterstützten Version eines Profils

Bedingungen: Es wurde ein korrektes Profil angegeben

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	profileName	Name des abgefragten Profils	String	[1..1]
Output	profileVer	Version des erfragten Profils	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

GetServiceLevel

Beschreibung: Abfrage eines bestimmten Servicelevels, entweder über den Provider oder für ein bestimmtes Rig/RigSet

Bedingungen: Es wurde ein korrektes Attribute angegeben

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	serviceAttribute	Name des abgefragten Servicelevels	String	[1..1]
Input	rigSetID	ID des abgerufenen Rigs oder RigSets	String	[0..1]
Output	serviceAttrVal	Visuell lesbare Aussage über das Servicelevel	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

GetAllServiceLevels

Beschreibung: Liefert eine Liste aller definierten Servicelevel eines Providers oder für ein bestimmtes Rig/RigSet.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	rigSetId	ID des abgerufenen Rigs oder RigSets	String	[0..1]
Output	serviceAttributes	Liste aller Servicelevel inklusive Werten	String	[0..*]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

5.1.2 Szenario

Szenario 1: Abfragen der Profile

Abbildung 10 zeigt ein Szenario, bei dem ein Nutzer alle unterstützten Profile bei einem Provider anfordert oder nur die Version eines bestimmten Profils.

Szenario 2: Abfragen der Service Level

Abbildung 11 zeigt einen Nutzer, der zunächst eine Liste aller Servicelevel eines Providers abfragt, und anschließend die verantwortliche Person für ein definiertes RigSet erfragt.

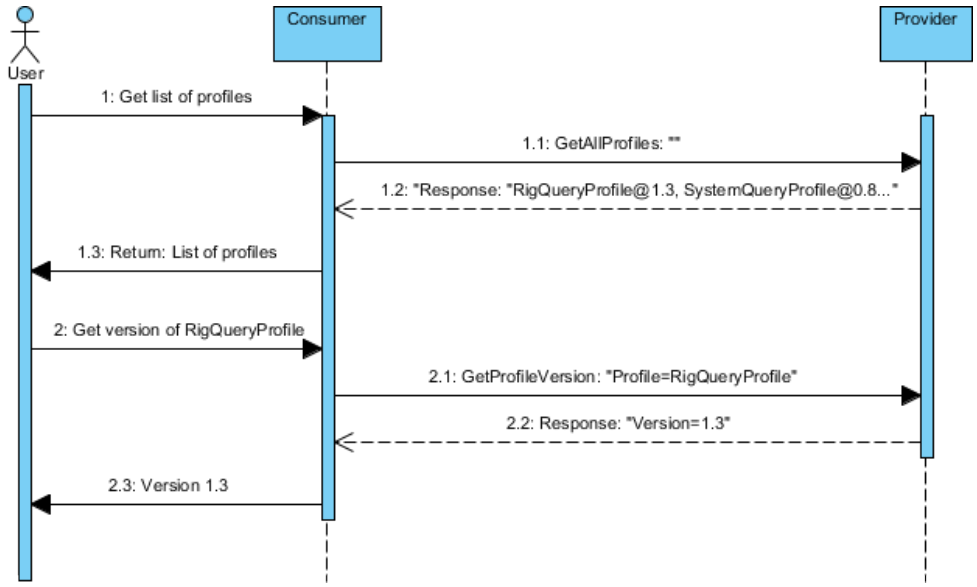


Abbildung 10 Szenario: Abfrage der Profile

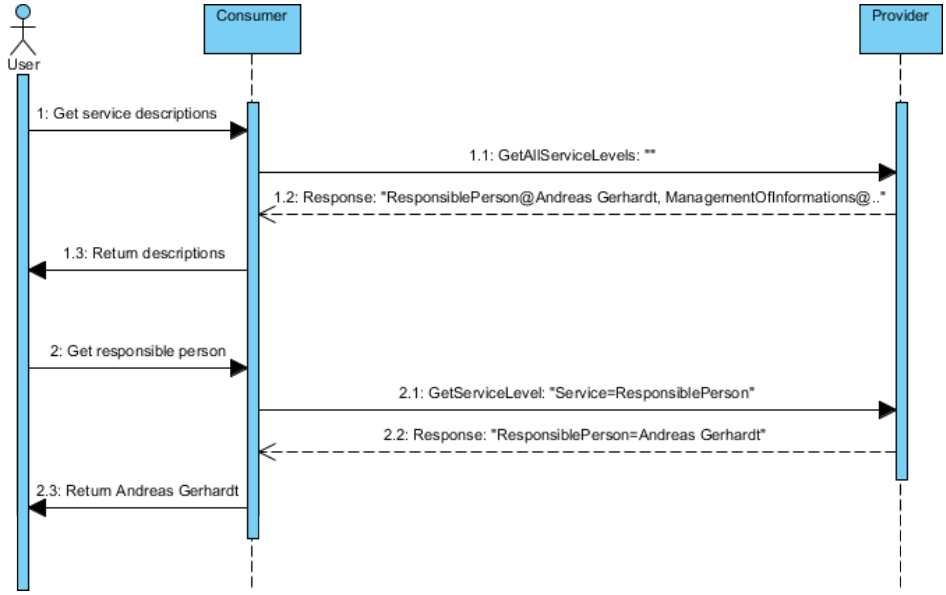


Abbildung 11 Szenario: Abfrage der Servicelevel

5.2 RigQueryProfile

Dieses Profil dient Abfragen bezüglich der von einem Provider bereitgestellten Rigs in einem RLS. Rigs sind innerhalb eines Systems hierarchisch organisiert. Für ein bestimmtes Experiment können mehrere autarke Hardwareaufbauten existieren,

welche zu einem RigSet zusammengefasst werden. Führt man diese Struktur fort, über die Experimentebene hinaus, erhält man eine hierarchisch angeordnete Sicht auf die vorhandenen Rigs. Dadurch muss es dem Nutzer möglich werden, die untergeordneten Rigs abzufragen. Diese Funktion übernimmt die Operation GetRigs. Der Nutzer hat hier die Möglichkeit, entweder die ID eines bestimmten RigSets zu verwenden oder die Operation ohne Eingabeparameter zu starten. Sollte keine ID als Eingabeparameter vorhanden sein, werden die IDs der Top-Level-Rigs des Providers zurückgegeben. Diese kann der Nutzer anschließend verwenden, um sich in der Hierarchie nach unten zu bewegen. Des Weiteren besteht die Möglichkeit, Informationen über ein definiertes Rig oder RigSet abzufragen, sowie den Status des Rigs zu erfahren. Sollte die ID eines Nutzers oder einer Nutzergruppe angegeben sein, bezieht sich die Abfrage auf Rigs oder Eigenschaft, die für diese ID gelten.

5.2.1 Operationen

GetRigs

Beschreibung: Liefert entweder die Top-Level-Rigs eines Providers oder die untergeordneten Rigs eines RigSets. Sollte ein Nutzer, bzw. eine Nutzergruppe, angegeben sein, werden nur die Rigs oder RigSets angezeigt, die von dieser Gruppe verwendet werden können.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	userID	Die ID des Users für den die Abfrage ist	String	[0..1]
Input	groupID	Die ID der Gruppe für die die Abfrage ist	String	[0..1]
Input	rigSetID	Die ID des abgefragten Rig oder RigSet	String	[0..1]
Output	rigSets	Liste aller untergeordneten Rigs	String	[0..*]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

GetRigStatus

Beschreibung: Liefert Informationen über den Status eines Rigs oder RigSets. Dazu zählt, ob ein Rig online ist und zur Nutzung bereit steht sowie eine visuell lesbare Statusnachricht. In dieser kann ein Status durch den Provider angegeben werden, beispielsweise dass das Rig gerade gewartet wird und ab wann es wieder verfügbar ist.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	userID	Die ID des Users für den die Abfrage ist	String	[0..1]
Input	groupID	Die ID der Gruppe für die die Abfrage ist	String	[0..1]
Input	rigSetID	Die ID des abgefragten Rig oder RigSet	String	[1..1]
Output	online	Gibt an, ob das Rig oder RigSet online ist	Boolean	[0..1]
Output	available	Gibt an, ob das Rig oder Rigs des RigSet zur sofortigen Nutzung verfügbar sind	Boolean	[0..1]
Output	statusMessage	Visuell lesbare Statusnachricht des Providers	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

GetRigInfo

Beschreibung: Liefert eine XML-Datei mit Informationen über ein definiertes Rig oder ein RigSet

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	userID	Die ID des Users für den die Abfrage ist	String	[0..1]
Input	groupID	Die ID der Gruppe für die die Abfrage ist	String	[0..1]
Input	rigSetID	Die ID des abgefragten Rig oder RigSet	String	[1..1]
Output	rigSetProperties	XML-File mit einer Provider-spezifischen Liste an Eigenschaften des Rig/Set	XML-String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

5.2.2 Szenario

Nutzung des Rig Query Profile:

Der Nutzer fragt zunächst in (1) die IDs der Top-Level-Rigs des Providers ab und anschließend in (2) die Spezifikationen der Top-Level-Rigs. Hierbei erfährt der Nutzer, dass die Rigs in RigSet 66 Hardwareaufbauten des BotLab-Labors sind. Nun fragt der Nutzer in (3) die Rigs des RigSets ab und kann über die Methode GetServiceLevel des System Query Profile die Hardwarespezifikationen der

einzelnen Rigs abfragen. Hat der Nutzer das gewünschte Rig gefunden, fordert er in (4) den Status des Rigs an. Anschließend hat der Nutzer die Möglichkeit über das Basic Rig Access Profile Zugriff auf das genannte Rig zu erhalten und beispielsweise ein Booking zu initiieren. Dieser Vorgang ist in Abbildung 12 als Bestandteil des Szenarios für das Basic Rig Access Profile beschrieben.

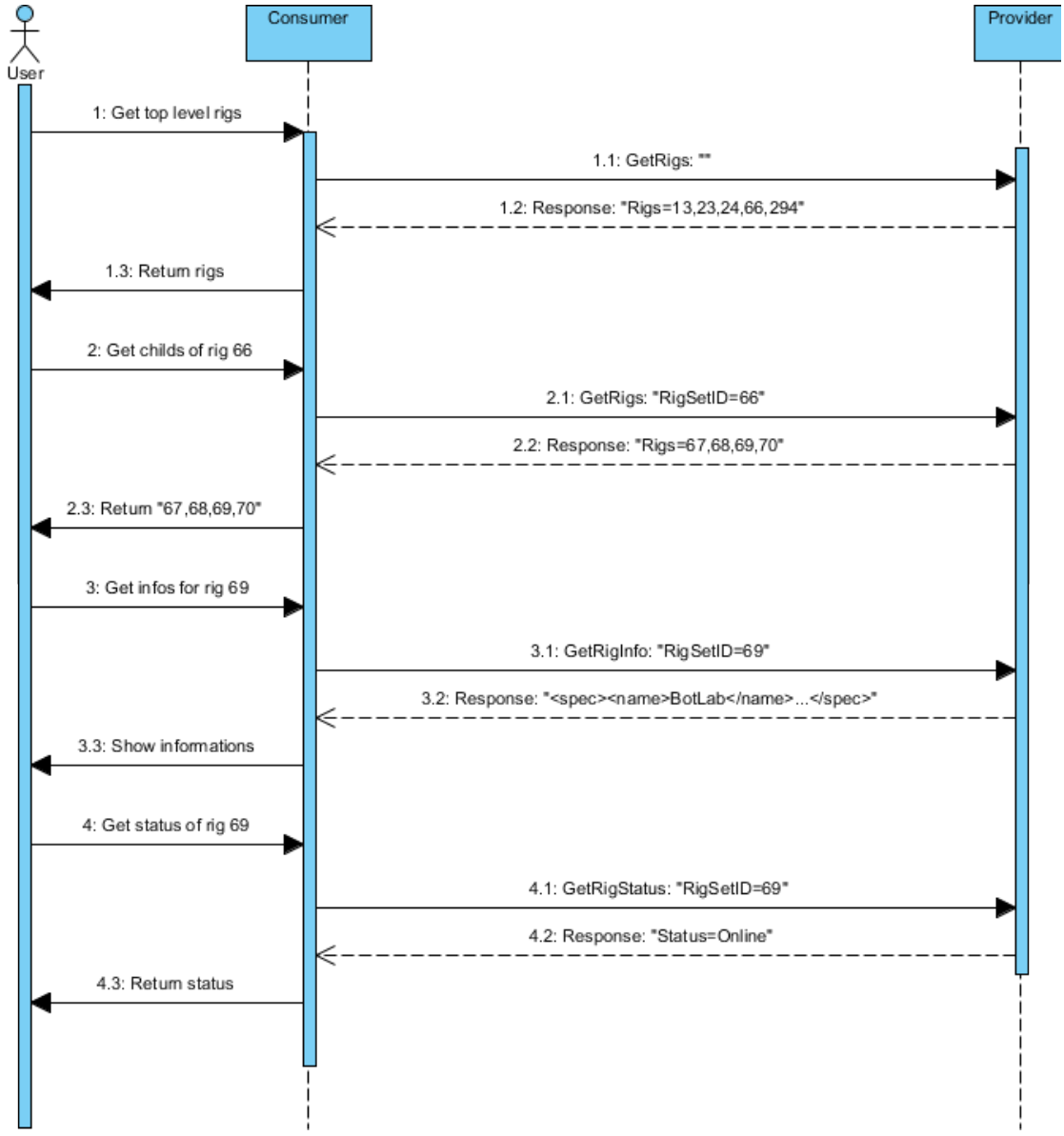


Abbildung 12 Szenario Rig Query Profile

5.3 Experiment Query Profile

Eine wichtige Funktion bei der Nutzung von RLS sind Abfragen bezüglich des Status eines Experiments. Das Experiment Query Profile bietet hier drei Möglichkeiten. Über die Methode GetAllActiveExperiments kann man eine Liste aller aktiven Experimente für einen definierten Nutzer oder eine Nutzergruppe anfordern. Die Liste beinhaltet Paare, bestehend aus ID und Status eines Experiments. Zur Abfrage eines gegebenen Experiments stellt das Profil eine Methode GetExperimentStatus bereit. Zusätzlich lassen sich über die Methode GetExperimentIDsWithStatus alle Experimente mit einem bestimmten Status abfragen. Nach der korrekten Ausführung eines Experiments können die Ergebnisse über die Methode GetExperimentResults abgeholt werden.

5.3.1 Status einer Experimentdurchführung

Hier werden die möglichen Status eines Experiments definiert und beschrieben. Ein Experiment, welches sich in einem der definierten Zustände befindet, wird als aktives Experiment bezeichnet. Abbildung 13 zeigt ein Zustandsdiagramm, welches die Übergänge der Status definiert.

- **Created**: Das Experiment wurde erstellt, aber noch nicht ausgeführt. In der Datenbank Des Providers wurde ein Container zur Speicherung der Ergebnisse und von Informationen zur Durchführung angelegt.
- **Queued**: Das Experiment befindet sich in einer Warteschlange und wartet auf die Ausführung.
- **Executing**: Das Experiment wird gerade ausgeführt.
- **Completed**: Die Ausführung des Experiments ist abgeschlossen. Mögliche Resultate können nun angefordert werden.
- **Abandoned**: Die Ausführung des Experiments wurde durch den Nutzer abgebrochen. Dies kann bei beiden Arten von Experimenten auftreten.
- **Terminated**: Die Ausführung des Experiments wurde durch den Provider

abgebrochen, bspw. wenn die Durchführung eines Batch-Experiments den zugesicherten Zeitrahmen überschreitet.

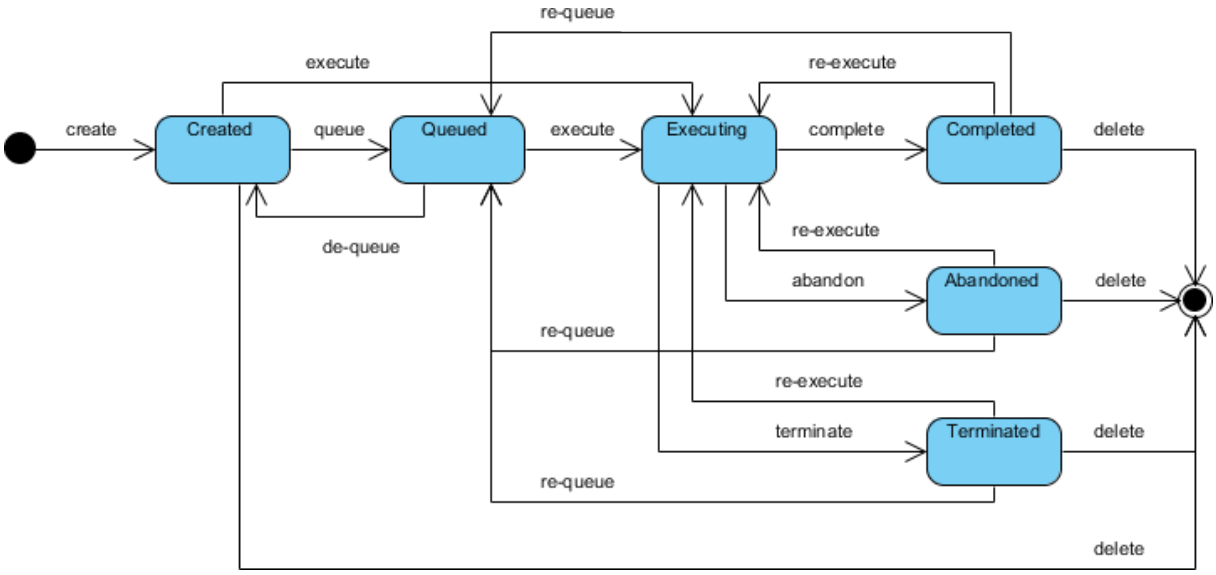


Abbildung 13 Zustandsdiagramm Experimentstatus

5.3.2 Operationen

GetAllActiveExperiments

Beschreibung: Liefert eine Liste aller aktiven Experimente, bestehend aus Paaren von ID und Status eines Nutzers oder einer Nutzergruppe.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	userID	Die ID des Users für den die Abfrage ist	String	[0..1]
Input	groupID	Die ID der Gruppe für die die Abfrage ist	String	[0..1]
Input	rigSetID	Die ID des abgefragten Rig oder RigSet	String	[1..1]
Output	experiments	Liste mit IDs aller aktiven Experimente	String	[0..*]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

GetExperimentStatus

Beschreibung: Liefert den Status eines bestimmten Experiments zum Zeitpunkt der Abfrage

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID des abgefragten Experiments	String	[1..1]
Output	expStatus	Der Status des Experiments zum Zeitpunkt der Abfrage	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

AbandonExperiment

Beschreibung: Fordert das Abbrechen eines laufenden Experiments an. Die Ausführung wird abgebrochen und der Status der Durchführung auf Abandoned gesetzt. Ergebnise werden nicht gespeichert.

Bedingungen: Das Experiment befindet sich in der Ausführung

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	Die ID des abzubrechenden Experiments	String	[1..1]
Input	userID	Die ID des Nutzers, der Eigentümer ist	String	[0..1]
Input	groupID	Die ID der Gruppe, die Eigentümer ist	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

GetExperimentResults

Beschreibung: Liefert die Ergebnisse eines korrekt ausgeführten Experiments oder partielle Ergebnisse für Batch-Experimente, die sich noch in der Ausführung befinden.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	Die ID des abgefragten Experiments	String	[1..1]
Output	results	Resultate des Experiments in einer Datei	File	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

5.3.3 Szenario

Nutzung des Experiment Query Profiles:

Der Nutzer holt zunächst in (1) eine Liste seiner aktiven Experimente beim Provider. Experiment 104 ist bereits in der Warteschlange und wartet auf seine Ausführung. Der Nutzer entschließt sich zu warten und fragt später in (2) erneut den Status des Experiments ab. Da sich das Experiment immer noch in der Warteschlange befindet, beschließt der Nutzer in (3) eine Liste aller beendeten Experimente abzufragen. Der Provider teilt dem Nutzer mit, dass Experiment 567 beendet ist, woraufhin sich der Nutzer in (4) dazu entschließt, erst die Ergebnisse dieses Experiments zu laden, während sich Experiment 104 noch in der Warteschlange befindet.

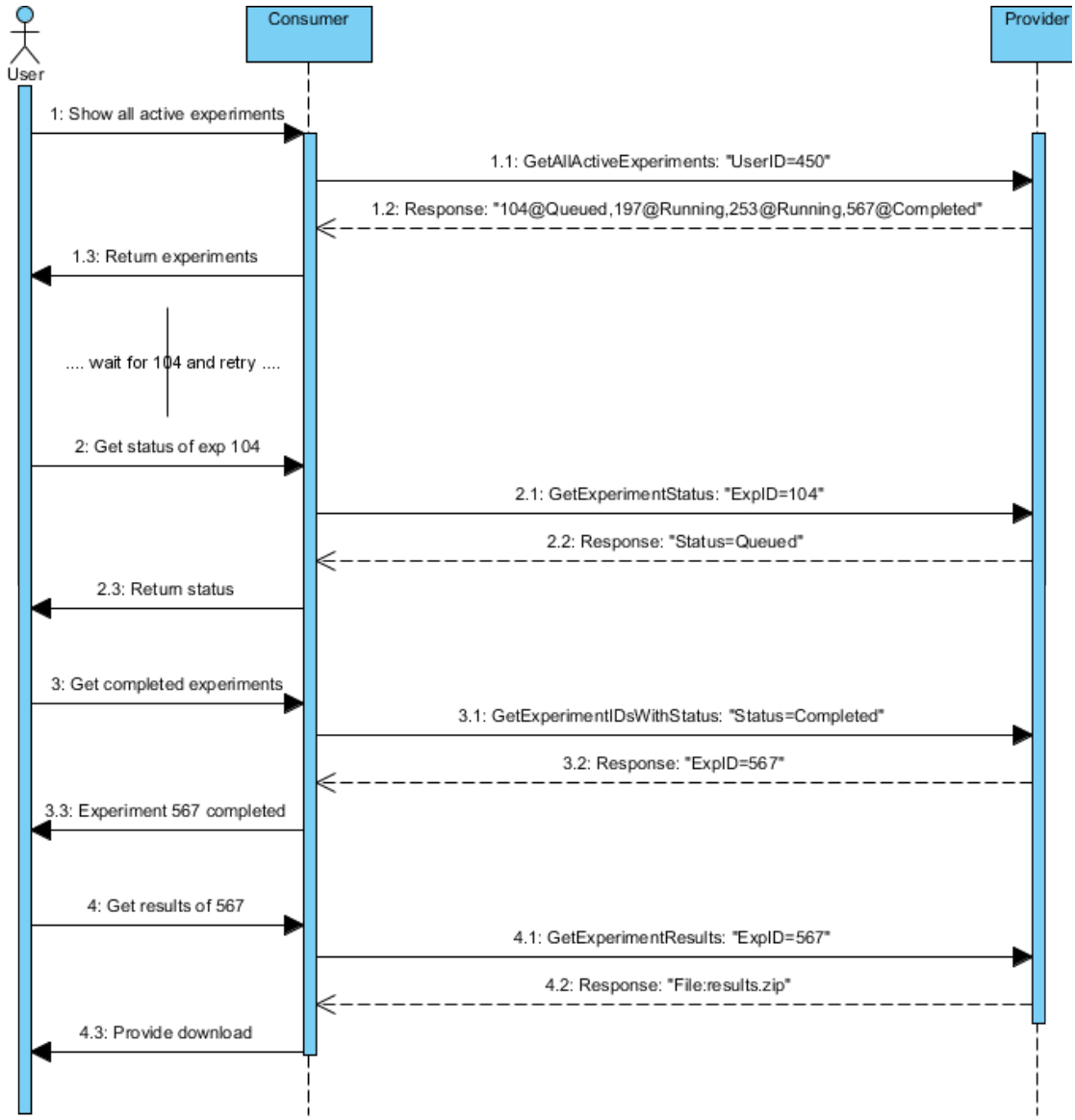


Abbildung 14 Szenario: Verwaltung von Experimenten

5.4 Basic Rig Access Profile

Die in diesem Profil definierten Operationen erlauben einem Nutzer Zugriff auf Rigs zu erhalten und wichtige Informationen abzufragen. Über die Methode RigAccessType lässt sich abfragen ob ein Rig gebucht werden muss oder ob sich Experimente auf diesem Rig in eine Warteschlange einreihen müssen. Die Methode RigAvailability bietet die Möglichkeit, sich über freie Zeitslots für eine Buchung zu informieren. Eine wichtige Funktion stellt das Buchen von Rigs dar. Mittels der Methode RigBooking lässt sich eine Buchung für ein spezifiziertes Experiment anfordern. Im Regelfall muss die Buchung erst durch den Eigentümer des Rigs bestätigt werden. Der Nutzer kann optional eine eMail-Adresse bei der Buchung angeben über welche eine Bestätigung empfangen werden kann. Die Methode RigBookingStatus erlaubt es einem Consumer kontinuierlich den Status einer angeforderten Buchung abzufragen. Eine Buchung lässt sich über die Methode RigBookingCancel wieder abbrechen. Zusätzlich erlaubt die Methode RigBookingsQuery, eine Liste aller existierenden Buchungen für einen bestimmten Nutzer oder eine Nutzergruppe anzufordern. Mit RigQueue, RigQueueStatus und RigQueueCancel sind äquivalente Methoden definiert, welche die Funktionalität zur Verwendung von Warteschlagen bereitstellen. Alternativ zur RigBookingsQuery-Methode lassen sich Experimente in Wartenschlangen, mit dem Status Queued, durch die Methode GetExperimentIDsWithStatus des Experiment Query Profiles abfragen.

5.4.1 Operationen

RigAccessType

Beschreibung: Liefert eine Aussage, ob ein spezifiziertes Rig gebucht werden muss oder ob Experimente in eine Warteschlange eingefügt werden müssen.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	rigSetID	Die ID des abgefragten Rig oder RigSet	String	[1..1]
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Grupp an Nutzern	String	[0..1]
Output	bookable	Gibt an, ob ein Rig gebucht werden muss	Boolean	[0..1]
Output	queueable	Gibt an, ob das Rig über eine Warteschlange verfügt.	Boolean	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigAvailability

Beschreibung: Liefert eine Liste an verfügbaren Zeitslots für einen definierten Zeitraum und ein spezifiziertes Rig.

Bedingungen: Der spezifizierte Zeitraum liegt in der Zukunft

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	rigSetID	Die ID des abgefragten Rig oder RigSet	String	[1..1]
Input	startTime	Die Startzeit des abgefragten Zeitslots	Long	[1..1]
Input	endTime	Die Endzeit des abgefragten Zeitslots	Long	[1..1]
Output	timeFrames	Liste aller verfügbaren Zeitslots innerhalb des spezifizierten Zeitslots	Long	[0..*]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigBooking

Beschreibung: Fordert eine Buchung für ein bestimmtes Experiment auf einem spezifizierten Rig an.

Bedingungen: Der Nutzer darf das angegebene Rig buchen und der Zeitraum der Buchung ist nicht vergeben

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	rigSetID	Die ID des Rig, das verwendet werden soll	String	[1..1]
Input	userID	Die ID des Eigentümers der Reservierung	String	[0..1]
Input	groupID	Die ID der Grupp, die Eigentümer der Reservierung ist	String	[0..1]
Input	expID	Die ID der Experiment Definition der Reservierung	String	[1..1]
Input	startTime	Die Startzeit der Reservierung	Long	[1..1]
Input	endTime	Die Endzeit der Reservierung	Long	[1..1]
Input	eMail	Die eMail des Nutzers	String	[0..1]
Output	bookingID	Die ID der Reservierung für zukünftige Anfragen	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigBookingStatus

Beschreibung: Fragt den Status einer Buchung ab. Dieser kann drei Werte annehmen. Bei der Anforderung der Buchung wird der Status auf Pending gesetzt. Dies impliziert, dass noch keine Entscheidung getroffen wurde. Sollte

die Buchung bestätigt werden, wechselt der Status auf Confirmed, bei einer Ablehnung auf Denied.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	bookingID	Die ID der abgefragten Reservierung	String	[1..1]
Output	status	Der Status der Reservierung	String	[0..1]
Output	redemption	Gibt an ob das Experiment ausgeführt werden kann	Boolean	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigBookingCancel

Beschreibung: Fordert das Entfernen einer zuvor geforderten oder bereits existierenden Buchung an.

Bedingungen: Der Nutzer ist Eigentümer der Reservierung.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	bookingID	Die ID der Reservierung, die entfernt werden soll	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigBookingsQuery

Beschreibung: Liefert eine Liste aller existierender Buchungen für einen definierten Nutzer oder eine definierte Nutzergruppe.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Gruppe	String	[0..1]
Output	bookings	Liste aller Reservierungen, inklusive Details	String	[0..*]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigQueue

Beschreibung: Fügt ein spezifiziertes Experiment in die Warteschlange eines Rigs ein.

Bedingungen: Der Nutzer ist Eigentümer des Experiments und seine ID ist in der Liste der erlaubten Nutzer

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	Die ID des Experiments	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigQueueStatus

Beschreibung: Fragt den momentanen Status eines zuvor in die Warteschlange eingefügten Experiments ab.

Bedingungen: Der Nutzer ist Eigentümer des Experiments.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	Die ID des Experiments	String	[1..1]
Output	status	Der Status des Experiments	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

RigQueueCancel

Beschreibung: Entfernt ein Experiment aus der Warteschlange eines spezifizierten Rigs.

Bedingungen: Der Nutzer ist Eigentümer des Experiments

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	Die ID des Experiments	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

5.4.2 Szenario

Szenario 1: Reservierung und Abbrechen einer Reservierung:

Aus einer Experimentbeschreibung weiß der Nutzer, dass RigSet 34 Versuchsaufbauten für ein bestimmtes Experiment bereitstellt. Der Nutzer fragt nun in (1) die vorhandenen Rigs ab und kann sich in (2) informieren ob ein bestimmtes Rig gebucht werden muss, oder über eine Warteschlange verfügt. Da das abgefragt Rig gebucht werden muss, informiert sich der Nutzer in (3) über die verfügbaren Zeiten. Über das Client Interface trifft der Nutzer eine Wahl für einen bestimmten Zeitraum und fordert in (4) eine Reservierung des definierten Rigs an. Da eine Reservierung erst durch den Eigentümer des Rigs bestätigt werden muss, informiert sich der Nutzer mittels (5) oder (6) über seine eingereichten Reservierungen und erfährt schließlich, dass die Reservierung akzeptiert wurde. Zu einem späteren

Zeitpunkt erfährt der Nutzer, dass sich der spezifizierte Termin mit einer anderen Aufgabe überschneidet und storniert die Reservierung in (7).

Szenario 2: Einfügen und Entfernen eines Experiments in eine Warteschlange

Dieses Szenario ähnelt dem Szenario für eine Reservierung und beschreibt die Nutzung einer Warteschlange. Der Nutzer informiert sich zunächst in (1) und (2) über ein bestimmtes RigSet und erstellt anschließend in(3) ein Batch Experiment, welches er auf dem gefundenen Rig ausführen möchte. In (4) fügt er das erzeugte Experiment in die Warteschlange und erfährt in einer späteren Abfrage (5), dass das Experiment bereit zur Ausführung ist. Da dem Nutzer in der Zwischenzeit ein Fehler im Algorithmus seines auszuführenden Programms aufgefallen ist, lässt er das Experiment in (6) aus der Warteschlange entfernen.

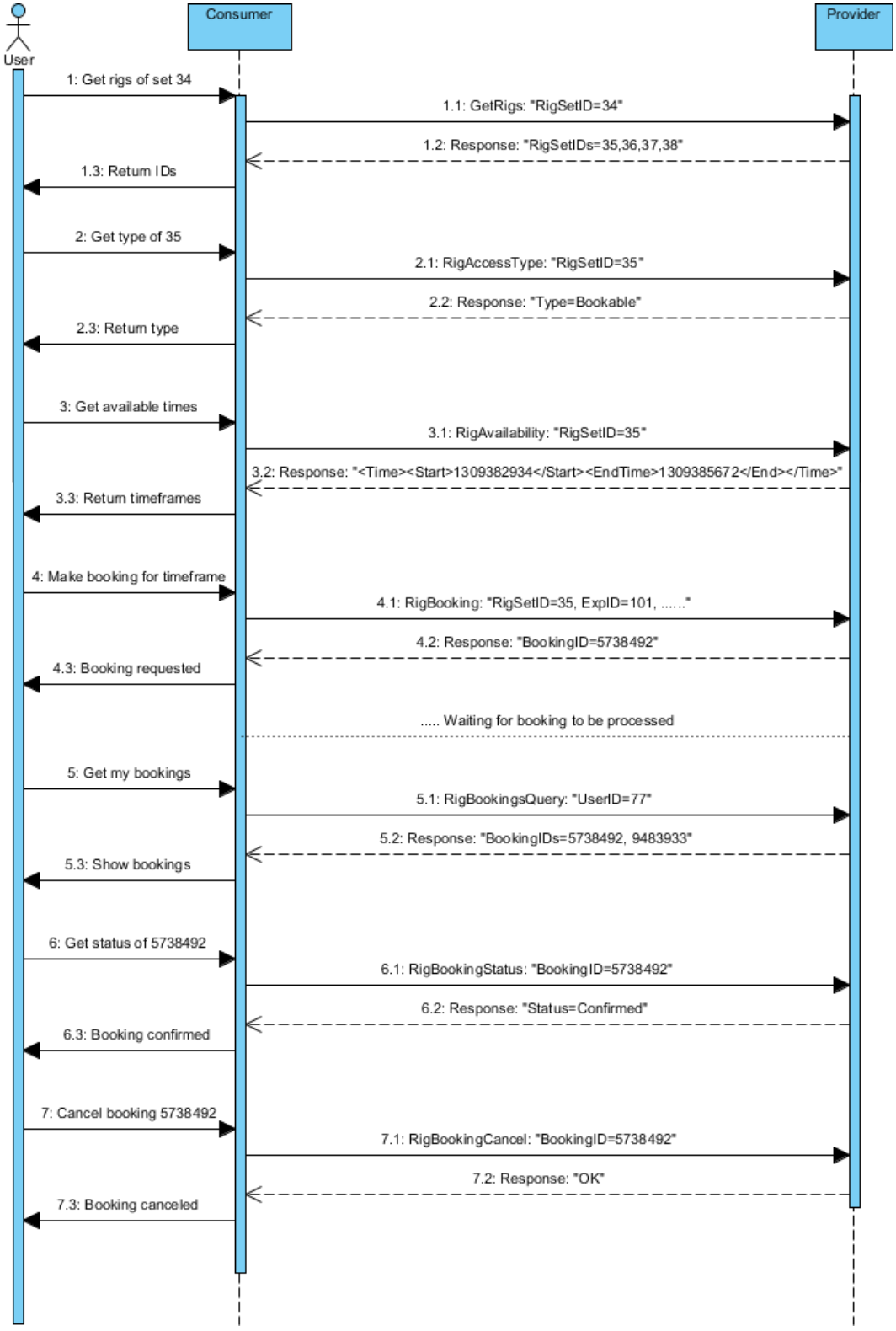


Abbildung 15 Szenario: Buchung eines Rig

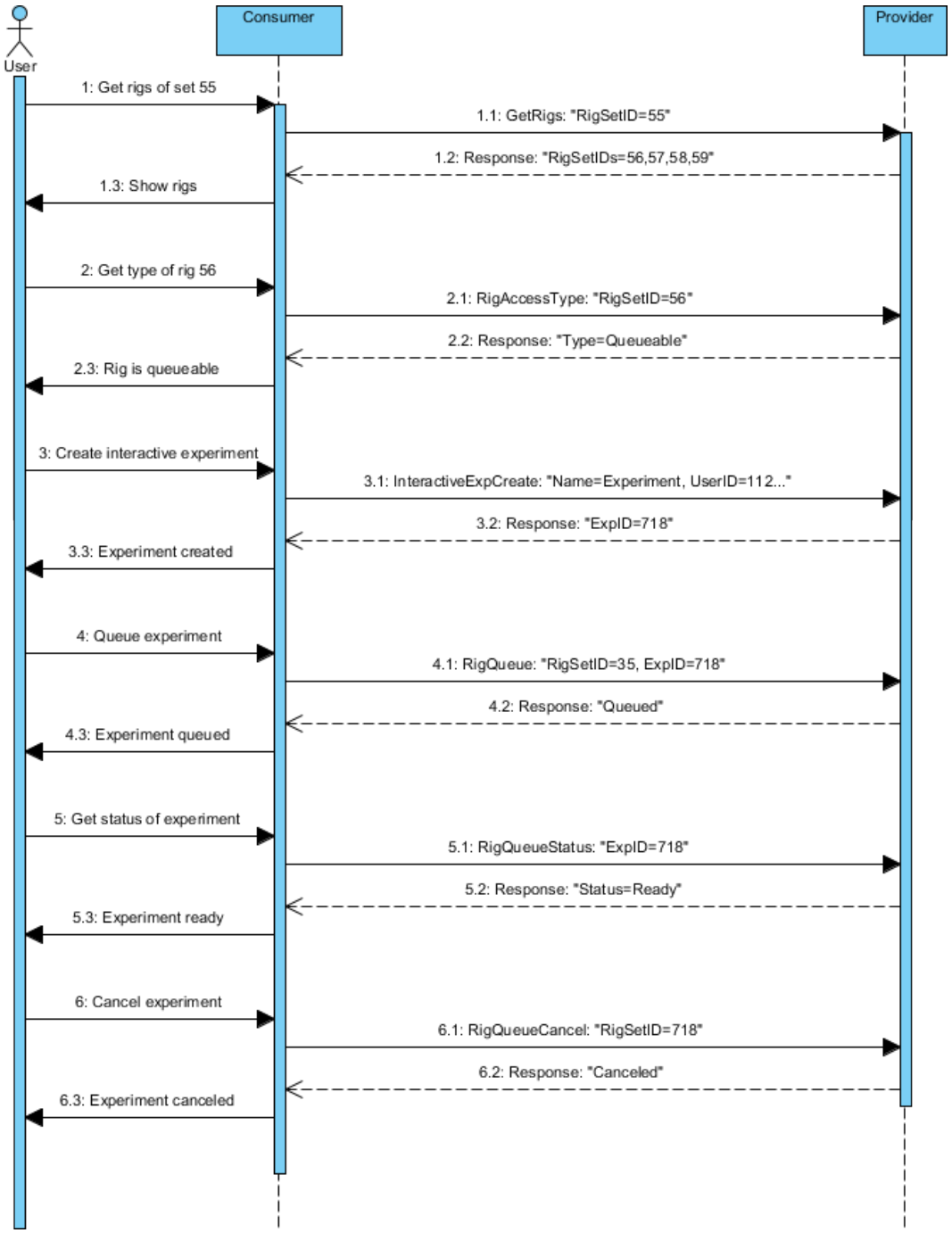


Abbildung 16 Szenario: Nutzung einer Warteschlange

5.5 Basic Batch Experiment Profile

Das Basic Batch Experiment Profile definiert Operationen, welche zur Ausführung von batchbasierten Experimenten notwendig sind. Dazu zählen Operationen, welche die bekannten CRUD-Methoden Create, Read, Update und Delete implementieren. Mittels BatchExpGetSpecs lassen sich Informationen anfordern, die für die korrekte Ausführung eines batchbasierten Experiments erforderlich sind. Im weiteren Verlauf

kann ein Nutzer entweder über die Methode BatchExpLaunchProvIF ein durch den Provider bereitgestelltes Interface starten oder mittels BatchExpLaunchConsIF ein Interface des Consumers benutzen. Für die Generierung eines generischen Interfaces durch den Consumer müssen Informationen zu den Ein- und Ausgaben des Experiments angefordert werden. Diese sind in Sektion 4.9 dieses Kapitels definiert. In beiden Fällen muss über das Interface und die Methode BatchExpSetDefn eine gültige Definition für die Ausführung des Experiments gesetzt werden. Die bereits gesetzten Definitionen lassen sich über die Methode BatchExpGetDefn anfordern und können durch einen erneuten Aufruf von BatchExpSetDefn geändert werden. Ist eine gültige Definition vorhanden, kann die Ausführung des Experiments durch die Methode BatchExpExecute gestartet werden.

5.5.1 Operationen

BatchExpCreate

Beschreibung: Erzeugt einen neuen, leeren Container zur Durchführung eines Experiments. Die Details können im weiteren Verlauf geändert werden.

Bedingungen: Der Nutzer ist berechtig, das angegebene Experiment auszuführen.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	rigSetID	Die ID des Rigs oder RigSets	String	[1..1]
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Gruppe	String	[0..1]
Input	name	Name der Experiment Definition	String	[0..1]
Input	description	Beschreibung der Experiment Definiton	String	[0..1]
Output	expID	ID der erzeugten Experiment Definition	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpRead

Beschreibung: Fordert die Details einer erzeugten Definition eines Experiments an.
Bedingungen: Der Nutzer ist Eigentümer des Containers.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	rigSetID	Die ID des Rigs oder RigSets	String	[1..1]
Output	userID	Die ID des Nutzers	String	[0..1]
Output	groupID	Die ID der Gruppe	String	[0..1]
Output	name	Name der Experiment Definition	String	[0..1]
Output	description	Beschreibung der Experiment Definiton	String	[0..1]

Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpUpdate

Beschreibung: Ändert eine existierende Definition eines Experiments. Alle Parameter sind optional du nur die spezifizierten Parameter werden geändert.

Bedingungen: Der Nutzer ist Eigentümer des Experiments.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Input	rigSetID	Die ID des Rigs oder RigSets	String	[0..1]
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Gruppe	String	[0..1]
Input	name	Name der Experiment Definition	String	[0..1]
Input	description	Beschreibung der Experiment Definiton	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpDelete

Beschreibung: Löscht eine zuvor erstellte Definition eines Experiments.

Bedingungen: Der Nutzer ist Eigentümer des Experiments.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Gruppe	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpLaunchProvIF

Beschreibung: Liefert die URL zu einem durch den Provider bereitgestellten Interface.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	uiURL	Die URL der Benutzeroberfläche	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpGetSpecs

Beschreibung: Fordert Details an, die für die Ausführung des Interfaces eines Experiments wichtig sind. Die gesendete Datei ist in Unterkapitel 4.9 beschrieben.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	specs	XML-Datei mit Informationen zur Ausführung des Experiments	XML-String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpSetDefn

Beschreibung: Wird genutzt um die Parameter zur Ausführung des Experiments zu setzen.

Bedingungen: Der Nutzer ist Eigentümer des Experiments.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Input	expDefn	XML-Datei der Definition des Batch-Experiments	XML-String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpGetDefn

Beschreibung: Liefert die bereits angelegte Definition der Parameter einer Durchführung eines Experiments.

Bedingungen: Der Nutzer ist Eigentümer des Experiments.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	expDefn	XML-Datei der Definition des Batch-Experiments	XML-String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

BatchExpExecute

Beschreibung: Fordert die Ausführung eines Experiments mit gültiger Definition an.

Bedingungen: Alle notwendigen Parameter, die zur Ausführung des Experiments benötigt werden wurden spezifiziert.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

5.5.2 Szenario

Ausführung eines Batch Experiments:

Ein Mitarbeiter eines Instituts erzeugt in (1) ein Batch Experiment, welches für die Forschungsarbeit des Instituts wichtig ist. In (2) möchte er das User Interface starten. Der Consumer erfragt in (3) die nötigen Spezifikationen und in (4) die Adresse des Interfaces. Daraufhin startet er das Interface. Der Mitarbeiter setzt in (5) über das Interface die benötigten Parameter und Definitionen. Später prüft ein zweiter Mitarbeiter die Einstellungen in (6) und entdeckt einen Fehler. Er modifiziert die Parameter in (7) und teilt dem Provider anschließend in (8) mit, dass die Ausführung des Batch-Experiments beginnen kann. Nach der Ausführung können die Ergebnisse über die Methode GetExperimentResults angefordert werden.

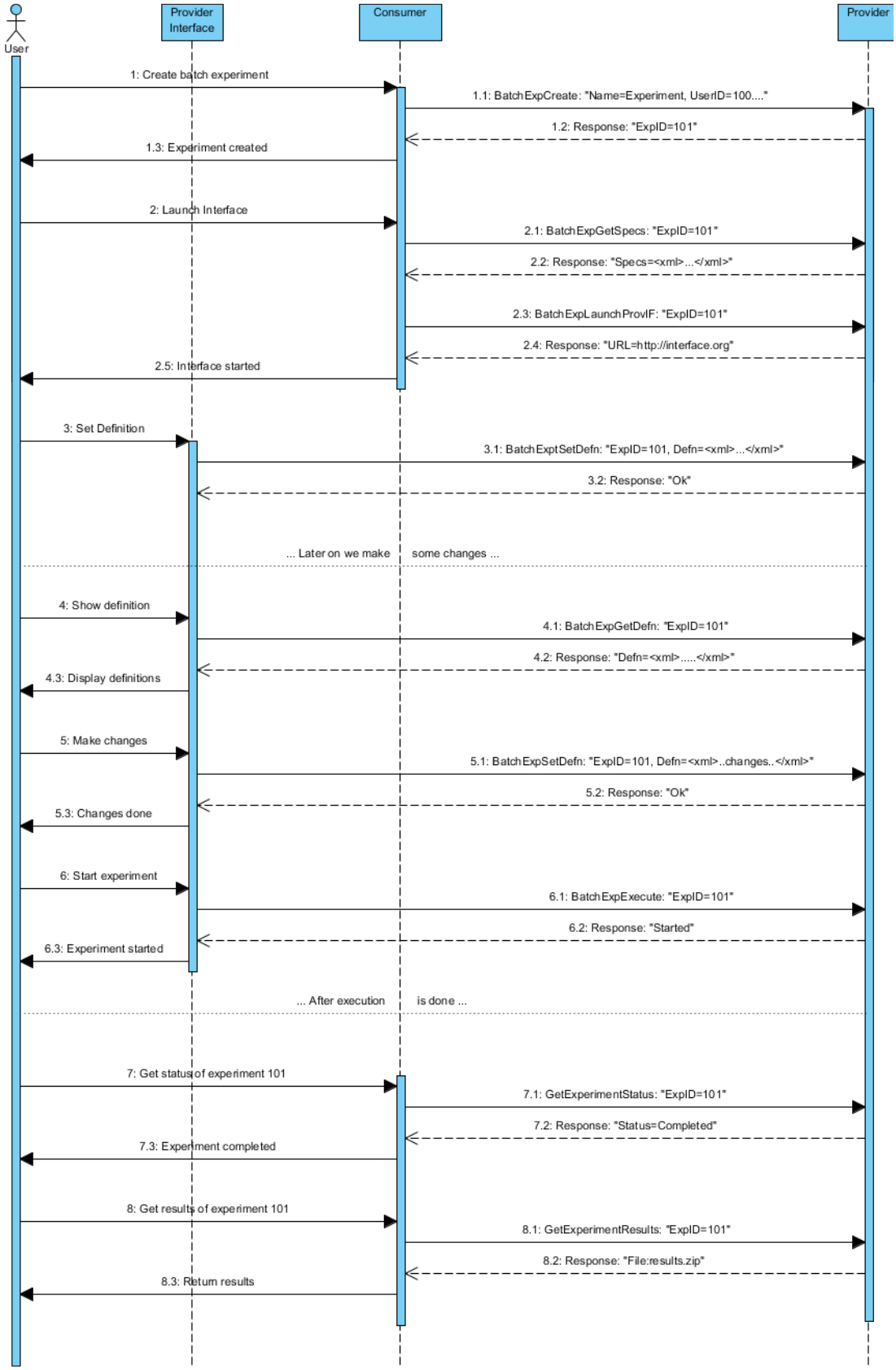


Abbildung 17 Szenario Ausführung eines batchbasierten Experiments

5.6 Basic Interactive Experiment Profile

Das Basic Interactive Experiment Profile entspricht in weiten Teilen den Operationen des Profils für batchbasierte Experimente. Es werden ebenfalls Methoden zum Erzeugen, Lesen, Ändern und Löschen bereitgestellt und Methoden zum Starten von grafischen Oberflächen bereitgestellt. Da interaktive Experimente auch während der Durchführung Eingaben erfordern, wurde dieses Profil um eine Methode zum Senden von Befehlen ergänzt. Diese wird bei der Verwendung eines Interfaces gebraucht, das durch den Consumer bereitgestellt wird. Ebenfalls verfügbar ist eine Methode zum Beenden des Interfaces. Dem Provider wird dadurch mitgeteilt, dass keine weiteren Befehle zu erwarten sind. Der Status des Experiments kann daraufhin zu Completed wechseln.

InteractiveExpCreate

Beschreibung: Erzeugt einen neuen Container zur Durchführung eines Experiments.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	rigSetID	Die ID des Rigs oder RigSets	String	[1..1]
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Gruppe	String	[0..1]
Input	name	Name der Experiment Definition	String	[0..1]
Input	description	Beschreibung der Experiment Definiton	String	[0..1]
Output	expID	ID der erzeugten Experiment Definition	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InteractiveExpRead

Beschreibung: Liest die Informationen einer bereits erzeugten Definition eines Experiments aus.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der erzeugten Experiment Definition	String	[1..1]
Output	rigSetID	Die ID des Rigs oder RigSets	String	[1..1]
Output	userID	Die ID des Nutzers	String	[0..1]
Output	groupID	Die ID der Gruppe	String	[0..1]
Output	name	Name der Experiment Definition	String	[0..1]
Output	description	Beschreibung der Experiment Definiton	String	[0..1]

Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InteractiveExpUpdate

Beschreibung: Operation zum Ändern einer bestehenden Definition eines Experiments.

Bedingungen: Der angegebene Nutzer oder die angegebene Gruppe muss Eigentümer der Definition sein

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der erzeugten Experiment Definition	String	[1..1]
Input	rigSetID	Die ID des Rigs oder RigSets	String	[1..1]
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Gruppe	String	[0..1]
Input	name	Name der Experiment Definition	String	[0..1]
Input	description	Beschreibung der Experiment Definiton	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InteractiveExpDelete

Beschreibung: Löscht eine erzeugte Definition eines Experiments aus dem Speicher des Providers.

Bedingungen: Der angegebene Nutzer oder die angegebene Gruppe muss Eigentümer der Definition sein.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Input	userID	Die ID des Nutzers	String	[0..1]
Input	groupID	Die ID der Gruppe	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InteractiveExpGetSpecs

Beschreibung: Liefert eine XML-Datei mit Details zur Erzeugung eines Interfaces für das Experiment.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	rigSetID	Die ID des Rigs oder RigSets	String	[1..1]
Output	Specs	Details zur Interaktion in XML-	XML-	[0..1]

		Datei	String	
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InteractiveExpLaunchConsIF

Beschreibung: Fordert die Ausführung eines interaktiven Experiments an unter Verwendung einer grafischen Oberfläche des Consumers

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InteractiveExpSendCommand

Beschreibung: Sendet eine XML-Datei mit Befehlen zur Ausführung eines interaktiven Experiments

Bedingungen: Das Experiment befindet sich in der Ausführung.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Input	execInfos	Auszuführende Befehle in XML-Datei	XML-String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InterctiveExpEndConsIF

Beschreibung: Beendet ein laufendes, interaktives Experiment.

Bedingungen: Das Experiment befindet sich in der Ausführung.

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

InteractiveExpLaunchProvIF

Beschreibung: Liefert eine URL zum Starten einer Benutzeroberfläche, die durch den Provider bereitgestellt wird.

Bedingungen: Keine

Input/Output	Name	Beschreibung	Datentyp	Kardinalität
Input	expID	ID der Experiment Definition	String	[1..1]
Output	uiURL	Die URL der Benutzeroberfläche	String	[0..1]
Output	responseCode	Der zurück gelieferte Antwortcode	Integer	[1..1]
Output	errorMessage	Visuell lesbare Fehlermeldung	String	[0..1]

5.6.2 Szenario

Durchführung eines interaktiven Experiments:

Der Nutzer erzeugt in (1) ein neues interaktives Experiment und reserviert in (2) ein Rig für die Durchführung. In (3) möchte der Nutzer das Interface starten, worauf hin sich der Consumer die Spezifikationen holt und das Interface generiert. Über das Interface kann der Nutzer nun in (4) Befehle an das Rig senden und somit die Ausführung des Experiments steuern. Nachdem die Durchführung vollständig ist, beendet der Nutzer in (5) das Interface.

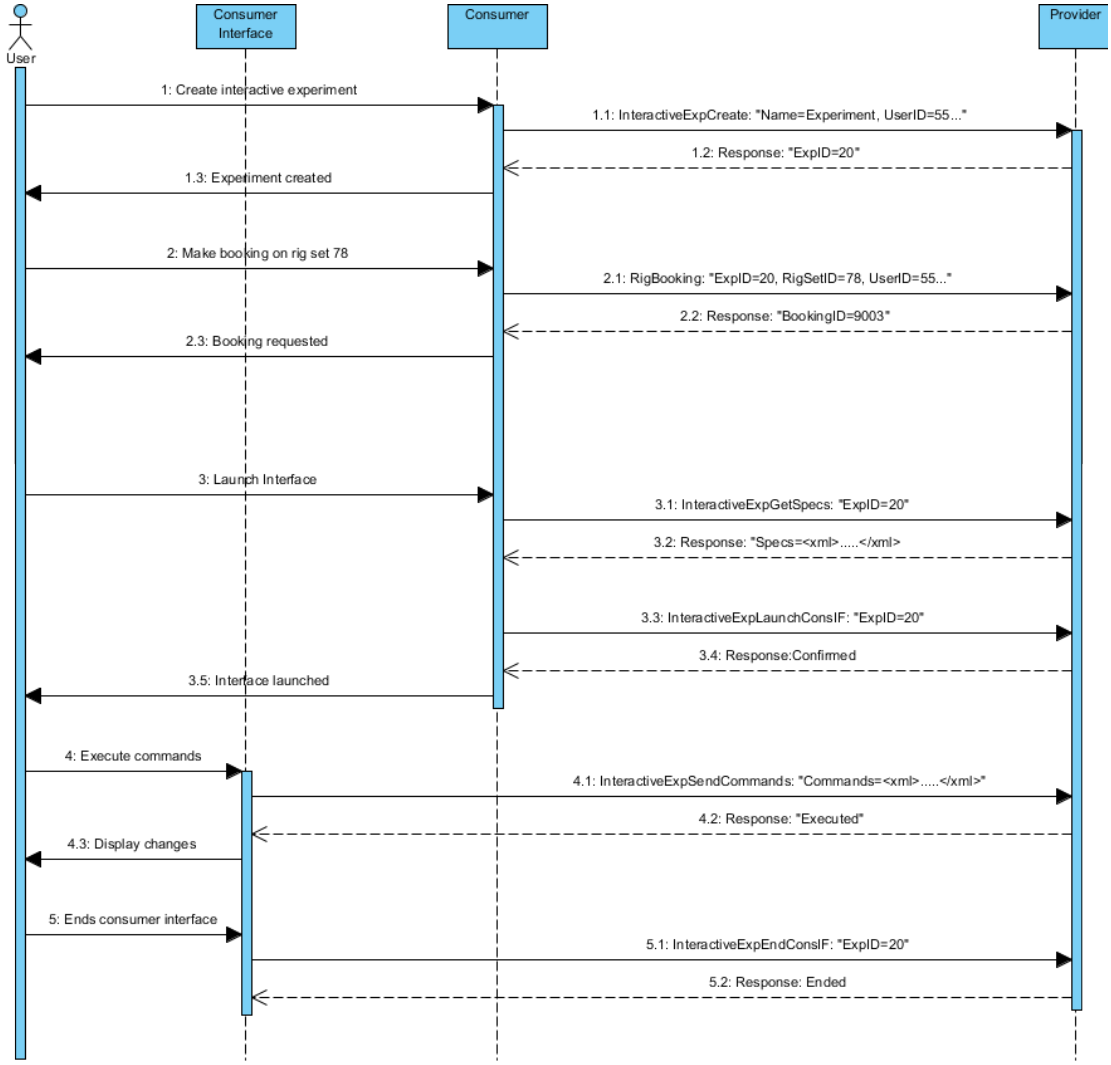


Abbildung 18 Szenario: Ausführung eines interaktiven Experiments

4.7 Definition der Antwortcodes

Der entwickelte Standard definiert einen verbindlichen Code, welcher in jeder Antwort enthalten sein muss und eine optionale Fehlermeldung. Der Code dient zur Unterscheidung sowohl von korrekten als auch von fehlerhaften Ausführungen der Operationen und wird vom System des Consumers verwendet. Das System hat so die Möglichkeit, zwischen temporären und dauerhaften Fehlern, fehlerhaften bzw. unvollständigen Eingaben oder Fehlern innerhalb der Dienste zu unterscheiden und dem Nutzer dadurch einen möglichen Lösungsansatz anzubieten. Im Falle einer fehlerhaften Ausführung enthält die Fehlermeldung eine visuell lesbare Nachricht welche auf dem Bildschirm angezeigt wird und den Nutzer über die Fehlergründe informiert.

Zunächst wurde nur ein optionaler Code für den Fall einer fehlerhaften Operation definiert. Dieser hatte zwei bedeutende Nachteile. Es mussten im Fehlerfall zuerst die Ergebnisse auf Vollständigkeit und Korrektheit überprüft werden, daraufhin der Code selbst und zuletzt die Fehlermeldung abgefragt werden. Der verbindliche Code vereinfacht dieses Vorgehen, in dem zuerst der Code überprüft wird und nur im Fall einer korrekten Operation die Ergebnisse verarbeitet werden. Des Weiteren ist es schwierig, die Daten auf Korrektheit und Vollständigkeit zu überprüfen, da Parameter in bestimmten Operationen optional sein können, eine Operation über keine Antwortparameter verfügt oder Parameter zurückgegeben werden, die keinen Fehlerzustand abbilden können. Liefert eine Funktion zum Beispiel einen Parameter vom Typ boolean zurück, kann dieser in Java nicht den Wert null annehmen und somit nur die zwei Zustände false und true abbilden, welche sowohl im Fehlerfall, als auch im Fall einer korrekten Ausführung möglich sind. Eine korrekte Überprüfung der Ergebnisse ist dadurch nicht möglich. Der zweite bedeutende Nachteil ist, dass der optionale Code nicht zwischen verschiedenen Fällen einer korrekt ausgeführten Operation unterscheiden kann. Die Laufzeit eines Batch-Experiments kann mehrere Wochen andauern. Während dieser Zeit ist es einem Nutzer möglich, partielle Ergebnisse über die Methode GetExperimentResults anzufordern. Sowohl der Nutzer, als auch der Consumer müssen eine Möglichkeit haben, diesen Fall zu unterscheiden.

Die Codes sind in fünf Gruppen mit unterschiedlichen Startnummern unterteilt, wobei die erste Gruppe Codes für den Fall einer korrekten Ausführung definiert und die Gruppen zwei bis fünf Codes für eine fehlerhafte Ausführung definieren. Dabei wurde die Gruppierung der Fehlercodes logisch zusammenhängend, nach dem Hintergrund der Operation vorgenommen, beispielsweise Fehler die bei Operationen auftreten, die ein Rig betreffen. Eine Liste der definierten Codes ist in den folgenden Unterkapiteln dargestellt.

5.7.1 Erfolgreiche Operationen

Codes mit einer 1 als Startnummer stehen für korrekt ausgeführte Operationen. Version 1.0 des Standards definiert hier mögliche Antworten. Code 100 ist der allgemeine Code für eine korrekt ausgeführte Operation, welche korrekte und vollständige Ergebnisse liefert. Code 101 definiert den bereits beschriebenen Fall von partiellen Ergebnissen.

```
Code:  100
Name:  OK
Bedeutung:  Die Operation wurde korrekt ausgeführt und liefert korrekte und
            vollständige Ergebnisse
Verwendung:  Alle operationen


Code:         101
Name:         Ok, partial results
Bedeutung:    Die Operation wurde korrekt ausgeführt, es werden jedoch nur
              Teilergebnisse angezeigt.
Verwendung:   GetExperimentResults
```

Listing 1: Erfolgreiche Operationen

5.7.2 Generelle Fehler

Diese Codes beschreiben allgemeine Fehler, welche in allen Operationen auftreten können. Hierzu gehören Fehler auf Seite des Providers und Fehler der Dienste. Code 251 definiert einen Fehler welcher verwendet wird, wenn kein anderer Fehler geeignet ist.

```
Code:         200
Name:         Provider Error
Bedeutung:    Gibt einen Fehler auf Seite des Providers an. Die Operation
              wurde nicht ausgeführt.
Verwendung:   Alle Operationen


Code:         201
Name:         Temporary Unavailable
Bedeutung:    Informiert den Client, dass die angeforderte Operation zur Zeit
              nicht verfügbar ist. Die Operation wurde nicht ausgeführt.
Verwendung:   Alle Operationen


Code:         250
Name:         Service Error
Bedeutung:    Beschreibt einen Fehler innerhalb der Services des Inter-
              operability Standards. Die Ausführung der Operation wurde
              abgebrochen.
Verwendung:   Alle Operationen


Code:         251
Name:         Unknown Exception
Bedeutung:    Wird als Fehlermeldung zurückgegeben, falls kein definierter
              Code geeignet ist.
Verwendung:   Alle Operationen
```

Listing 2: Allgemeine Fehler

5.7.3 Fehler bei Experimenten

Diese Codes definieren Fehler, die bei Abfragen bezüglich Experimenten auftreten können. Ein Teil dieser Fehler orientiert sich an Fehlern der Datenhaltung, beispielsweise, dass ein Experiment nicht existiert oder nicht angelegt werden kann. Andere Codes definieren Fehler, die für die Operationen des Standards spezifisch sind, etwa dass keine Spezifikation für Interfaces gefunden werden konnte.

```
Code:         300
Name:         Experiment Does Not Exist
Bedeutung:    Impliziert, dass kein Experiment mit der spezifizierten ID
              gefunden werden konnte. Es wurden keine Aktionen vorgenommen.
Verwendung:   BatchExpDelete,        BatchExpExecute,        BatchExpGetDefn,
              BatchExpGetSpecs,     BatchExpLaunchProvIF,     BatchExpSetDefn,
              InteractiveExpDelete,                    InteractiveExpGetSpecs,
              InteractiveExpLaunchConsIF,          InteractiveExpLaunchProvIF,
              InteractiveExpSendCommand,    RigQueue,    GetExperimentResults,
              GetExperimentStatus


Code:         301
Name:         Experiment Already Exists
Bedeutung:    Wird zurückgegeben, falls eine Definition eines Experiments mit
              identischen Parametern bereits vorhanden ist. In diesem Fall
              liefert die Operation die existierende ID zurück.
Verwendung:   BatchExpCreate, InteractiveExpCreate


Code:         302
Name:         Input Data Missing
Bedeutung:    Gibt an, dass die Eingabeparameter nicht vollständig sind und
              die   Operation   nicht   ausgeführt   werden   kann.   Fehlende
              Eingbeparameter können auftreten, falls ein oder mehrere
              Parameter nicht spezifiziert wurden oder falls Parameter für
              die Ausführung eines Experiments fehlen.
Verwendung:   BatchExpExecute,          RigBooking,          RigBookingsQuery,
              GetAllActiveExperiments, GetExperimentIDsWithStatus, GetRigs


Code:         303
Name:         No Definition Found
Bedeutung:    Informiert den Client, dass keine Definition für das gegebene
              Experiment gefunden werden konnte.
Verwendung:   BatchExpGetDefn


Code:         304
Name:         No Specification Found
Bedeutung:    Informiert   den   Client,   dass   keine   Spezifikation   für   die
              Ausführung des Interfaces des eingegebenen Experiments gefunden
              werden konnte.
Verwendung:   BatchExpGetSpecs, InteractiveExpGetSpecs


Code:         305
Name:         No Interface Found
Bedeutung:    Impliziert, dass kein Interface für das eingegebene Experiment
              gefunden werden konnte.
Verwendung:   BatchExpLaunchProvIF, InteractiveExpLaunchProvIF
```

```
Code:         306
Name:         Incorrect Commands
Bedeutung:    Wird benutzt um den Nutzer zu informieren, dass die
              eingegebenen Befehle für die Ausführung eines Experiments
              fehlerhaft waren.
Verwendung:   InteractiveExpSendCommand


Code:         307
Name:         No Results Available
Bedeutung:    Wird zurückgegeben falls keine Experimente für das
              spezifizierte Experiment vorhanden sind. Experimente können
              jedoch zu einem späteren Zeitpunkt verfügbar sein.
Verwendung:   GetExperimentResults


Code:         308
Name:         Experiment Not Queueable
Bedeutung:    Dieser Code wird benutzt, um den Client zu informieren, dass
              das eingegebene Experiment nicht in eine Warteschlange
              eingefügt werden kann.
Verwendung:   RigQueue


Code:         309
Name:         Unknown Status
Bedeutung:    Wird zurückgegeben, falls ein Status angegeben wurde, der nicht
              definiert ist.
Verwendung:   GetExperimentIDsWithStatus


Code:         310
Name:         Missing Rigts
Bedeutung:    Impliziert, dass der Nutzer nicht über die nötigen Rechte
              verfügt, um diese Operation durchzuführen
Verwendung:   Alle Operationen der Basic*-Profile
```

Listing 3: Fehler im Zusammenhand mit Experimenten

5.7.4 Fehler im Zusammenhang mit Rigs

Diese Sammlung an Codes definiert Fehler, die bei Abfragen bezüglich Rigs auftreten können. Dazu zählen sowohl Fehler die im Rahmen einer Buchung auftreten können als auch beim Einfügen in eine Warteschlange. Zusätzlich werden Fehler definiert für Anfragen mit inkorrekten Angaben.

```
Code:         400
Name:         Rig Does Not Exist
Bedeutung:    Wird genutzt um dem Nutzer mitzuteilen, dass das spezifizierte
              Rig nicht existiert.
Verwendung:   RigAccessType,    RigAvailability,    GetAllActiveExperiments,
              GetExperimentIDsWithStatus, GetRigInfo, GetRigStatus, GetRigs,
              GetAllServiceLevels, GetServiceLevel

Code:         401
Name:         Rig Not Bookable
```

```
Bedeutung:    Indiziert, dass das angegebene Rig nicht gebucht werden kann.
Verwendung:   RigBooking

Code:         402
Name:         Rig Not Queueable
Bedeutung:    Gibt an, dass auf dem angegebenen Rig keine Warteschlange
              verfügbar ist.
Verwendung:   RigQueue

Code:         403
Name:         Rig Not Available
Bedeutung:    Wird verwendet um den Nutzer zu informieren, dass auf dem
              angegebenen Rig, zur spezifizierten Zeit keine Reservierung
              vorgenommen werden kann.
Verwendung:   RigBooking

Code:         404
Name:         No Informations Available
Bedeutung:    Teilt dem Nutzer mit, dass für das angegebene Rig keine
              Informationen verfügbar sind.
Verwendung:   GetRigInfo

Code:         405
Name:         Booking Does Not Exist
Bedeutung:    Wird zurückgegeben, wenn der Nutzer eine nicht existierende
              Buchungs ID angibt.
Verwendung:   RigBookingCancel, RigBookingStatus
```

Listing 4: Fehler im Zusammenhang mit Rigs

5.7.5 Fehler bezüglich Providerabfragen

Diese Codes definieren Fehler bei Abfragen bezüglich der Leistungen eines Providers. Zu zählen die Profile und Servicelevel. Der Fehlercode ProfileNotSuppported sollte verwendet werden, wenn keine der Operationen eines Profils unterstützt werden. Sollten nur Teile des Profils nicht unterstützt werden, ist der Fehlercode OperationNotSupported zu verwenden.

```
Code:         500
Name:         Profile Does Not Exist
Bedeutung:    Wird benutzt um anzuzeigen, dass ein nicht existierender
              Profilname angegeben wurde.
Verwendung: GetProfileVersion


Code:         501
Name:         Profil Not Supported
Bedeutung:    Indiziert, dass keine der Operationen des Profils unterstützt
              werden.
Verwendung:   Alle Operationen


Code:         502
Name:         Operation Not Supported
Bedeutung:    Indiziert, dass diese Operation nicht unterstützt wird.
```

```
Verwendung:   Alle Operationen


Code:         503
Name:         Service Attribute Does Not Exist
Bedeutung:    Wird zurückgegeben, falls der Nutzer ein Servicelevel abfragen
              will, das nicht spezifiziert ist.
Verwendung:   GetServiceLevel
```

Listing 5: Fehler bezüglich Providerabfragen

5.8 Definierte Servicelevel

Dieses Unterkapitel erläutert die definierten Servicelevel. Provider erhalten so die Möglichkeit, nicht-funktionale Anforderungen zu spezifizieren, die ein Nutzer abfragen kann. Servicelevel können sowohl für einzelne Rigs oder RigSets angegeben werden, als auch für die gesamten Labore eines Providers. PaymentInformations wird zum jetzigen Zeitpunkt vermutlich nicht gebrauch, kann aber in späteren Versionen eine Rolle spielen.

ServiceAvailability

Liefert eine Aussage des Providers über die garantierte Verfügbarkeit angebotener Versuchsaufbauten und Hardware.

ServiceReliability

Trifft eine Aussage über die Zuverlässigkeit der angebotenen Versuchsaufbauten und Hardware. Behandelt insbesondere auch den Fall eines aufgetretenen Fehlers und auf welche Weise das Experiment beendet oder neugestartet wird.

Responsible Person

Spezifiziert eine Person oder eine Institution, welche für die Instandhaltung und Wartung eines bestimmten Aufbaus oder RigSet verantwortlich ist. Die Aussage beinhaltet Informationen zur Kontaktaufnahme für Rückfragen und Probleme. Sollte bei der Anfrage kein RigSet spezifiziert sein, werden die Kontaktdaten der für den Gesamtbetrieb verantwortlichen Person oder Institution des Providers gesendet.

InformationSecurity

Aussage des Providers über den Grad der Informationssicherheit, wie etwa den Zugriff auf Versuchsergebnisse durch Dritte und ob die Ergebnisse und Informationen vorher anonymisiert werden.

DataQuality

Liefert eine Aussage über die Qualität der gemessenen Daten und Ergebnisse eines Experiments generell, oder eines RigSet im Speziellen.

ExperimentRetention

Gibt an, wie lange die Ergebnisse eines Experiments nach dessen Durchführung verfügbar sind und wie oft die Ergebnisse abgerufen werden können, bevor Sie gelöscht werden.

ExperimentProvenance

Trifft eine Aussage über die Verfolgung der Herkunft von gemessenen und erzeugten Daten. Dazu zählen verwendete Hard- und Software, Sensoren, welche Berechnungen fanden statt und welche Daten wurden während der Durchführung ausgetauscht und bearbeitet.

HardwareSpecification

Allgemeine Aussage über die bereitgestellte Hardware im Ganzen oder die Spezifikation der Hardware für ein definiertes Rig. Zur Reproduktion von Ergebnissen eines Experiments ist oftmals die identische Hardwarekonfiguration nötig, welche sich bei mehreren Rigs eines RigSet unterscheiden kann.

PaymentInformations

Liefert eine Aussage über die Preise und Zahlungsbedingungen für die Nutzung eines Rigs oder der Dienste eines Providers. Diese Informationen mögen in der Zukunft wichtig sein, wenn externen Firmen der Zugriff auf Ressourcen eines Providers gestattet werden soll, oder für Experimente in denen teure Materialien verbraucht werden.

5.9 Spezifikation eines Consumer-Interfaces

Sowohl für interaktive, als auch für Batch-Experimente, soll ein Consumer die Möglichkeit erhalten, eigene, grafische Oberflächen zu entwickeln und seinen

Nutzern zur Verfügung zu stellen. Diese können entweder als separates, ständig verfügbares Interface entwickelt werden oder als generische Oberfläche zum Zeitpunkt der Anforderung durch den Nutzer erzeugt werden. Consumer müssen somit die verwendeten Elemente der Oberfläche kennen, bzw. welche Ein- und Ausgaben bei der Durchführung eines Experiments wichtig sind. Diese Elemente können über die Operationen BatchExperimentGetSpecs und Interactive ExperimentGetSpecs (siehe 4.5/ 4.6) von dem jeweiligen Provider des Experiments angefordert werden. Die zurück gelieferten Spezifikationen müssen allgemeiner Natur sein und können sich nicht an bestimmten Programmiersprachen oder Werkzeugen orientieren. Der hohe Grad an Unabhängigkeit, welcher durch den Einsatz von Webservices gegeben ist, muss auch für die Spezifikation der Elemente gelten. So ist ein Consumer bei der Entwicklung der Oberflächen flexibel und kann auf Sprachen wie PHP, C# oder Java zurückgreifen. Ergänzend zu Ein- und Ausgabeparametern soll die Möglichkeit bestehen, wichtige Informationen in Textform zu präsentieren. Im Rahmen der Diplomarbeit wurde eine Sammlung an Basiselementen definiert welche im Folgenden dargestellt sind.

<description>-Element

Dieses Element wird benutzt, um dem Provider die Möglichkeit zu bieten, Beschreibungen und Details zur Durchführung des Experiments anzugeben. Das Element verfügt über zwei Attribute, headline, zur Angabe eines Titels und description, zur Angabe der eigentlichen Nachricht. Nachrichten müssen visuell lesbar sein.

<boolean>-Element

Definiert entweder eine Zustandsanzeige oder eine Überprüfung. Eine Zustandsanzeige kann zum Beispiel bei Ausgabeparametern wichtig sein, um den Erfolg eines Experiments anzuzeigen. Eine Überprüfung kann für die Einstellungen des Experiments verwendet werden, beispielsweise um zu erfragen, ob ein bestimmter Zusatzparameter benutzt werden soll. Das Element verfügt über drei Attribute

<integer>- und <double>-Elemente

Diese Elemente sind für Eingabeparameter vom Typ Integer, bzw. Double, bestimmt. Zusätzlich zu den Attributen `title` und `description` kann hier ein minimaler und maximaler Wert definiert werden. Die Attribute dazu werden als `minValue` und `maxValue` bezeichnet. Der Provider hat so die Möglichkeit, den Wertebereich der Parameter einzuschränken. Optional kann ein Attribut `default` angegeben werden, welches einen voreingestellten Wert repräsentiert.

<string>-Element

Elemente vom Typ String können sowohl für Ein-, als auch für Ausgabeparameter verwendet werden. Bei der Erstellung des Interfaces muss berücksichtigt werden, dass

die Zeichenlänge nicht beschränkt ist und der Text somit über mehrere Zeilen gehen kann. Das <string>-Element verwendet ebenfalls die Attribute `title` und `description`, sowie ein Attribut `value`. Dient das Element zur Eingabe ist value optional, im Falle eines Ausgabeparameters muss hier ein Text vorhanden sein.

<choice>-Element

Dieses Element dient zur Auswahl mehrerer Optionen. Das Element besitzt die Attribute title und `description`, sowie ein Attribut `default` zur Bestimmung eines vorgegebenen Wertes. Die Auswahlmöglichkeiten können nicht in den Attributen erfasst werden, weshalb ein neues Element mit dem Tag <option> definiert wurde. Sollte das default-Attribut verwendet werden, muss der Wert dem Titel eines der vorhandenen option-Elemente entsprechen.

<option>-element

Dieses Element ist ein Unterelement von <choice> und definiert eine Auswahlmöglichkeit innerhalb der <choice>-Gruppe. Als Attribute können hier ebenfalls ein Titel und eine Beschreibung angegeben werden.

Dateiaufbau

Hier wird der Aufbau einer zu übertragenden Spezifikation definiert. Eine Spezifikation ist in einer einzelnen Datei in XML zu verfassen und muss ein wohlgeformtes und gültiges XML-Dokument, in der XML-Version 1.0, dargestellen. Als Kodierung wird UTF-8 vorausgesetzt. Einzelne Elemente werden in einem Tag definiert, das dem Namen des Elements entsprich, die Attribute entsprechend den bereits genannten Definitionen. Elemente zur Ausgabe von Beschreibungen werden von einem description-Tag umschlossen, Elemente zur Eingabe von Parametern werden innerhalb eines input-Tags gruppiert und Elemente zur Ausgabe entsprechend in einem output-Tag. Sowohl description- als auch input- und output-Tag werden von einem elements-Tag umschlossen, welches die verwendeten Elemente beschreibt. Ein Beispiel einer korrekten Datei zeigt das folgende Listing 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
    <descriptions>
        <message headline="Requirements" description="…"/>
        <message headline="Results" description="…"/>
    </descriptions>
    <elements>
        <input>
            <integer title="…" description="…" minValue="…" maxValue="…"/>
            <choice title="Choice of used sensor" default="Sensor A">
                <option title="Sensor A" description="…"/>
                <option title="Sensor B" description="…"/>
            </choice>
            ……
         </input>
```

```
        <output>
            <string title="…" description="…" default="…"/>
            ……
        </output>
    </elements>
</interface>
```

Listing 6: Beispiel einer Spezifikation zur Definition der Elemente eines Interfaces

6. Implementierung

Dieses Kapitel widmet sich der Umsetzung der erarbeiteten Konzepte. Im Rahmen der Diplomarbeit wurde die Komponente zur Kommunikation mit anderen RLS für das Projekt LiLa der Universität Stuttgart implementiert. Die Komponente implementiert auf Serverseite die im vorhergehenden Kapitel beschriebenen Profile als Webservices. Ergänzt werden die Services um eine API, welche die Clientseite

implementiert und Methoden zum Aufrufen der Services bereitstellt. Die Services bauen auf dem vorhandenen Buchungssystem und der API der LiLa Ontology auf. Vor der eigentlichen Implementierung musste zunächst das Datenmodell erweitert und die Services der vorhandenen Komponenten ergänzt werden. Das Ende des Kapitels beschreibt die Konfiguration und Evalutaion der implementierten Komponente.

6.1 Änderungen am Datenmodell

Dieses Unterkapitel beschreibt die Änderungen am bestehenden Datenmodell. Auch wenn die existierenden RLS bereits über ein standardisiertes Datenmodell verfügen, sind Änderungen nötig, um die Funktionalität der neuen Komponente für externe Nutzer zu gewährleisten. Die Änderungen beinhalten jedoch nur Erweiterungen und neue Entitäten, wodurch bestehende Komponenten des RLS nicht betroffen sind. Das Entity-Relationship-Modell in Abbildung 19 zeigt Teile des vorhandenen Datenmodells und vorgenommene Änderungen.

6.1.1 Externer Identifikator

Um Entitäten nach außen hin verfügbar zu machen, müssen wir zunächst vier der vorhandenen Entitäten um einen externen Identifikator erweitern. Dazu zählen die Entitäten RIG als Repräsentant einer Hardwareinstallation, Reservation als Repräsentant eines Bookings und Agent. Die Entität Agent repräsentiert Nutzer des RLS, welche bisher nicht in Einzelnutzer und Gruppen unterteilt sind. Als vierte Entität erweitern wir das InteractionPackage, welches alle benötigten Informationen zur Durchführung eines Experiments speichert. Der Identifier bekommt den Namen ExternalIdentifier und stellt eine eindeutige Folge von 32 Zeichen dar. Diese wird in der Implementierung benutzt, um einen Universally Unique Identifier (UUID) [30] zu speichern. Dieser Identifier ist ein Standard für Identifikatoren, welcher von der Open Software Foundation mit der Absicht spezifiziert wurde, Informationen in verteilten Systemen ohne zentrale Koordination eindeutig kennzeichnen zu können.

6.1.2 Entität Gruppe

Wie bereits beschrieben verfügt das Datenmodell zum Zeitpunkt der Implementierung nicht über eine klare Trennung zwischen Einzelnutzer und einer Gruppe an Nutzern. Solche Gruppen können beispielsweise Teilnehmer eines Seminars oder einer Vorlesung sein, für die der zugehörige Professor Experimente zu Übungszwecken definiert. Wir benötigen also eine neue Entität namens Group, welche als Sammlung von Entitäten, des Typ Agent spezifiziert ist. Diese speichert eine Liste an Agents, welche als Mitglieder der Gruppe definiert ist und folglich den Namen memberList trägt. Eine zweite Liste namens groupOwner soll die IDs, der verantwortlichen

Personen der Gruppe speichern. Diese sind in der Lage, Änderungen an der Gruppe vorzunehmen und neue Mitglieder hinzuzufügen. Beide Listen wurden als String mit variabler Länge spezifiziert, da Datenbanken keine Listen oder Arrays abbilden können. Die übrigen Attribute einer Group entsprechen den Attributen eines Agents. Um eine logische Trennung zwischen Einzelnutzern und Gruppen zu erreichen, wird die Entität Group nicht von Agent abgeleitet.

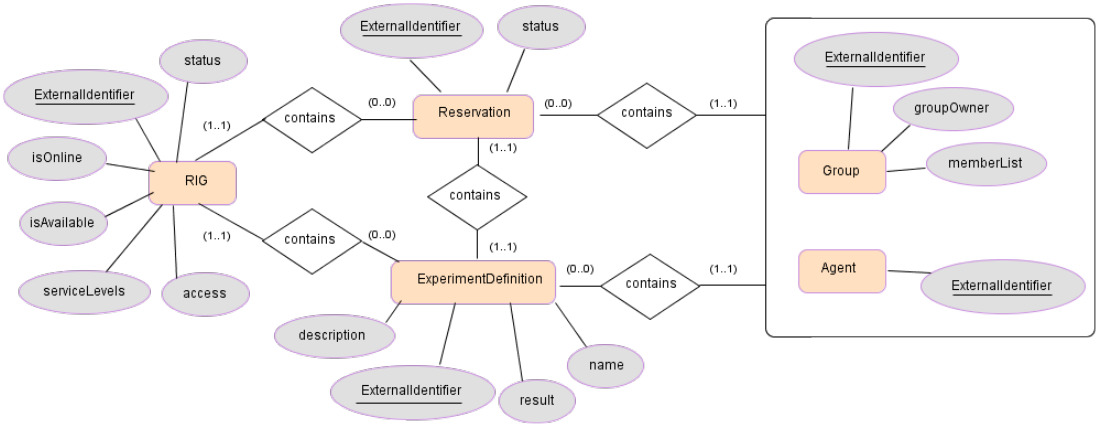


Abbildung 19 ER-Diagramm: Änderungen am LiLa Datenmodell

6.1.4 Entität Reservation

Die Entität Reservation wird, zusätzlich zu ExternalIdentifier, um ein Attribut status erweitert, welches in der Operation GetBookingStatus benötigt wird. Dieses speichert den Status der Reservierung und kann folglich drei Werte annehmen, Pending, Confirmed und Denied. Da eine Reservierung immer über einen Status verfügen muss, darf dieser nie null sein. Der Status wird bei der Erzeugung auf Pending gesetzt. Auf Grund der vorgegebenen Möglichkeiten wurde der Status als String einer festen Länge spezifiziert.

6.1.5 Entität RIG

Die Entität RIG musste ebenfalls um Attribute ergänzt werden. Die bisherige Entität erlaubt keine Bildung einer Hierarchie und das Speichern von untergeordneten RIGs. Somit war es bisher nicht möglich, ein RigSet zu bilden. Um dies zu ermöglichen, benötigen wir zwei neue Listen. Die erste Liste speichert alle untergeordneten RIGs, wodurch das speichernde RIG als Top-Level-Rig eines RigSet fungiert. Die zweite Liste speichert alle übergeordneten RIGs. Dies ist wichtig zur Bestimmung der Top-Level-Rigs und um beide Wege innerhalb der Hierarchie abzubilden. Beide Listen können leer sein und den Wert null annehmen. Sollte ein RIG über keine übergeordneten RIGs verfügen, kann das RIG als Top-Level-Rig des Providers angesehen werden und ist kein Bestandteil eines RigSet. Verfügt es über keine untergeordneten RIGs, so ist es entweder Teil eines RigSet oder ein eigenständiges Rig. Die Listen wurden als String mit variabler Länge spezifiziert.

Da ein Nutzer die Servicelevel eines Anbieters auch für ein spezielles Rig oder RigSet abfragen kann, benötigt die Entität ein Attribut zur Speicherung der Werte der einzelnen Servicelevel. Aus Sicht der Implementierung bietet sich hier eine zweidimensionale Map an, bestehend aus Name und zugehöriger Beschreibung eines Service Levels. Die Abbildung auf das Datenmodell sieht hierzu einen String variabler Länge vor, der im Falle von nicht spezifizierten Beschreibungen auch leer sein kann.

Um den Zugriff auf eine Hardwareinstallation für bestimmte Nutzer oder Nutzergruppen einzuschränken, benötigt die Entität RIG zwei Attribute. Das erste Attribut definiert eine List an IDs der Entitäten Agent und Group und speichert somit alle zugangsberechtigten Nutzer ab. Diese Liste reicht jedoch nicht aus, da auch Installationen ohne Zugriffsbeschränkungen existieren. Verfügt die Liste über keine IDs, so kann nicht eindeutig bestimmt werden ob alle oder keine Nutzer auf das RIG zugreifen dürfen. Wir benötigen also ein weiteres Attribut, namens freeAcces, um diesen Zustand eindeutig zu bestimmen. Somit können wir zunächst abfragen ob der Zugriff eingeschränkt ist und entsprechend die Liste der verifizierten Nutzer auslesen. Die Liste erhält den Namen accessibleBy und wird als String variabler Länge spezifiziert, das Attribut zur Bestimmung des Zugriffs als binärer Datentyp.

Als nächsten wurde RIG um zwei Attribute erweitert, um den Status des RIG zu beschreiben. Die Attribute isOnline und isAvailable spezifizieren, ob ein RIG online und einsatzbereit ist. Beide Werte entsprechen dem Datentyp Boolean und können so als binäre Daten angegeben werden. Sie dürfen jedoch nicht den Wert null annehmen. Ergänzend hierzu definiert das Attribut statusMessage eine visuelle lesbare Nachricht, die eine Beschreibung des Status enthält. Anbieter von Rigs oder Provider können so den Status näher erläutern, beispielsweise ob die Hardwareinstallation gerade gewartet wird und wann sie wieder verfügbar ist.

6.1.6 Entität Experiment Definition

Zur Durchführung von Experimenten wird ein Container benötigt. Der Ablauf sieht wie folgt aus. Ein Professor kann für seine Übungsgruppe ein Experiment definieren. Hierzu wird ein InteractionPackage benötigt, welches alle benötigten Informationen und Details speichert. Ein Teilnehmer der Übungsgruppe kann nun das defininierte InteractionPackage verwenden, um eine Reservierung für ein spezielles Rig vorzunehmen und das Experiment auszuführen. Aufgabe des Containers ist, alle Informationen zu speichern, die nötig sind um das Experiment durchzuführen. Dazu zählen der Nutzer, welcher das Experiment durchführt, die ID des durchgeführten Experiments und das verwendete Rig. Des Weiteren muss der Container eine Möglichkeit bieten, Ergebnisse abzulegen, die während der Durchführung des Experiments entstanden sind. Ergänzend wollen wir dem Studenten die Möglichkeit geben, die Durchführung des Experiments mit einem Namen und einer Beschreibung zu versehen. Dies kann nützlich sein, falls der Nutzer ein identisches Experiment mit

unterschiedlichen Parametern oder Versuchsabläufen ausführen möchte. Der benötigte Container wird durch die Entität Eperiment Definition definiert, welche zunächst einen Identifier benötigt. Dieser orientiert sich an dem bereits verwendeten Attribut ExternalIdentifier. Das Attribut interactionPackage speichert die ID des verwendeten Experiments, owner speichert die ID des Nutzers und rig speichert die ID der verwendeten Hardwareinstallation. Alle Identifikatoren, ebenso der Name und die Beschreibung der Durchführung, wurden als String spezifiziert. Während die angegebenen Identifikatoren vorhanden sein müssen, können Name und Beschreibung leer sein. Das Attribut result speichert die Ergebnisse der Experimentdurchführung, welche zum jetzigen Zeitpunkt in einer einzelnen Datei vorhanden sein müssen. Sollten mehrere Ergebnisse vorhanden sein, müssen diese zu einer einzelnen Datei komprimiert werden. Da keine Maximalgröße für erzeugte Dateien vorgegeben ist, wird in der Entität eine Adresse zur erzeugten Datei gespeichert, welche als String definiert ist.

6.1.7 Batch-Experimente und Queues

Warteschlangen und Batch-Experimente werden derzeit nicht durch das Projekt LiLa der Universität Stuttgart unterstützt und sind somit nicht Bestandteil der Implementierung. Eine Entität für Batch-Experimente müsste sich an einem InteractionPackage orientieren und die Informationen zur Durchführung von Experimenten speichern. Da bei interaktiven Experimenten keine Programme gespeichert werden müssen, sollte die Entität für Batch-Experimente diese Erweitern um die Möglichkeit einer Speicherung der auszuführenden Programme und Startparameter. Zur Umsetzung der Warteschlangen müsste eine neue Softwarekomponente in das System integriert werden die Warteschlangen bereitstellt. Diese müssten in eigenen Prozessen laufen und könnten in den Entitäten nur verlinkt werden. Des Weiteren wäre eine umfangreiche Verwaltung der Warteschlangen erforderlich.

6.2 Erweiterung der Services zur Datenhaltung

Durch die Erweiterung des Datenmodells müssen die zugehörigen Services des Buchungssystems und der Verwaltungssoftware des Datenmodells erweitert werden. Diese stellen Methoden zur Verwaltung der Entitäten in der Datenbank bereit und verfügen über Methoden für Queries auf den gespeicherten Daten. Beide Komponenten sind in Java implementiert und verwenden Spring [32] als Framework.

Für die Persistenz der Daten ist Hibernate als Teil des Spring Frameworks verantwortlich.

Als erstes mussten neue Services für die zwei neuen Entitäten Group und Experiment Definition geschrieben werden. Das Basisset dieser Services sind Operationen zur Implementierung der CRUD-Methoden. Die Create-Methode erzeugt dabei ein neues Element und die Delete-Methode löscht ein Element. Beide führen jedoch noch keine Änderung am Datenbestand durch. Dies passiert durch die Update-Methode. Erweitert wurden die Services um Methoden zur Umsetzung der Operationen des Standards. Beispiele hierfür sind Abfragen auf dem Datenbestand um alle Experimente eines bestimmten Nutzers zu finden oder Experimente mit einem bestimmten Status. Es mussten jedoch nicht nur neue Services geschrieben werden, sondern auch vorhandene geändert. Abbildung 19 verdeutlicht, dass viele neue Attribute zum bestehenden Datenmodell hinzugefügt wurden. Diese Änderungen spiegeln sich in den bereits vorhandenen Services wieder. So mussten Abfragen implementiert werden, zur Suche nach Top-Level-Rigs oder für alle Rigs auf denen ein bestimmter Nutzer Zugriffsrechte hat. Auf eine Nennung aller Änderungen wird verzichtet, da diese wenig neue Informationen liefern würde.

```
public class GroupServiceImpl implements GroupService {

    private HibernateTemplate hibernateTemplate;

    public final void setSessionFactory(final SessionFactory sessionFactory) {
        this.hibernateTemplate = new HibernateTemplate(sessionFactory);
    }

    @Override
    public final Group createGroup(final Agent owner) {
        return new Group(owner, URLPREFIX + UUID.randomUUID());
    }

    @Override
    public final Group readGroup(final String identifier) throws GroupReadException {
        Group group = hibernateTemplate.get(Group.class, identifier);
        return group;
    }

    @Override
    public final void updateGroup(final Group group) throws GroupUpdateException {
        hibernateTemplate.saveOrUpdate(group);
    }

    @Override
    public final void deleteGroup(final Group group) {
        hibernateTemplate.delete(group);
    }
```

```
}
```

Listing 7: GroupService mit Hibernate Aufrufen

6.3 Servicedefinitionen

Namespaces:

Interoperabilität: http://online-lab.org/InteroperabilityStandard
Services: http://online-lab.org/InteroperabilityStandard/Services
Profile: http://online-lab.org/InteroperabilityStandard/Services/<Profilname>
Parameter: http://online-lab.org/InteroperabilityStandard/Services/Parameters/<Profilname>

Die Beschreibungen der Schnittstellen sind in WSDL verfasst, wobei für jedes entwickelte Profil ein eigener Service definiert wird. Für das Binding zum Nachrichtenprotokoll wurde Document als Style ausgewählt, da dieser eine umfangreichere Definition der Parameter mit XSD erlaubt [33]. Als Kodierung wurde literal gewählt, da encoded nicht konform ist zu den Spezifikationen der WS-Interoperability. Die Definitionen der Parameter sind in XSD spezifiziert und in separaten Dateien untergebracht. Die WS-Interoperability Organization gibt vor, dass für jede Ein- und Ausgabenachricht, unter Verwendung von document/literal, maximal ein Parameter verwendet werden soll [34]. Diese Anforderung wurde direkt umgesetzt, da sich so auch die Definitionen der Nachrichten von denen der Parameter klar trennen lassen. Anhang A1 zeigt beispielhaft die Nachrichten des Basic Rig Access Profile mit zugehörigen Elementdefinitionen.

Die in Kapitel 5 definierten Kardinalitäten wurden über die Attribute der XSD-Elemente realisiert. Verwendet wurden die Attribute minOccurs, max Occurs und nillable wie folgt:

Kardinalität	minOccurs	maxOccurs	nillable
[0..1]	0	1	true
[0..1]	0	1	true
[1..1]	1	1	true
[0..*]	0	unbounded	true

6.4 Implementierung der Softwarekomponente

Sowohl die implementierten Services, als auch die Client API, wurden als Maven-Projekte entwickelt, um eine einfache Wartung zu gewährleisten. Maven ist für das Einbinden der Projektabhängigkeiten und für das Erstellen einer Web-Archiv Datei

zuständig. In der Projektdatei von Apache Maven [IM02] werden neben den Bibliotheken von Java das Spring Framework, sowie die Komponenten des Projekt LiLa eingebunden, welche über ein eigenes Repository beziehbar sind. Auf die Verwendung von generierenden Tools wie wsgen wurde bewusst verzichtet. Eine erste Generierung verursachte Fehler und war in den Parameterklassen inkonsistent. Es wurden XML- und JAXB-Elemente vermischt und Parameter zum Dateitransfer, von XSD nach Java, fehlerhaft umgewandelt. Die Paketstrukturen orientieren sich an den Namespaces, die in den WSDL-Dateien spezifiziert wurden (siehe 6.3). Maven erlaubt den Aufbau von Komponenten als modulare Pakete. So können Client API und Services in einem Projekt gruppiert werden.

Zunächst wurde ein Prototyp als Referenz entwickelt, der ohne die Aufrufe der Datenbankverwaltung und des Buchungssystems implementiert wurde. Dieser kann von Betreibern anderer RLS verwendet werden und muss nur um die Aufrufe der eigenen Verwaltungssysteme ergänzt werden. Für die Referenzimplementierung wurde bewusst weitestgehend auf die Verwendung von Frameworks oder Werkzeugen verzichtet. Die Implementierung der Webservices fand unter dem Einsatz von Standardbibliotheken der Programmiersprache Java statt. Als einziges Framework wurde Metro in der Version 2.0 verwendet um den Dateitransfer mit MTOM zu implementieren. Aufbauend auf dem Prototyp wurden die Webservices für das Projekt Lila implementiert und anschließen die Client API. Beide Teile sind in den folgenden zwei Unterkapiteln beschrieben.

6.4.1 Services

Dieses Unterkapitel beschreibt die Implementierung der Webservices und die Algorithmen der Operationen. Wie in Kapitel 6 beschrieben, wurde jedes der spezifizierten Profile als eigenständiger Webservice mit multiplen Operationen implementiert. Für jeden Ein- und Ausgabeparameter einer WSDL-Operation wurde eine Parameterklasse implementiert, welche die verwendeten Attribute und Methoden zum Schreiben und Lesen der Attribtue enthält. Für jedes Profil wurden die Parameterklassen um eine Factory erweitert, welche für das Umwandeln der Datentypen in JAXB- und XML-Elemente verantwortlich ist. JAXB[35] ist Bestandteil der Web Services Interoperability Technology und definiert eine Programmierschnittstelle zum Binden von Daten an Java-Klassen aus einer XML-Schema-Instanz. Die Schnittstelle ist verantwortlich für das Marshalling und Unmarschalling von Klassen beim Versenden von Nachrichten. Die in den JAXB-Elementen verwendeten Namensräume, definiert als QNames, entsprechen den definierten Namensräumen der XSD- und WSDL-Dateien. Die so erzeugten Resultate werden anschließend an die Endpunktimplementierung des Services übergeben und versendet.

Die Operationen des System Query Profile müssen Beschreibungen von Serviceleveln und die Version der unterstützten Profile abfragen können. Zu diesem Zweck wurde

eine Properties Datei erstellt, welche die definierten Profile und Servicelevel samt Werten enthält. Die Servicelevel in dieser Datei enthalten die globalen Beschreibungen des Providers. Die Datei kann zur Laufzeit über den Klassenpfad des Anwendungsservers angefordert werden. Somit sind die Werte jederzeit zugänglich und können im laufenden Betrieb durch ein Austauschen der Datei geändert werden. Listing X zeigt den Aufbau der Datei.

```
profiles.SystemQueryProfile             = 1.0
profiles.RigQueryProfile                = 1.0
profiles.ExperimentQueryProfile         = 1.0
profiles.BasicRigAccessProfile          = 1.0
profiles.BasicBatchExperimentProfile  = 0.0
profiles.BasicInteractiveExperimentProfile = 1.0

serviceLevels.ResponsiblePerson = "Andreas Gerhardt"
.....
```

Listing 8: Konfigurationsdatei für Profile und Servicelevel

Während die Operation getProfileVer eine einzelne Version eines Profils ausliest, erstellt getAllProfiles eine Liste aus allen Profilen und sendet diese als Antwort. Das gleiche Prinzip gilt für die Operationen der Servicelevel. Diese müssen jedoch vorher unterscheiden ob die Anfrage für die globalen Beschreibungen oder für ein spezifisches Rig gestellt wurde. Sollten die Servicelevel eines Rig angefordert werden, muss zunächst über den RigService das angeforderte Rig ausgelesen und die dort enthaltene Liste serviceLevel bearbeitet werden.

Diese Operationen des Rig Query Profile sind für das Abfragen von Rigs verantwortlichen und müssen zuerst die Eingabeparameter für Nutzer und Nutzergruppen abfragen. Beide Parameter sind optional, da nur einer der beiden spezifiziert sein muss. Somit ist zunächst eine Fallunterscheidung zu machen und die passende ID zu bestimmen. Sollte eine der beiden IDs angegeben sein, werden die Anfragen nur für Rigs bearbeitet, auf denen der gegebene Nutzer über die nötigen Rechte verfügt. Alternativ werden alle Rigs abgefragt. Um Nutzer oder Nutzergruppen verwenden zu können muss erst ein Objekt der Entität Agent bzw. Group aus der Datenbank ausgelesen werden. Dazu wird der AgentService der Ontology API verwendet. Als nächstes folgen die Aufrufe des RigServe um die entsprechenden Rigs auszulesen und die Attribute abzufragen. Hierbei muss gesondert auf die Fehlermeldungen der API eingegangen werden und zwischen Fehlern auf Providerseite, Fehlern bei der Eingabe oder Fehlern innerhalb des Services unterschieden werden.

Operationen des Experiment Query Profile definieren Funktionalität zur Verwaltung von Experimenten. Die Operation GetAllActiveExperiments generiert eine Liste aller aktiven Experimente für ein gegebenes RigSet. Sollte ein Nutzer oder eine Nutzergruppe angegeben sein, werden nur Experimente der entsprechenden Person ausgelesen. Somit lesen wir zuerst den zugehörigen Agent oder die zugehörige Gruppe aus und rufen dann den AgentService auf um eine Abfrage auf der Datenbank mit den entsprechenden Daten zu tätigen. GetExperimentIDsWithStatus geht ähnlich vor jedoch werden nicht alle aktiven Experimente abgefragt, sondern die Anfrage um den Parameter Status erweitert. Für die Abfrage des Status eines Experiments muss nur die entsprechende ExperimentDefinition abgefragt und das Attribut ausgelesen werden. Die Operation AbandonExperiment wird zur Zeit nicht unterstützt, da keine Batch-Experimente verfügbar sind. Sie müsste den Status der ExperimentDefinition ändern und die Ausführung des Experiments auf dem Rig abbrechen. Mittels GetExperimentResults lassen sich die Ergebnisse eines Experiments anfordern. Diese werden in der ExperimentDefinition als Datei gespeichert. Nähere Informationen zum Dateitransfer finden sich in auf Seite 89.

Das Basic Rig Access Profile definiert Operationen für den Zugriff auf Rigs. Hierzu wird vor allem der RigService der Ontology API benötigt. Möchte ein Nutzer beispielsweise über die Operation RicAccessType abfragen, ob ein Rig buchbar ist lesen wir das spezifische Rig aus und geben den geforderten Wert zurück. Dies ist bei der Operation RigAvailability nicht möglich, da keine verfügbaren Zeiten gespeichert werden, sondern nur belegte Zeitslots. Die Operation muss folglich alle Reservierungen des spezifizierten Zeitraums abfragen und die freien Zeiten selbst ermitteln. Diese werden in einer Liste aus Zeitslots, mit Beginn- und Endzeit, gespeichert und an den Nutzer zurückgesendet. Bestandteil dieses Profils sind ebenfalls die Methoden zur Verwaltung von Buchungen und Warteschlangen. Dementsprechend wird hier das Buchungssystem eingebunden und verwendet. Um eine Buchung zu tätigen muss zunächst der Nutzer bzw. die Nutzergruppe ausgelesen werden und eine Verifikation stattfinden, ob Zugriffsrechte auf das angegebene Rig vorhanden sind. Ist dies der Fall muss der angegebene Zeitrahmen mit der Liste der bereits getätigten Reservierungen verglichen werden. Ist der Zeitrahmen verfügbar kann der ReservationService aufgerufen und eine neue Reservierung erzeugt und gespeichert werden. Der Nutzer erhält die erstellte IDs des Booking und kann so die Reservierung in der Zukunft abbrechen oder den Status überprüfen. Die Operationen zur Verwaltung von Warteschlangen werden nicht unterstützt und liefern entsprechend den Fehlercode 501 für nicht unterstützte Operationen zurück.

Das Basic Batch Experiment Profile wird komplett nicht unterstützt. Somit liefert jede der enthaltenen Operationen beim Aufruf den Fehlercode 500 zurück. Bisher existieren keine Pläne um an der Universität Stuttgart in absehbarer Zukunft die Nutzung von Batch-Experimenten anzubieten. Die Operationen des Profils für interaktive Experimente werden unterstützt. Diese bieten Funktionalität zum Erstellen, Lesen, Ändern und Löschen von interaktiven Experimenten. Hierbei ist eine

Unterscheidung zwischen dem Experiment und der Durchführung eines Experiments wichtig. Ein Experiment wird bspw. von einem Professor für seine Übungsgruppe erstellt und beinhaltet alle nötigen Informationen und Software zur Durchführung eines Experiments. Diese Informationen werden in einem InteractionPackage gespeichert. Für die Durchführung des Experiments ist eine Art Container notwendig, der die Informationen über den Nutzer speichert, der das Experiment ausführt, Informationen über das verwendete InteractionPackage und Rig sowie eine Möglichkeit zum Speichern von Informationen bereithält. Dieser Container wird durch die Entität ExperimentDefinition umgesetzt, für welche ein eigener Service in der Ontology API implementiert wurde (siehe Kapitel 6.3). Dieser wird nun aufgerufen um die entsprechenden Abfragen zu tätigen. Die Operationen zur Nutzung von Interfaces wurden implementiert. Die Spezifikation von Interfaces ist jedoch Bestandteil dieser Diplomarbeit, weswegen im Backend keine entsprechende Komponente vorhanden ist. Das Interface des Providers kann jedoch über das InteractionPackage ausgelesen und dem Nutzer gesendet werden.

Client

Ebenfalls implementiert wurde ein Client zur Anwendung auf Seite eines Consumers. Dieser dient als Schnittstelle zwischen einer Anwendung und den Services des RLS der Universität Stuttgart. Die Struktur der Klassen orientiert sich an den Services und beinhaltet für jedes Profil ein Paket mit Parameterklassen und eines mit zugehöriger Port- und Serviceimplementierung. Die aufrufende Komponente übergibt dem Client die nötigen Parameter. Dieser wandelt sie über die Factory in entsprechende JAXB-Elemente um und erzeugt eine Parameterklasse der Operation. Diese wird an den Port übergeben und gesendet.

Dateitransfer

Im Rahmen der Services ist es nötig, Dateien zwischen Consumer und Provider auszutauschen. Dies betrifft insbesondere Operationen zum Senden von Ergebnissen sowie der Spezifikation eines Interfaces oder den Dateien mit Definitionen von Experimenten. Zur Übertragung der Dateien wurde der Message Transmission Optimization Mechanism (MTOM) [36] verwendet. MTOM stellt die Empfehlung des W3C für die Übertragung von binären Daten in Webservices dar und basiert auf dem XML-binary Optimized Packaging (XOP) [37]. Dieses definiert ein Infoset, welches als Container für binäre und textuelle Informationen genutzt wird. Für die Verwendung von MTOM in der Implementierung muss die Bibliothek Metro eingebunden werden. Diese stellt Annotationen bereit, um die Nutzung von MTOM einzuschalten. Dateien werden als Anhang gesendet. MTOM definiert einen Schwellenwert für die Größe der Datei. Wird dieser Wert überschritten, wird die Datei als Anhang übertragen. Setzt man diesen Wert auf 0, wird jede Datei als Anhang gesendet. Für die Übertragung der Datei wurde das TCP/IP-Protokoll eingeschaltet.

Umsetzung der Antwortcodes

Die umfangreichen Antwortcodes erfordern eine ausführliche Behandlung in der Implementierung. Wie bereits in Kapitel 5 beschrieben ist es wichtig, dass sowohl Consumer, als auch Nutzer, genau unterscheiden können wieso eine Operation fehlgeschlagen ist. Die Services müssen hierbei zwischen Fehlern des Providers, des Consumers und des Nutzers unterscheiden können. Jeder Service führt dazu eine Fallunterscheidung aus und übermittelt an Hand der Fehler den entsprechenden Antwortcode und eine Fehlermeldung.

Der Client verwendet einen `ExceptionHandler` zur Bestimmung der Fehlermeldung an Hand des zurück gelieferten Antwortcodes. Die Fehlermeldungen wurden als Exceptions implementiert, wodurch das System des Consumers aufgefordert wird, eine Unterscheidung der verschiedenen Fehler zu treffen. Das Objekt der spezifischen Parameterklasse wird an den Handler übergeben. Dieser liest den Antwortcode aus und generiert die entsprechende Exception und Fehlermeldung.

6.5 Umsetzung der nicht-funktionalen Anforderungen

Datenschutz:

Die Entitäten des Datenmodells basieren auf Identifikatoren und erlauben keine Rückschlüsse auf eine Person oder Institution. Aus Gründen des Datenschutzes wurde ein externer Identifikator in die entsprechenden Entitäten aufgenommen. Dieser trennt die internen Daten zu Personen von dem systeminternen Identifikator. Einzige Ausnahme bildet die eMail-Adresse eines Nutzers oder einer Nutzergruppe bei der Buchung einer Hardwareinstallation. Diese ist jedoch optional. Ergänzend werden die Daten verschlüsselt und signiert. Somit können keine dritten Personen auf die verwendeten Daten zugreifen.

Konsistenz:

Ein Teil der Konsistenz ist bereits über das Buchungssystem und die Ontology API abgedeckt. Ein Ändern der Daten durch die Services erfolgt am Ende einer Operation. Sollte die Änderung fehlschlagen wird die Operation abgebrochen und der Nutzer informiert.

Verfügbarkeit:

Die Komponente ist als Webservices implementiert, welche auf einem Anwendungsserver dauerhaft verfügbar sind. Nutzer können zu jedem beliebigen Zeitpunkt on-demand auf Ressourcen zugreifen.

Wartbarkeit und Erweiterbarkeit:

Die entwickelte Komponente erlaubt eine einfache Wartung. Die Schnittstellenbeschreibungen mit XSD und WSDL können leicht geändert werden, genauso wie das Hinzufügen, bzw. Löschen, von Operation. Sollten lediglich Eingabe- und Ausgabeparameter der Operationen geändert werden, sind diese getrennt von der Beschreibung durch WSDL modifizierbar.

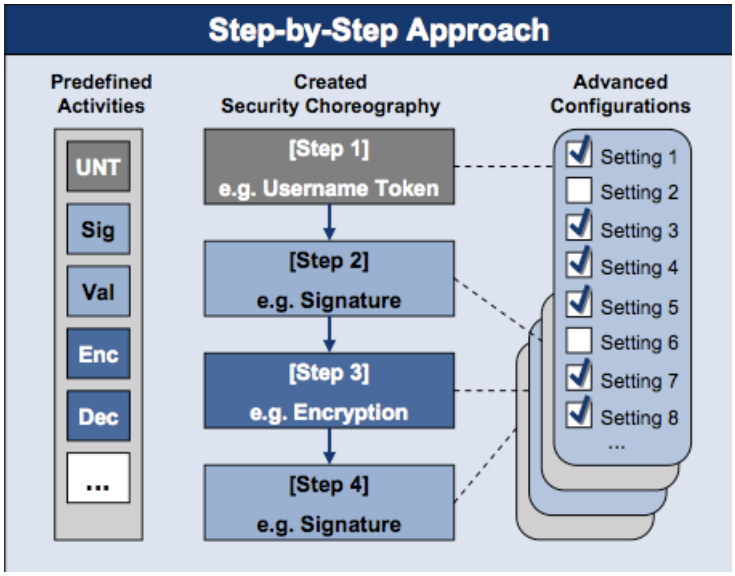
Der entwickelte Code ist dokumentiert und evaluiert worden. Die Erläuterungen der Diplomarbeit und in dem geschriebenen Standard ermöglichen ein leichtes Einlernen in die Komponente. Somit fällt es zukünftigen Entwicklern leicht, die Komponente zu ergänzen.

Migration:

Die implementierte Komponente lässt sich leicht migrieren und auf anderen Systemen zum Einsatz bringen. Der entwickelte Prototyp erlaubt eine schnelle Anpassung an ein neues System und die Kapitel 6.6 beschriebene Konfiguration ist einfach.

Sicherheit:

Das Konzept der Sicherheit besteht aus zwei Aspekten, einer Nutzer-Authentifizierung und dem Verschlüsseln und Signieren der Daten in einer Nachricht. Die Nutzer-Authentifizierung erreichen wir über das UsernameTokenProfile der WS-



Security Spezifikation. Hauptbestandteil dieses Profils ist der UsernameToken, welcher im Header einer Nachricht übertragen wird und den Nutzernamen, das Passwort in Klartext oder als Hash-Wert, die Nonce und einen Zeitstempel beinhaltet. Nonce und Zeitstempel sollen einen möglichen Angriff verhindern, in dem ein Angreifer den UsernameToken modifiziert. Die Nonce ist ein einmaliges Zufallswort, das zur Bildung des Hash-Wertes aus dem Klartextpasswort benutzt wird. Als Mechanismus wurde die Benutzerauthentifizierung mittels symmetrischem Schlüssel gewählt. Als Benutzer wird hierbei nicht jeder Benutzer eines RLS betrachtet, sondern ein Consumer, der Zugriff auf das RLS des Providers hat. Diese Variante vereinfacht die Datenhaltung der Benutzer signifikant. Neue Teilnehmer aus anderen Domänen können so leicht auf die Funktionalitäten des Providers zugreifen. Die Benutzer werden im Truststore des Anwendungsservers abgespeichert. Dieser muss in der zugehörigen WS-Policy über das TrustStore-Element verlinkt werden.

Die Verwendung des Truststore des Anwendungsservers ist im nächsten Unterkapitel beschrieben. Auf Serverseite muss ein SOAPHandler einkommende Nachrichten abfangen und überprüfen, ob der UsernameToken im Header der Nachricht gesetzt wurde. Benutzername und Passwort werden an Hand des Truststore überprüft. Auf Seite des Client muss ein SOAPHandler implementiert werden, welcher in Listing 9 dargestellt ist. Die Methode handleMessage überprüft zunächst, ob es sich um eine aus- oder eingehende Nachricht handelt und setzt anschließend die beiden Werte für den Benutzernamen und das Passwort.

Abbildung 20 Schritte von WS-Security

```
public class UsernameTokenHandler
implements SOAPHandler<SOAPMessageContext>
{
   ...
   public boolean handleMessage(SOAPMessageContext context) {...}
   ...
   public boolean handleFaut(SOAPMessageContext context) {...}
   ...
   public void close(MessageContext context) {...}
   ...
}
```

Listing 9: Darstellung der Methoden eines SOAPHandler

Zur Sicherheit der Daten und Vertraulichkeit wird der Body einer Nachricht verschlüsselt und zur Wahrung der Datenintegrität signiert. Die WS-Security Spezifikation bietet hierzu SecurityTokens, welche ebenfalls im Header der Nachricht übertragen werden. Diese ermöglichen es dem Empfänger, die Signatur der Daten zu überprüfen und die Daten zu entschlüsseln. Die Verschlüsselung und Signatur wird an Hand von Zertifikaten erstellt, welche ebenfalls im Anwendungsserver hinterlegt werden. Als Mechanismus wurde eine asymmetrische Verschlüsselung unter Zuhilfenahme von X.509-Zertifikaten gewählt. Diese Zertifikate sind ein Standard für eine Public-Key-Verschlüsselung. Die Zertifikate beinhalten den öffentlichen Schlüssel und speichern Details zum Eigentümer, zur Gültigkeit des Zertifikats und zu den verwendeten Algorithmen. Das Verfahren entspricht dem üblichen Verfahren asymmetrischer Verschlüsselung. Der Sender verschlüsselt den Inhalt der Nachricht mit seinem privaten Schlüssel und fügt den öffentlichen Schlüssel zur Nachricht hinzu. Der Empfänger kann diesen dann benutzen um die Nachricht wieder zu entschlüsseln. Sowohl auf Server, als auch auf Clientseite muss ein Handler verfügbar sein. Dieser überprüft die ein- und ausgehenden Nachrichten und ver-, bzw. entschlüsselt die Inhalte. Als zweiter Schritt werden die Signaturen überprüft. Das Konzept erfüllt die Anforderungen an die Sicherheit und unterbindet die Manipulation und das Auslesen durch Dritte.

6.6 Konfiguration

Dieses Unterkapitel beschreibt die Konfiguration und das Bereitstellen der Services auf einem Anwendungsserver. Zu Testzwecken wurde Apache Glassfish 3 verwendet. Die hier erläuterten Vorgehensweisen beziehen sich auf diesen Anwendungsserver und können bei anderen Servern abweichen.

Es sind nur wenige Schritte nötigt, um die Komponente einsetzen zu können. Diese betreffen ebenfalls die Konfiguration der verwendeten Komponenten des Buchungssystems und der Ontology API. Zunächst muss eine Datei mit Parametern

für die LiLa-Komponenten in den Klassenpfad des Anwendungsservers eingefügt werden. Diese beinhaltet Informationen zur Datenbank und Adressen des Buchungssystems. Ebenfalls in den Klassenpfad einzufügen ist die bereits beschriebene Datei zur Ablegung der Profilversionen und Beschreibungen der Servicelevel.

Als nächstes müssen die Einstellungen der Java Virtual Machine für den Anwendungsserver konfiguriert werden. Dabei sind die Werte für den minimal verwendeten Arbeitsspeicher auf 512MB und für den maximal verwendeten Arbeitsspeicher auf mindestens 1024BM zu setzen. Diese Werte entsprechen den benötigten Werten während des Tests. Der zugewiesene Arbeitsspeicher während dem Betrieb kann nach oben hin abweichen.

Nun müssen noch die Einstellungen für WS-Security vorgenommen werden. Das Erstellen und Verwaltne von X.509-Zertifikaten ist nicht Gegenstand der Diplomarbeit, diese müssen somit bereits vorhanden sein. Eine Beschreibung der Verwendung von Zertifikaten in Glassfish findet sich in [38]. Nutzerlogins können entweder über die Admin-Konsole gespeichert oder durch vorhandene Truststores eingefügt werden. Der Pfad zu den Einstellungen in der Admin-Konsole laute:

Configuration → Security → Realms → File → Manage Users

6.7 Evaluation

Zur Unterscheidung von möglichen Fehlerquellen wurden zuerst die Services getestet und verifiziert und darauf aufbauen der Client. Der Test der Funktionalität des Buchungssystems und der Ontology API war nicht Bestandteil der Evaluation.

Die Services wurden auf einer virtuellen Maschine auf einem Apache Glassfish Anwendungsserver in der Version 3 bereitgestellt und eine neue Datenbank mit Testwerten gefüllt. Diese Konfiguration entspricht einer möglichen Anwendung in der Produktion. Die Webservices wurden anschließend mit SoapUI auf ihre Funktionalität getestet. Für jede Operation wurde ein festes Set an Testwerten verwendet. Die Testwerte bestanden aus gültigen Werten und Werten die Fehler erzeugen mussten. Hier wurden beispielsweise fehlerhafte Eingaben oder in der Datenbank nicht existierende Identifikatoren verwendet. Änderungen an der Datenbank, die während eines Szenarios von Bedeutung sind, wurden direkt über Aufrufe der Datenbank vorgenommen. Hierzu zählt die Änderung des Status einer Reservierung oder das Hinzufügen von Ergebnissen zu einem Experiment.

Nach der Verifikation der Services wurde der Client getestet. Dieser verwendete die bereits getesteten Services zur Überprüfung der Funktionalität. Das Hauptaugenmerk der Tests lag Abfragen, die Fehler produzieren und Tests bei denen Dateien

übertragen wurden. Die Aufgabe des Clients im Wesentlichen ist das entgegennehmen von Parameter, welche anschließend an die Services gesendet werden. Von der Verwendung der Dateien abgesehen, beinhalten die Operationen des Client keine nennenswerte Algorithmik. Ebenfalls getestet wurde eine Benutzerauthentifizierung unter Verwendung der beschriebenen UsernameToken. Hierzu wurde ein Benutzer im Anwendungsserver angelegt und der Truststore in die Client API integriert.

7. Zusammenfassung

Gegenstand dieser Diplomarbeit war ein Standard zur Interoperabilität von Remote Laboratory Systems. Zunächst wurden verwandte Arbeiten vorgestellt, die sich mit der Thematik beschäftigten oder für die Durchführung der Diplomarbeit von Bedeutung waren. Als Basis für das Verständnis wurden RLS beschrieben und wichtige Begriffe definiert. Ergänzend wurde Grid Computing vorgestellt als Grundlage für den Vergleich zu RLS und die verwendeten Technologien als Grundlage für die Implementierung. Es wurde ein Vergleich zu Grid Computing gezogen und eine Einordnung in wissenschaftliche Paradigmen vorgenommen. Als Einleitung der Konzepte spielt der Begriff Interoperabilität eine wichtige Rolle. Eine Definition und Abgrenzung zu Kompatibilität soll diesen erläutern und in den Kontext von RLS bringen.

Als nächster Schritt wurde die Systemübersicht, sowie die funktionalen und nicht-funktionalen Anforderungen an die Komponente präsentiert. Über die Anwendungsfälle wird ein festes Set an Funktionalität spezifizert, welches als Grundlage für die Einteilung der Operationen in Profile dient. Die Profile kapseln eine zusammengehörende Funktionalität und wurden durch Szenarios bechrieben. Mehrere Operationen je Profil bilden die gewünschte Funktionalität ab und wurden durch Interfaces der Parameter definiert. Im Rahmen der Operationen war es nötig eine umfangreiche Liste an Fehlercodes und eine Menge von Serviceleveln zu definieren. Für das Bereitstellen von generischen Benutzeroberflächen wurde eine Spezifikation definiert, welche ein festes Basisset an Elementen beschreibt. Durch diese Elemente wird ein Consumer in die Lage versetzt ein generisches Interface für die Definition einer Experimentdurchführung zu erzeugen.

Anschließend wurde die Umsetzung der Services und des Client vorgestellt. Hierzu war zunächst eine Änderung des Datenmodells und der zugehörigen Methoden in der Ontology API nötig. Das vorhandene Datenmodell wurde um Entitäten und Attribute erweitert, die für den Einsatz der entwickelten Komponente nötig sind. Jedes Profil wurde als ein SOAP-basierter Webservice implementiert, welcher unter Verwendung eines existierenden Buchungssystems und einer Komponente zur Datenhaltung auf die Ressourcen eines RLS zugreifen kann und somit die Nutzung durch externe

Systeme ermöglicht. Abschluss bildet eine Erläuterung der Konfiguration der Komponente und eine Beschreibung der durchgeführten Tests.

7.1 Ausblick

Systeme für virtuelle Labore sind ein wichtiger Bestandteil der Lehre und Forschung geworden. Das Interesse an einer globalen Infrastruktur ist groß und eine solche Zusammenarbeit von Laboren ist Gegenstand vieler Forschungsprojekte. Der in dieser Arbeit entwickelte Standard versetzt existierende Systeme in die Lage zu interagieren und zusammen zu arbeiten. Studenten der beteiligten Universitäten erhalten dadurch die Möglichkeit, auf eine deutlich größere Menge an Experimenten und Ressourcen zuzugreifen. Anstatt identische Hardwareinstallationen in jedem RLS bereitzustellen, können sich Provider absprechen und die Verantwortung für spezifische Experimente verteilen. Bisher mussten Provider ihre Ressourcen in viele verschiedene Experimente investieren, damit diese in ihrem Verbund verfügbar sind. Nun können Experimente verteilt werden was zu einer Steigerung der Qualität führt.

Die Entwicklung von RLS ist noch nicht am Ende sondern fängt gerade erst an. Wünschenswert wäre die Bildung eines Intergrid, an welchem nicht nur die Partner des GOLC beteiligt sind. Der Nutzen für die Forschung und Lehre wäre sehr hoch. Eine Erweiterung der Infrastruktur könnte auch Schulen einbinden, welche nicht über die nötigen Mittel für qualitativ hochwertige Experimente verfügen. Ein möglicher Schritt wäre auch die Eingliederung in vorhandene Strukturen des Grid Computing um auf weitere Ressourcen zuzugreifen und die eigenen Ressourcen einem breiteren Publikum anzubieten. Die Zukunft verspricht spannend zu sein und die zeitnahe Umsetzung des Standards durch die beteiligten Provider kann einen großen Schritt bedeuten, auch für die Universität Stuttgart.

Literaturverzeichnis

[1] Science Says – A collection of Quotations on the History, Meaning and Practive of Science
Rob Kaplan, 2000

[2] iLabs - Massachusetts Institute of Technology
http://icampus.mit.edu/ilabs/

[3] LiLa Booking System: Architecture and Conceptual Model of a Rig Booking System for On-Line Laboratories
V.Mateos, A.Gallardo, T.Richter, L.Bellido, P.Debicki and Victor Villagra

[4] LiLa - Lirabry of Labs - Forschungsprojekt der Universität Stuttgart

[5] GOLC Technical Metadata Profile, 2011
Michael E. Auer, Danilo Garbi Zutin, David Lowe, David Boehringer, Pascal Grube, Thomas Richter, Irene Schumm, Claus Spiecker, Nicole Natho

[6] State of the art about remote laboratories paradigms - foundations of ongoing mutations
C.Gravier, J.Fayolle, B. Bayard, M.Ates, J.Lardon - 2008

[7] Software Technologies, Architectures and Interoperability in Remote Laboratories
Mehmet Efe Özbek, Ali Kara, Musa Atas

[8] Interoperability of Remote Laboratories Systems
Herbert Yeung, David Lowe, Steve Murray

[9] What is the Grid? A Three Point Checklist
Ian Foster, 2002

[10] Reference Model for Service Oriented Architecture 1.0, 2006
OASIS Standard

[11] UDDI Version 3.0.2 Specification
OASIS Standard

[12] Quelle Abbildung SOA Prinzip
http://de.wikipedia.org/w/index.php?title=Datei:Webservice.svg&filetimestamp=20120605135458

[13] SOAP Version 1.1 Specifications, 2007
World Wide Web Consortium, W3C

[14] Web Service Description Language Specification
World Wide Web Consortium, 2001s

[15] Web Services Policy 1.5 Specification, 2007
W3C, http://www.w3.org/TR/ws-policy/

[16] Web Services Security: SOAP MEssage Security 1.0, 2004
OASIS Standard

[17] The Fourth Paradigm - Data Intensive Scientific Discovery
Jim Gray, editiert durch Tony Hey, Stewart Tansley, KriSTin Tolle

[18] The Grid 2 - Blueprint for a new Computing Infrastructure, 2004
Edited by Ian Foster, Carl Kesselman

[19] The Large Hadron Collider
http://lhc.web.cern.ch/lhc/

[20] Introduction to Grid Computing - IBM Redbooks
Bart Jacob, Michael Brown, Kentaro Fukui, Nihar Trivedi - 2005

[21] The Open Grid Services Architecture, Version 1.5
I. Foster, H. Kishimoto A. Savva et. al - 2006

[22] Globus Documentation Project - Globus Forum
http://gdp.globus.org

[23] Open Grid Services Infrastructure Version 1.0
S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt

[24] IT Wissen - Online Lexikon für Informationstechnologie
Artikel zu Grid Computing in 3.2 und Artikel zu Kompatibilität in 4.1

[25] Wikipedia Artikel zu Interoperabilität
http://wikipedia.org/Interoperabilität

[26] AFUL Interoperability Working group, Deutsche Übersetzung der Definition von Interoperabilität
http://interoperability-definition.info/de/

[27] A High-Level Framework for Network based Ressource Sharing
James E. White, 1976

[28] JSON-RPC 2.0 Specification, http://json-rpc.org/wiki/specification

[29] Architectural Styles and the Design of Network-based Software Architectures
Roy Thomas Fielding, 2000

[30] Universally Unique Identifier (UUID)
RFC4122 der Open Software Foundation

[31] Heterogene Softwaresysteme, 2010
Cornelius Köpp, Universität Hannover

[32] Spring Application Framework für Java,
Spring Source - VMWare

[33] Article: Which style of WSDL should I use?
from IBM Developer Works, Russell Butek, 2005

[34] Basic Profile Version 1.1
Web Services Interoperability Organization

[35] Java Architecture for XML Binding (JAXB)
Java Specification Request 222

[36] SOAP Message Transmission Optimization Mechanism
W3C Recommendation, 2005

[37] XML-binary Optimized Packaging
W3C Recommendation, 2005

[38] Configuring Keystores and Truststores at:
http://docs.oracle.com/cd/E17802_01/webservices/webservices/reference/tutorials/wsit/doc/WSIT_Security6.html

[39] Remote-Labs.eu - Towards Future Education
http://remote-labs.eu

[40] Grid Computing von Tobias Wagner, 2012

[41] Working with SOAP Messages - http://docs.oracle.com
Oracle Corporation

[42] Standardisierungen in der Grid Community
Robert Schrader

A. Anhang

A.1 WSDL Beschreibung des Rig Query Profile als Beispiel

```
<?xml version = "1.0" encoding = "utf-8"?>
<wsdl:definitions name = "RigQueryProfileService"
targetNamespace = "http://online-lab.org/InteroperabilityStandard/Services/RigQueryProfile"
xmlns:tns  ="http://online-lab.org/InteroperabilityStandard/Services/RigQueryProfile"
xmlns:par  = "http://onlinelab.org/InteroperabilityStandard/Services/Parameters/RigQueryProfile"
xmlns:tcp  ="http://java.sun.com/xml/ns/wsit/2006/09/policy/soaptcp/service"
xmlns:wsu = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsoms = "http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization"
xmlns:wsp  = "http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsdb = "http://www.w3.org/2001/XMLSchema"
xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/">

    <wsdl:documentation>
        Definitions of the web services defined in the GOLC InteroperabilityStandard for Remote
        Laboratory Systems v1.0.
        This file defines the web services used in the Rig Query Profile.
    </wsdl:documentation>

    <wsp:Policy wsu:Id="Transport_Policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <wsoms:OptimizedMimeSerialization/>
                <tcp:OptimizedTCPTransport enabled="true"/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>

    <xsd:schema>
        <xsd:import namespace = "........"  schemaLocation = "RigQueryProfileParameters.xsd"/>
    </xsd:schema>
```

```
 <wsdl:message name = "GetRigsRequest">
      <wsdl:part name = "getRigsInput"  element = "par:getRigsInput"/>
 </wsdl:message>

 <wsdl:message name = "GetRigsResponse">
      <wsdl:part name  = "getRigsResult" element = "par:getRigsResult"/>
 </wsdl:message>

 <wsdl:message name = "GetRigInfoRequest">
      <wsdl:part name  = "getRigInfoInput“ element = "par:getRigInfoInput"/>
</wsdl:message>

<wsdl:message name = "GetRigInfoResponse">
       <wsdl:part name = "getRigInfoResult“  element = "par:getRigInfoResult"/>
  </wsdl:message>

  <wsdl:message  name = "GetRigStatusRequest">
       <wsdl:part  name = "getRigStatusInput"  element = "par:getRigStatusInput"/>
  </wsdl:message>

  <wsdl:message name  = "GetRigStatusResponse">
       <wsdl:part name  = "getRigStatusResult" element = "par:getRigStatusResult"/>
  </wsdl:message>

 <wsdl:portType name = "RigQueryProfile_PortType">

      <wsdl:operation  name = "GetRigs">
            <wsdl:documentation>
                  ...
            </wsdl:documentation>
            <wsdl:input  message = "tns:GetRigsRequest"/>
            <wsdl:output message = "tns:GetRigsResponse"/>
      </wsdl:operation>

      <wsdl:operation  name = "GetRigInfo">
            <wsdl:documentation>
                  ...
            </wsdl:documentation>
                  <wsdl:input  message = "tns:GetRigInfoRequest"/>
                  <wsdl:output message = "tns:GetRigInfoResponse"/>
            </wsdl:operation>

      <wsdl:operation  name = "GetRigStatus">
            <wsdl:documentation>
                  ...
            </wsdl:documentation>
            <wsdl:input  message = "tns:GetRigStatusRequest"/>
            <wsdl:output message = "tns:GetRigStatusResponse"/>
      </wsdl:operation>

</wsdl:portType>

<wsdl:binding name = "RigQueryProfile_Binding" type = "tns:RigQueryProfile_PortType">
```

```
   <soap:binding style = "document" transport = "http://schemas.xmlsoap.org/soap/http" />

        <wsp:PolicyReference URI="#Transport_Policy" wsdl:required="true" />

        <wsdl:operation name = "GetRigs">
        <soap:operation soapAction = "GetRigs"/>
                <wsdl:input>
                        <soap:body use = "literal"/>
                </wsdl:input>
                <wsdl:output>
                        <soap:body use = "literal"/>
                </wsdl:output>
        </wsdl:operation>

        <wsdl:operation name = "GetRigInfo">
        <soap:operation soapAction = "GetRigInfo"/>
                <wsdl:input>
                        <soap:body use = "literal"/>
                </wsdl:input>
                <wsdl:output>
                        <soap:body use = "literal"/>
                </wsdl:output>
        </wsdl:operation>

        <wsdl:operation name = "GetRigStatus">
        <soap:operation soapAction = "GetRigStatus"/>
                <wsdl:input>
                        <soap:body use = "literal"/>
                </wsdl:input>
                <wsdl:output>
                        <soap:body use = "literal"/>
                </wsdl:output>
        </wsdl:operation>

   </wsdl:binding>

<wsdl:service name = "RigQueryProfile">
        <wsdl:documentation>
                ...
        </wsdl:documentation>
        <wsdl:port binding = "tns:RigQueryProfile_Binding" name = "RigQueryProfile_PortType">
                <soap:address location = "http://nflgolcdev/....."/>
        </wsdl:port>
</wsdl:service>

 </wsdl:definitions>
```

Listing 10 WSDL Beschreibung des Rig Query Profile

A.2 XSD Typ-Definitionen des Rig Query Profile

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes"?>
<xsd:schema version = "1.0"
targetNamespace="http://online-lab.org/InteroperabilityStandard/.../Parameters/RigQueryProfile"
xmlns:tns = "http://online-lab.org/InteroperabilityStandard/Services/Parameters/RigQueryProfile"
xmlns:xmime = "http://www.w3.org/2005/05/xmlmime"
xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

<xsd:element name = "getRigsInput">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name = "groupID" type = "xsd:string" minOccurs = "0" maxOccurs = "1" nillable = "."/>
    <xsd:element name = "rigSetID" type = "xsd:string" minOccurs = "0" maxOccurs = "1" nillable = "."/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name = "getRigsResult">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element name = "rigSets" type = "xsd:string" minOccurs = "0" maxOccurs = "unbounded" ... />
    <xsd:element name = "responseCode" type = "xsd:int" minOccurs = "1" maxOccurs = "1"  ... />
    <xsd:element name = "errorMessage"  type = "xsd:string" minOccurs = "0" maxOccurs = "1 ... />
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name = "getRigStatusInput">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name = "groupID"  type = "xsd:string" minOccurs = "1" maxOccurs = "1" ... />
    <xsd:element name = "rigSetID" type = "xsd:string" minOccurs = "1" maxOccurs = "1" ... />
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name = "getRigStatusResult">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name = "online“ type = "xsd:boolean" minOccurs = "1" maxOccurs = "1" ... />
    <xsd:element name = "available" type = "xsd:boolean" minOccurs = "1" maxOccurs = "1" ... />
    <xsd:element name = "statusMessage" type = "xsd:string"  minOccurs = "0" maxOccurs = "1" ... />
    <xsd:element name = "responseCode" type = "xsd:int" minOccurs = "1" maxOccurs = "1" ... />
    <xsd:element name = "errorMessage“ type = "xsd:string"  minOccurs = "0" maxOccurs = "1" ... />
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name = "getRigInfoInput">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name = "groupID" type = "xsd:string" minOccurs = "0" maxOccurs = "1" ... />
    <xsd:element name = "rigSetID" type = "xsd:string" minOccurs = "1" maxOccurs = "1" ... />
  </xsd:sequence>
</xsd:complexType>
```

```
</xsd:element>

<xsd:element name = "getRigInfoResult">
<xsd:complexType>
   <xsd:sequence>
      <xsd:element name = "rigSetProperties" type = "xsd:base64Binary" minOccurs = "0" maxOccurs = "1"
                   nillable = "true" xmime:expectedContentTypes="text/xml"/>
      <xsd:element name = "fileName" type = "xsd:string" minOccurs = "0" maxOccurs = "1" ... />
      <xsd:element name = "responseCode" type = "xsd:int" minOccurs = "1" maxOccurs = "1.... />
      <xsd:element name = "errorMessage" type = "xsd:string"  minOccurs = "0" maxOccurs = "1" ... />
   </xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:schema>
```

Listing 11 XSD Typ-Definitionen des Rig Query Profile

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Leonberg, 16. Juni 2012

_____

Andreas Gerhardt