Fakultät für Technik

Studiengang Technische Informatik

Design und Implementierung einer Testautomatisierung der Schnittstellentests zwischen Einsatzleitsystemen und einem Notrufabfragesystem

Bachelor-Thesis

Erstprüfer	Prof. Dr. rer. nat. Richard Alznauer
Zweitprüfer	Prof. Dr. –Ing. Thomas Greiner
Betrieblicher Betreuer	Dipl. Ing. Helmut Heck
vorgelegt von	Moritz Hegel
Matrikelnummer	303663

Abgabetermin	04.08.2014

Erklärung

Ich versichere, die beiliegende Bachelor-Thesis „Design und Implementierung einer Testautomatisierung der Schnittstellentests zwischen Einsatzleitsystem und einem Notrufabfragesystem“ selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet zu haben.

[Ort], den [Datum] (Unterschrift)

Kurzfassung

Die vorliegende Bachelorarbeit beschäftigt sich mit der Konzeptionierung und Implementierung einer Testautomatisierung der Schnittstellentests zwischen dem Einsatzleitsystem und einem Notrufabfragesystem. Es wird der Frage nachgegangen, ob eine Testautomatisierung mit den gegebenen Programmen realisierbar ist und welche Bedingungen hierfür erfüllt sein müssen. Ziel ist es, ein Konzept zu realisieren, um die Funktionen der Schnittstelle automatisiert testen zu können. Ein weiteres Ziel ist es die Erstellung der Testfälle durch dieses Konzept zu vereinfachen. Die Umsetzung erfolgt mit dem Automatisierungswerkzeug „QF-Test“.

Die erfolgreiche Umsetzung zeigt, dass die Testautomatisierung sowohl dabei helfen kann, den Aufwand für das Testen komplexer Systeme zu reduzieren als auch den Nutzen für die Anwender und die Firma erhöhen kann.

Inhalt

Abkürzungsverzeichnis vi

1 Einleitung 1
1.1 Software Test Allgemein 2
1.1.1 Aufgaben und Ziele des Testens 2
1.1.2 Testprinzipien 2
1.1.3 Testprozess 3
1.1.4 Definition eines Fehlers 3
1.1.5 Ursachenkette für Fehler 4
1.1.6 Fehlerwirkung / Fehlerzustand 4
1.1.7 Fehlermaskierung 4
1.1.8 Testverfahren 4
1.2 Grundlagen Testautomatisierung 5
1.2.1 Was ist unter Testautomatisierung zu verstehen? 5
1.2.2 Nutzen und Aspekte für die Testautomatisierung 6

2 Problemstellung 7

3 Ziele der Arbeit 8

4 Stand der Technik 9
4.1 „NORUMAT TIP“ System 9
4.2 BOSPORUS TIP Schnittstelle 10
4.2.1 Philosophie 10
4.2.2 Kommunikation 10
4.3 Testprozedur bisher 12

5 Lösungsvarianten 13
5.1 Komplett neues Konzept entwickeln 13
5.2 Verwenden was vorhanden ist 13

6 Gewählte Lösungsvariante 14

7 Verwendete Werkzeuge 15
7.1 QF-Test 15
7.1.1 Testsuite-Knoten 16
7.1.2 Prozeduren 16
7.1.3 Extrasequenzen-Knoten 16
7.1.4 Fenster und Komponenten 16
7.1.5 Sequenzknoten 17
7.1.6 Ablaufsteuerungsknoten 17
7.1.7 Programmknoten 18

7.1.8 Nützliche Funktionen ... 18
7.1.9 Protokoll ... 18
7.2 BsptTool ... 20
7.3 Groovy Console ... 21
7.3.1 Was ist Groovy? ... 21
7.3.2 Closure und each-Methode ... 21
7.3.3 Programmbeschreibung ... 23
7.4 Linphone ... 23

8 Konzeptdesign ... 24
8.1 Risikoanalyse ... 25
8.2 Funktionsbibliothek in QF-Test ... 27
8.2.1 Aufbau einer Prozedur ... 28
8.3 Testszenarien entwickeln ... 29
8.3.1 Testkriterien festlegen ... 29
8.4 Auswertung Testszenario ... 31
8.4.1 Aufbau XML Dokument ... 32
8.4.2 Aufbau eines Groovy Skripts ... 33
8.4.3 Darstellung der Testergebnisse im Protokoll ... 35

9 Vorgehensweise für die Entwicklung von Testszenarien mit QF-Test ... 36
9.1 Konkrete Entwicklungsbeschreibung an einem Testszenario Beispiel Notruf ... 36
9.2 Realisierte Sequenzdiagramme ... 39

10 Mögliche Testerweiterungen ... 43
10.1 Batch Modus ... 43
10.2 Daemon Modus ... 43

11 Zusammenfassung und Ausblick ... 44
11.1 Zusammenfassung ... 44
11.2 Ausblick ... 45

12 Literaturverzeichnis ... 46
12.1 Firmeninterne Quelle ... 46

Abbildungsverzeichnis ... 47

Glossar ... 49

Anhang A ... 50

Abkürzungsverzeichnis

E

ELS · Einsatzleitsystem

G

GUI · Graphical User Interface

I

ID · Identifikator
IDD · Interface Design Description

M

msg · Message

N

NORUMAT TIP · Notrufautomat Tetra Internet Protokoll

R

rc · Runcontext

S

SUT · System Under Test

U

uvm · und vieles mehr

X

XML · Extinsible Markup Language

Z

z. B. · zum Beispiel

1 Einleitung

Das vorliegende Dokument beschreibt die Konzeption und Umsetzung einer Testautomatisierung für die „BOSPORUS TIP“-Schnittstelle. Die „BOSPORUS TIP“-Schnittstelle ist ein Teil des Produktes „NORUMAT TIP“ und für die Anbindung unterschiedlicher Einsatzleitsysteme verantwortlich. Das „NORUMAT TIP“-System wird von der Firma THALES DEFENCE & SECURITY SYSTEMS GmbH entwickelt und vertrieben. Die Aufgabe des Systems ist es, bei Notfällen oder Katastrophen, den Disponenten bei seiner Arbeit zu unterstützen. Dabei spielt die Zuverlässigkeit des Systems eine enorm wichtige Rolle. Über das System werden beispielsweise Notrufe angenommen, bearbeitet und weitergeleitet. Es können Funkgeräte angeschlossen werden, die die Funkverbindungen zu den Einsatzkräften vor Ort ermöglichen. Die Benutzerschnittstelle (GUI) wird von THALES selbst entwickelt und stellt dem Disponenten sämtliche Funktionen zur Verfügung. Bei jeder Aktion die der Disponent über die Benutzerschnittstelle auslöst, wird über die „BOSPORUS TIP“-Schnittstelle das Einsatzleitsystem informiert.

Der „BOSPORUS TIP“-Schnittstelle kommt in diesem Zusammenhang eine besondere Rolle zu. Sollten aus irgendwelchen Gründen die Nachrichten nicht gesendet werden oder fehlerhaft sein, können die Einsätze der Feuerwehr oder Polizei nicht koordiniert und geplant werden. Die Folgen wären fatal. Deshalb muss gewährleistet sein, dass die „BOSPORUS TIP“-Schnittstelle einwandfrei funktioniert. Bei einem einzigen Notruf werden bis zu elf Nachrichten über die „BOSPORUS TIP“-Schnittstelle übertragen. Es muss gewährleistet werden, dass alle Nachrichten übertragen wurden und der Inhalt der Nachrichten korrekt ist. Damit die „BOSPORUS TIP“-Schnittstelle ausreichend überprüft werden kann, muss ein automatisiertes Testkonzept erarbeitet werden.

1.1 Software Test Allgemein

1.1.1 Aufgaben und Ziele des Testens

Unter Testen versteht man im Allgemeinen das stichprobenartige Überprüfen der Software. Dafür müssen die Randbedingungen vor dem Testen festgelegt werden. Um die korrekten Eigenschaften der Software überprüfen zu können, muss ein Soll-/Istvergleich durchgeführt werden.

Nach Spillner & Linz (Spillner & Linz, 2010) verfolgt Testen mehrere Ziele:

- Ausführung des Programms mit dem Ziel, Fehlerwirkungen nachzuweisen
- Ausführung des Programms mit dem Ziel, die Qualität zu bestimmen
- Ausführung des Programms mit dem Ziel, Vertrauen in das Programm zu erhöhen
- Analysieren des Programms oder der Dokumente, um Fehlerwirkungen vorzubeugen

1.1.2 Testprinzipien

Das Testen im Allgemeinen hat einen psychologischen Charakter, da die Tester meistens Menschen sind. Es gibt einige Testprinzipien, die in diesem Zusammenhang beachtet werden sollten. In der folgenden Tabelle sind einige Prinzipien nach Myers, Sandler & Badgett (Myers, Sandler, & Badgett, 2012) angegeben.

Tabelle 1 Vital Program Testing Guidelines (Myers, Sandler, & Badgett, 2012)

Principle Number	Principle
1	A necessary part of a test case is a definition of the expected output or result.
2	A programmer should avoid attempting to test his or her own program.
3	A programming organization should of test its own programs.
4	Any testing process should include a thorough inspection of the results of each test.
5	Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected.
6	Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does what it is not supposed to do.
7	Avoid throwaway test cases unless the program is truly a throwaway program.
8	Do not plan a testing effort under the tacit assumption that no errors will be found.
9	The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.
10	Testing is an extremely creative and intellectually challenging task.

1.1.3 Testprozess

Der Testprozess richtet sich nach dem Vorgehensmodell in der Softwareentwicklung. Am Anfang der Produktentwicklung wurde das allgemeine V-Modell angewendet. Das Modell beschreibt die Vorgehensweise bei der Entwicklung von Software und koppelt diese Entwicklungsstufen mit den dazugehörigen Teststufen (siehe Abbildung 1). Der Vorteil dieses Modells ist, dass beispielsweise die Systemanforderungsanalyse mit einem geeigneten Test validiert wird, hier der Akzeptanztest (siehe Abbildung 1 oben). Somit können Fehler auf dieser Entwicklungsstufe frühzeitig erkannt und behoben werden. Das Modell hat auch Nachteile, die sich aus der Praxis ergeben:

- Die Anforderungen an ein neues System sind zu Beginn nie vollständig bekannt.
- Es gibt für eine Anforderung unterschiedliche Realisierungsmöglichkeiten.
- Es gibt für manche Anforderungen keine Garantie, dass diese so realisiert werden können.

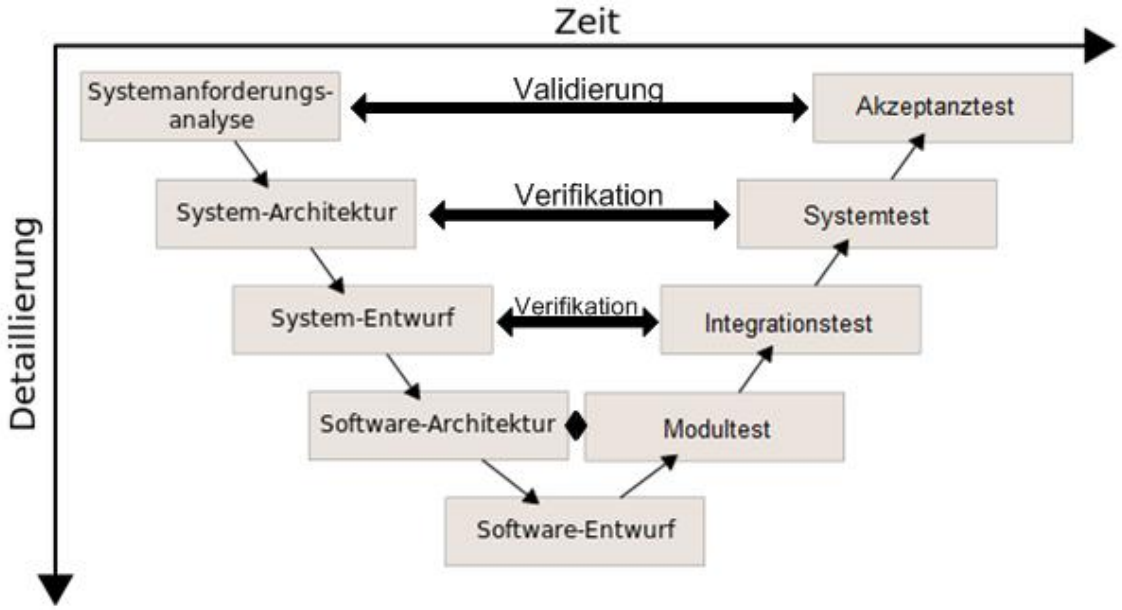

Abbildung 1: V-Modell nach Boehm 1979 (Pätzold & Seyfert, 2010)

Vor kurzem wurde der Entwicklungsprozess auf agile Softwareentwicklung nach Scrum umgestellt. Hierdurch ergeben sich neue Anforderungen an das Testsystem. Die Testzyklen werden kürzer, dafür wird der Umfang neuer Funktionen pro Release geringer.

1.1.4 Definition eines Fehlers

Ein Fehler ist die Nichterfüllung einer festgelegten Anforderung, eine Abweichung zwischen dem Ist-Verhalten (während der Ausführung der Tests oder des Betriebs festgestellt) und dem Sollverhalten (in der Spezifikation oder den Anforderungen festgelegt) (Spillner & Linz, 2010). Dabei ist zwischen einem Fehler und einem Mangel zu unterscheiden. Ein Mangel liegt dann beispielsweise vor, wenn die Anforderung zwar erfüllt

ist, aber die Funktion durch die Performance beeinträchtigt wird. Zum Beispiel: Das Speichern dieses Dokuments dauert, statt einigen Sekunden, mehrere Stunden. Dann ist die Funktion zwar vorhanden, aber wird durch das lange Warten unbrauchbar. Somit liegt ein Mangel vor.

1.1.5 Ursachenkette für Fehler

Anders als bei Hardwaresystemen, wo Fehler durch Alterung und Verschleiß auftreten, entstehen Fehler in Softwaresystemen beim Erstellen. Diese Softwarefehler kommen in der Regel erst bei der Ausführung zum Tragen.

1.1.6 Fehlerwirkung / Fehlerzustand

Beim Testen von Software können nur Fehlerwirkungen festgestellt werden. Eine Fehlerwirkung kann sich als falsche Anzeige eines Ausgabewertes oder das Abstürzen des Programms äußern. Die Fehlerwirkungen resultieren aus einem Fehlerzustand. Ein Fehlerzustand entsteht, wenn eine Anweisung im Programm falsch programmiert oder vergessen wurde. Den Fehlerursprung zu lokalisieren und den Fehlerzustand zu beheben ist Aufgabe der Entwickler.

1.1.7 Fehlermaskierung

Unter Fehlermaskierung versteht man die Überdeckung eines Fehlers durch einen oder mehrere andere Fehler. Ein einfaches Beispiel soll dies verdeutlichen.

Ein Programm berechnet die Summe aus zwei Zahlen. Dazu stellt es dem Benutzer zwei Eingabefelder (für jede Zahl ein Eingabefeld) zur Verfügung. Durch einen Fehlerzustand, ist die Eingabe der zweiten Zahl nicht möglich (Das Eingabefenster lässt nur Buchstaben zu). Die Fehlerwirkung zeigt an, dass die Summe aus einer Zahl und einem Buchstaben zum falschen Ergebnis führt. Nachdem der Fehler durch den Entwickler behoben wurde und nun Zahlen im zweiten Eingabefeld zugelassen werden, wird das Ergebnis von 2 + 2 = 5 ausgegeben. Das Programm funktioniert immer noch nicht richtig. Der zweite Fehler konnte aber nicht entdeckt werden, da er vom ersten Fehler überdeckt wurde. Korrekturen können somit zu Seiteneffekte führen (side effect).

1.1.8 Testverfahren

White Box

Das Ziel des White Box Verfahrens ist die einmalige Ausführung aller Quellcodeteile. Dazu muss der Programmtext zur Verfügung stehen. Die Tests können sich dabei auf unterschiedliche Kriterien beziehen. Diese Kriterien können folgende sein:

- Anweisungen

- Zweig- oder Entscheidungen
- Bedingungen→ Einfach, Mehrfach, Definiert
- Pfade

Die Einsatzgebiete der White Box-Verfahren sind die Komponenten- und Integrationstests. Hierfür werden tiefere Kenntnisse des Testers benötigt. Bei sicherheitskritischen Anwendungen ist es sinnvoll, das Testen der Anwendung durch ein externes Testteam durchführen zu lassen und nicht durch den Entwickler selbst. Unvoreingenommene Personen haben eine andere Sichtweise auf die Anwendung und können so Fehler identifizieren, die ein Entwickler nicht findet. Ob sich die längere Einarbeitungszeit von Außenstehenden rechnet, muss vom Management von Fall zu Fall unterschieden werden.

Black Box

Beim Black Box-Verfahren wird die Software getestet, ohne dass Kenntnisse der internen Struktur von Nöten sind. Durch Eingaben von außen wird das Testobjekt beeinflusst. Das Verhalten des Testobjekts wird dann anhand der Spezifikation geprüft. Wie die Informationen im System weiterverarbeitet werden, ist nicht bekannt und auch nicht wichtig. Dieses Testverfahren eignet sich beispielsweise für Systemtests und Komponententests. Des Weiteren ermöglicht das Verfahren, testgetriebene Softwareentwicklung. D.h. die Testfälle können vor der eigentlichen Entwicklung der Software erstellt werden. Dadurch werden die Fehler bei der Programmierung direkt erkannt und können behoben werden.

Ein Nachteil dieses Testverfahrens ist, das Fehler die gefunden werden, nicht genau lokalisiert werden können. In welcher Komponente der Fehler auftritt ist nicht erkennbar. Des Weiteren kann nicht festgestellt werden ob die Anforderungen und Spezifikation selbst Fehler enthalten. Wenn es um sicherheitsrelevante Software geht, können fälschlicherweise zusätzliche Funktionen, die nicht in der Spezifikation aufgeführt sind, enthalten sein. Diese können mit dem Black Box-Verfahren nicht herausgefunden werden.

1.2 Grundlagen Testautomatisierung

1.2.1 Was ist unter Testautomatisierung zu verstehen?

Testautomatisierung ist die Durchführung von ansonsten manuellen Testtätigkeiten durch Automaten. Diese Definition zeigt sowohl die mögliche Bandbreite als auch die gegebenen Grenzen der Testautomatisierung auf. Das Spektrum umfasst alle Tätigkeiten zur Überprüfung der Softwarequalität im Entwicklungsprozess, in den unterschiedlichen Entwicklungsphasen und Teststufen sowie die entsprechenden Aktivitäten von

Entwicklern, Testern, Analytikern oder auch der in die Entwicklung eingebundenen Anwender. Die Grenzen der Automatisierung liegen darin, dass diese nur die manuellen Tätigkeiten eines Testers übernehmen kann, nicht aber die intellektuelle, kreative und intuitive Dimension dieser Rolle. Diese Dimension ist jedoch maßgeblich für die Qualität des Softwaretests selbst. (Richard Seidl, 2012)

1.2.2 Nutzen und Aspekte für die Testautomatisierung[1]

Die Triebfeder für die Entscheidung, Testautomatisierung umzusetzen, ist der erzielte Nutzen. Folgende Vorteile durch Testautomatisierung sind meist gegeben:

1.2.2.1 Kosten –Aufwandsreduktion

- Die initiale Testerstellung amortisiert sich nach wenigen Regressionstests.
- Aufwand der wiederkehrenden Testdurchführung ist gut abschätzbar.
- Es ist nur geringer bzw. kein Personalaufwand für die Regressions-Testdurchführung notwendig.

1.2.2.2 Kritische Zeitpfade entschärfen

- Die Ausführung einer hohen Anzahl an Tests ist in kurzer Zeit möglich. Speziell in kritischen Projektphasen ist dieser Zeitvorteil eminent wichtig.
- Die Ausführung läuft über Nacht oder an Wochenenden. Dies führt zu einer Reduktion von Ressourcenengpässen.
- Experten werden nur noch für die Vorbereitung der automatisierten Tests, Behebung von fehlerhaften Tests und die Testauswertung benötigt.

1.2.2.3 Reproduzierbarkeit und Dokumentensicherheit

- Reproduzierbarkeit der Testdurchführung unter identischen Ausgangsbedingungen.
- Es erfolgt meist eine automatische Protokollierung der Test-Ergebnisse.

1.2.2.4 Erhöhung der Test- und Produktqualität

- Es entsteht eine höhere Qualität in der Testdurchführung und damit die Vermeidung von Fehlern durch Monotonie.

1.2.2.5 Ermöglicht neue Entwicklungsmethoden

- Iterative und agile Entwicklung sind erst mit Testautomatisierung vernünftig realisierbar.

[1] (Software Quality Lab GmbH, 2009)

2 Problemstellung

In einer großen Leitstelle können bis zu dreißig Disponenten gleichzeitig arbeiten. Das bedeutet, dass dreißig Notrufe zur selben Zeit angenommen und bearbeitet werden können. Damit summiert sich die Anzahl der Nachrichten, die über die „BOSPORUS TIP“-Schnittstelle übertragen werden, enorm. Die Schnittstelle beherrscht mehr als 400 Funktionen und jede Funktion kann mit anderen Funktionen kombiniert werden. Dadurch entstehen hunderte von Szenarien, die getestet werden müssen.

Des Weiteren gibt es keine grafische Benutzeroberfläche (GUI), mit der sich die Szenarien abbilden lassen. Die Kommunikation zwischen Einsatzleitsystem und „BOSPORUS TIP“ erfolgt über Nachrichten im XML Format. Stellt ein Einsatzleitsystem eine Anfrage (Request), erfolgt darauf die Antwort des Systems in Form von Events. Diese Events liefern die gewünschten Informationen. Ob eine Funktion korrekt ausgeführt wurde, kann nur durch Sichtprüfung der Parameter im Event festgestellt werden. Dazu muss nach jedem Empfangen einer Nachricht die Log Datei ausgewertet werden.

3 Ziele der Arbeit

Das Notrufabfragesystem „NORUMAT TIP“ kommuniziert über die „BOSPORUS TIP“-Schnittstelle mit verschiedenen Einsatzleitsystemen (ELS). Derzeit werden die Tests der Schnittstelle manuell mit Hilfe einer ELS-Simulation (BsptTool) durch händische Eingabe der XML-Steuerstrukturen und ebenfalls händischer Auswertung der Ergebnisse durchgeführt.

Diese Tests sollen automatisiert werden, um den Zeitaufwand eines Testdurchlaufs erheblich zu verkürzen. Dabei sollen reale Szenarien des Kunden in XML-Sequenzen umgesetzt werden, welche dann als Testfall durchlaufen werden. Die derzeit händischen Interaktionen sowohl am ELS-Simulator als auch am Notrufabfragesystem sollen mit dem Testautomatisierungstool „QF-Test“ (näheres dazu in Abschnitt 7.1) automatisiert werden. Als Ergebnis eines Testdurchlaufs soll ein Report über die erfolgreiche beziehungsweise nicht erfolgreiche Durchführung des Tests automatisch erstellt werden.

4 Stand der Technik

4.1 „NORUMAT TIP“ System

Das „NORUMAT TIP“ System besteht aus vielen unterschiedlichen Subsystemen (siehe Abbildung 2)

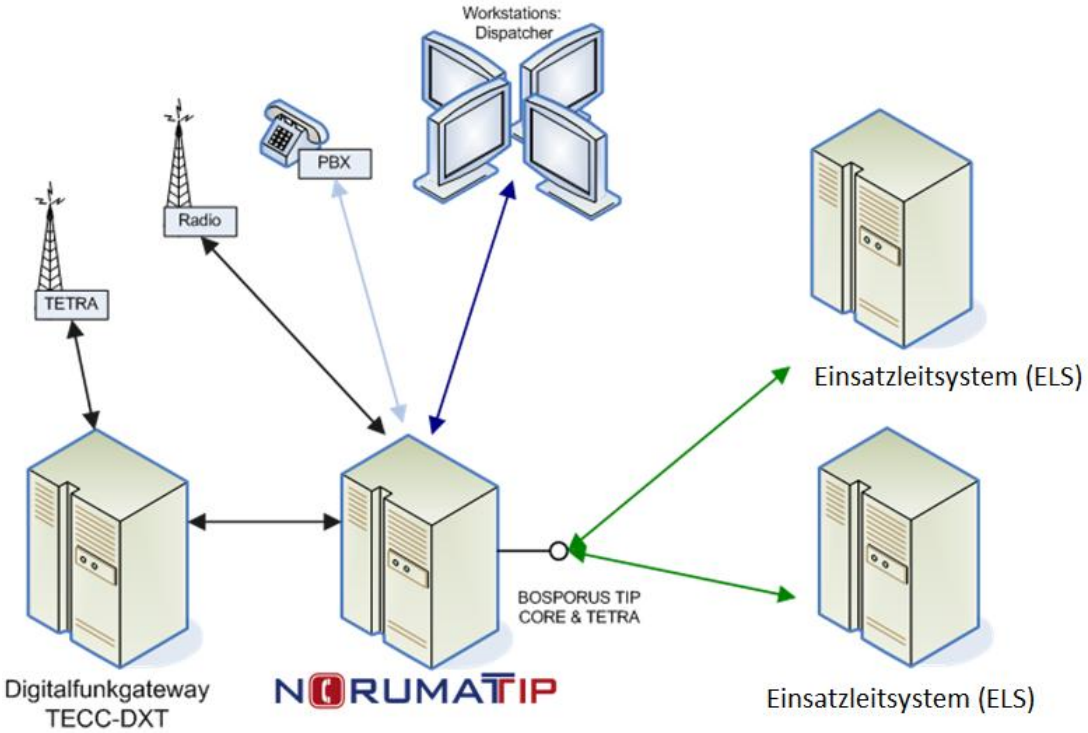

Abbildung 2: „NORUMAT TIP“ System Übersicht

Die „Workstations“ ermöglichen es dem Disponenten sämtliche Aktionen am System durchzuführen. Es können darüber beispielsweise Telefongespräche geführt werden. Die angeschlossenen analogen und digitalen Funkgeräte ermöglichen dem Disponenten zu Feuerwehren oder Polizeistationen Verbindung aufzunehmen. Eine Workstation besteht aus Mikrofonen, Headset, Handapparat, Betriebslautsprechern, Mithörlautsprechern, Monitoren und einem PC mit Maus und Tastatur. Die grafische Benutzeroberfläche wird von THALES selbst entwickelt.

Das „NORUMAT TIP“ System wurde modular aufgebaut und erlaubt dadurch die Anbindung anderer Systeme. Über die „BOSPORUS TIP“ Schnittstelle können verschiedene Einsatzleitsysteme angebunden werden. Diese Systeme übernehmen die Koordination und Planung von Rettungseinsätzen.

4.2 BOSPORUS TIP Schnittstelle[2]

Die „BOSPORUS TIP“-Schnittstelle stellt den Einsatzleitsystemen eine Steuer- und Informationsschnittstelle zur Verfügung. Die Schnittstelle bietet die Möglichkeit Informationen über den Systemzustand des „NORUMAT TIP“ zu erhalten sowie Anfragen an den „NORUMAT TIP“ zu senden.

4.2.1 Philosophie

4.2.1.1 Was soll BOSPORUS TIP leisten?

„BOSPORUS TIP“ soll Möglichkeiten, die ein Anwender über die grafische Benutzerschnittstelle hat, für angeschlossene Anwendungen zur Verfügung stellen.

4.2.1.2 Was soll BOSPORUS TIP nicht leisten?

„BOSPORUS TIP“ soll keine Brücke zu anderen Schnittstellen sein. Es ist nicht vorgesehen, Zugriffe auf spezielle Komponenten (z. B. Dokumentationssystem) über „BOSPORUS TIP“ zu ermöglichen.

4.2.2 Kommunikation

Grundsätzlich besteht die Kommunikation aus zwei Grundtypen von Nachrichten, Ereignissen (Events) und Anfragen (Requests). Events werden vom „NORUMAT TIP“ oder dem Einsatzleitsystem für Zustandsänderungen, Systemereignisse und an bestimmten Punkten der Verarbeitung versandt. Beim Keep-Alive-Mechanismus sendet der „NORUMAT TIP“ Requests, um zu sehen, ob die Gegenstelle noch da ist. Das bedeutet es werden sowohl Requests als auch Events zwischen „NORUMAT TIP“ und Einsatzleitsystem (ELS) bidirektional über „BOSPORUST TIP“ ausgetauscht (siehe Abbildung 3).

Ein Event besteht grundsätzlich aus einem XML-Header, der DOCTYPE Definition, einem öffnenden Tag, das den Sender des Events ausweist (z. B. BosporusTipEvent), einem öffnenden Tag für die Art des Events (z. B. CallEventType), dem eigentlichen Inhalt in Form von XML Beschreibungen der logischen Objekte und den jeweiligen schließenden Tags.

Requests sind sehr ähnlich aufgebaut. Sie bestehen ebenfalls aus einem XML-Header, der DOCTYPE Definition, einem öffnenden Tag, das in diesem Fall aber den Empfänger des Requests ausweist (BosporusTipRequest), einem öffnenden Tag für die Art des Requests (z.B. CallRequest), dem eigentlichen Inhalt in Form von XML Beschreibungen der Anfragen inklusive Parameter und den jeweiligen schließenden Tags.

[2]Abschnitt enthält Teile aus dem internen IDD Dokument von THALES (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013)

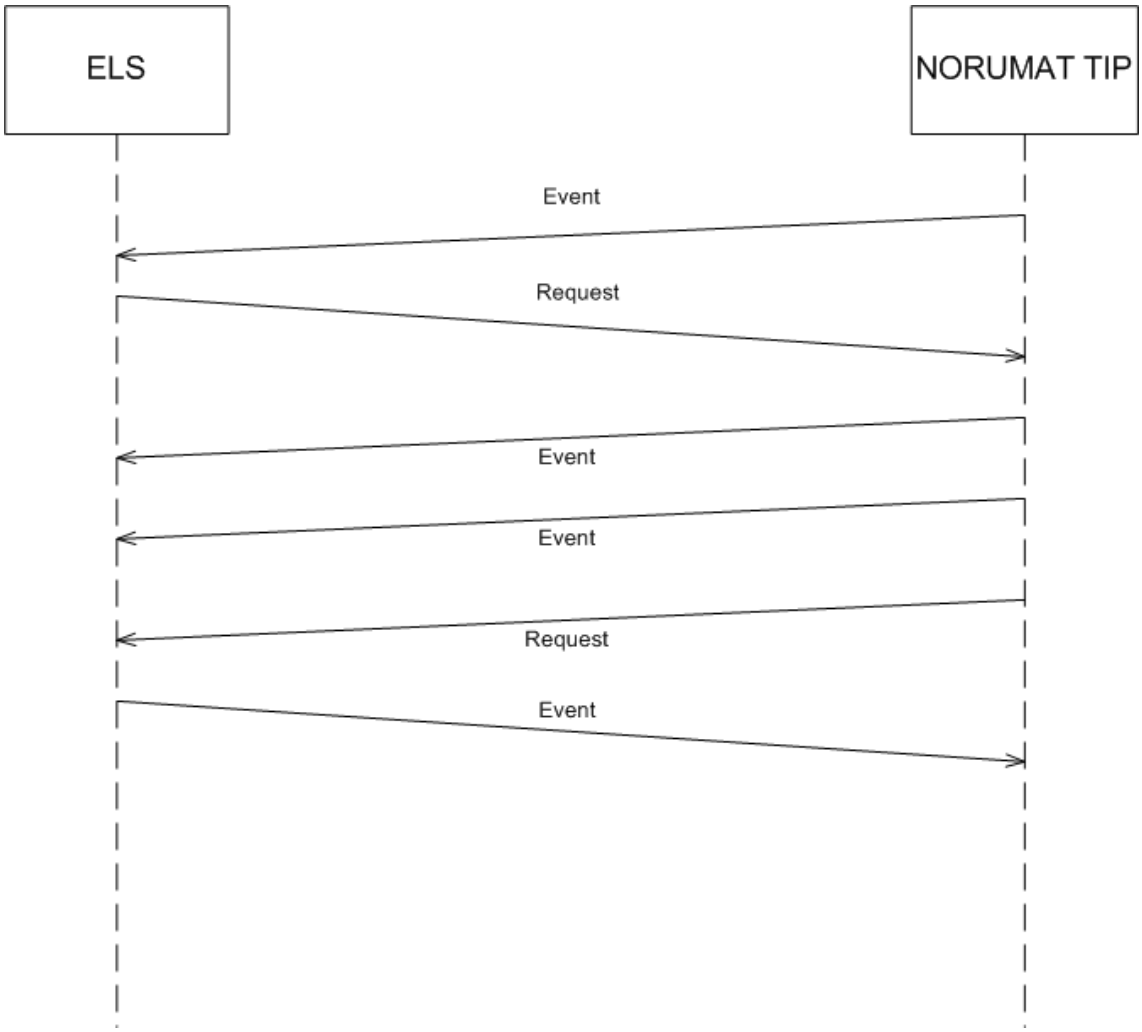

Abbildung 3: Allgemeine Kommunikation (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013)

4.3 Testprozedur bisher

Die „BOSPORUS TIP“-Schnittstelle wird mit dem BsptTool semiautomatisiert getestet. Es gibt für die einzelnen Testfälle jeweils ein Testskript. Über das BsptTool wird das Skript gestartet. Das Skript startet zuerst die Trace Funktion des Werkzeugs. Damit wird sämtlicher Datenverkehr zwischen BsptTool und „BOSPORUS TIP“-Schnittstelle geloggt und in eine Datei geschrieben. Der Pfad und der Dateiname kann über ein Eingabefeld in der Tracing Maske angegeben werden. Nach dem Start des Tracings arbeitet das Skript Schritt für Schritt das Testszenario ab. Nach bestimmten Schritten hält das Skript an und erwartet vom Tester eine Bestätigung. Dabei stehen dem Tester vier Buttons (yes, no, ok, cancel) im rechten unteren Bereich des Werkzeugs zur Verfügung, um einzelne Zwischenergebnisse zu bestätigen. Ist das Skript beendet, kann der Tester sich die Log Datei anschauen und überprüfen ob alle Parameter richtig übertragen wurden.

Diese Tests benötigen mindestens zwei Tester, damit die Richtigkeit der Testergebnisse verifiziert werden kann. Des Weiteren ist die Erstellung der Testskripte umständlich und unflexibel. Damit die einzelnen XML –Request mit den richtigen Parametern gesendet werden können, müssen diese aus einer Konfigurationsdatei ausgelesen und eingefügt werden. Einfache Änderungen an den Parametern aus dem Skript heraus sind nicht möglich.

5 Lösungsvarianten

Man verspricht sich von der Testautomatisierung der Schnittstelle die Reduktion des Testaufwands. Des Weiteren sollen Fehler schneller identifiziert und behandelt werden können. Die Kostenersparnis während des Testens runden die Vorzüge der Testautomatisierung ab. Zwei mögliche Lösungen haben sich nach der Recherche herauskristallisiert.

5.1 Komplett neues Konzept entwickeln

Die erste Variante verfolgt das Ziel, ein komplett neues Konzept zu entwickeln. Das bedeutet die Kommunikation zwischen „BOSPORUS TIP“ und Client, die Erstellung der Testszenearien, die Auswertung der Testszenarien, Erstellung der Protokolle muss konzipiert und umgesetzt werden. Die Lösung soll mit Skripten abgebildet werden. Die Programmiersprache ist nicht vorgegeben (bevorzugt wird aber Java oder Java Ähnliches).

5.2 Verwenden was vorhanden ist

Die zweite Variante verfolgt den Ansatz alle vorhandenen Werkzeuge so zu verbinden, dass die Aufgabe gelöst wird. Dabei übernimmt das „BOSPORUS TIP“ Werkzeug die Aufgabe, die Kommunikation zwischen „BOSPORUS TIP“ und Client zu verwalten. „QF-Test“ ermöglicht das Erstellen und Verwalten von Testszenarien sowie protokollieren der Ergebnisse. Dabei nimmt „QF-Test“ in diesem Zusammenhang eine zentrale Rolle ein. Der komplette Testablauf wird über dieses Programm gesteuert.

6 Gewählte Lösungsvariante

Die Lösung 5.2 hat folgende Vorteile:

- Teilprobleme bereits gelöst (Verbindung zwischen Server und Client)
- Fokus liegt auf dem Hauptproblem (Testautomatisierung entwickeln)
- Grafische Benutzeroberfläche zum Erstellen von Testszenarien
- Wartbarkeit durch bewährte Tools erhöht
- Erleichterte Fehleranalyse möglich

Die Kommunikation zwischen BsptTool und „NORUMAT TIP“ ist bereits vorhanden und muss nicht neu entwickelt werden. Die Entwicklung eines Testkonzepts und die konkrete Realisierung stehen im Vordergrund. Das Programm „QF-Test“ erleichtert die Erstellung und Wartbarkeit der Tests.

Die Lösungsvariante 1 (siehe Abschnitt 5.1) ist für den kurzen Zeitraum nicht vollständig zu realisieren. Man fängt bei dieser Lösung auf der grünen Wiese an. Es gibt keinerlei Vorgaben, außer dass eine Testautomatisierung realisiert werden soll. Der Aufwand, um die Aufgabe mit dieser Variante zu lösen, wäre viel zu hoch. Deshalb wurde die Lösungsvariante zwei (siehe Abschnitt 5.2) gewählt.

7 Verwendete Werkzeuge

7.1 QF-Test

„QF-Test“ ist ein Testautomatisierungstool der Firma QFS-Quality First Software. Die Software bietet die Möglichkeit, grafische Benutzeroberflächen, die in Java geschrieben sind, automatisiert zu testen. Dabei werden die Testszenarien mit einer grafischen Benutzeroberfläche erstellt. Damit die grafische Oberfläche eines Programms getestet werden kann, muss das zu testende Programm über „QF-Test“ gestartet werden. Der wesentliche Vorteil von „QF-Test“ im Vergleich zu anderen „Capture and Replay“ Werkzeugen ist, dass „QF-Test“ auf die Komponenten der zu testenden Anwendung Zugriff hat. Die grafische Benutzeroberfläche (GUI) dieser Anwendungen kann aus vielen Schichten, sogenannte Layer bestehen. In jedem Layer können Elemente wie Labels, Textfelder, Buttons uvm. untergebracht sein. Da „QF-Test“ vollen Zugriff auf diese Komponenten hat, können diese jederzeit ausgewählt werden, auch wenn sie durch andere Fenster verdeckt werden. Es spielt auch keine Rolle, wo sich die Elemente auf dem Bildschirm befinden. Sollten sich im Entwicklungszyklus der Software Elemente verschieben, muss der Test dafür in „QF-Test“ nicht angepasst werden. Dies erleichtert die Wartung und macht die Tests robust gegen Änderungen der Software. Die konkrete Vorgehensweise beim Erstellen von Testfällen mit „QF-Test“ wird in Abschnitt 9 genauer erläutert.

Im folgenden Abschnitt werden die wichtigsten verwendeten Funktionen von „QF-Test“ beschrieben. Eine vollständige Erläuterung sämtlicher verwendeter Funktionen können aus dem Benutzerhandbuch von „QF-Test“ entnommen werden (QFS, 2014). Jede Funktion wird durch ein entsprechendes Symbol dargestellt. Dadurch wird die Funktionsweise hervorgehoben und erleichtert somit die Entwicklung. Die folgenden Funktionsknoten können über das Kontextmenü aufgerufen werden. Abhängig davon auf welcher Hierarchieebene man sich befindet, stehen mehr oder weniger Funktionsknoten zur Verfügung.

7.1.1 Testsuite-Knoten

Die Testsuite bildet den Wurzelknoten des Baums. Seine grundlegende Struktur ist fest definiert. Der Wurzelknoten (siehe Abbildung 4) enthält eine beliebige Anzahl von „Testfall“- und „Testfallsatz“-Knoten, gefolgt von den Prozeduren, den Extrasequenzen und dem Fenster und Komponenten Knoten. Die Knoten der obersten Ebene werden der Reihe nach ausgeführt. (QFS, 2014)

Abbildung 4: Wurzelknoten einer QF-Testsuite

7.1.2 Prozeduren

Eine Prozedur (siehe Abbildung 5) ist eine Sequenz, die mittels eines Prozeduraufrufs von einer beliebigen anderen Stelle aus aufgerufen werden kann. In einer Prozedur werden häufig wiederkehrende Bestandteile einer Testsuite gekapselt. Die Parameter der Prozeduren werden nicht explizit angegeben, sondern ergeben sich durch die Variablenreferenzen der Kinder der Prozedur. (QFS, 2014)

Abbildung 5: Symbolbild Prozeduren

7.1.3 Extrasequenzen-Knoten

Der Knoten Extrasequenzen (siehe Abbildung 6) dient als eine Art Spielwiese zum Ausprobieren und Zusammenstellen von Tests. Hier können beliebige Knoten abgelegt werden. Alle Sequenzen, die neu aufgenommen wurden, landen in diesem Knoten. Auch zur Zwischenablage ist dieser Knoten geeignet (QFS, 2014).

Abbildung 6: Symbolbild Extrasequenz

7.1.4 Fenster und Komponenten

Der Fenster und Komponenten Knoten (siehe Abbildung 7) enthält alle Fenster der zu testenden Software. Die Komponenten werden von „QF-Test“ über ein Merkmal erkannt. Dieses Merkmal kann über eine Variable geändert werden. Die Variable wird der Prozedur übergeben und ermöglicht es dadurch, dass verschiedene Einträge eines Menüs individuell angesprochen werden können.

Abbildung 7: Symbolbild Fenster und Komponenten

7.1.5 Sequenzknoten

Diese einfachste Form aller Sequenzen (siehe Abbildung 8) hat beliebig viele Kind-Knoten, die sie der Reihe nach abarbeitet (QFS, 2014). Der Name einer Sequenz sollte beschreiben, welche Funktion die Sequenz übernimmt.

Abbildung 8: Symbolbild Sequenzknoten

7.1.6 Ablaufsteuerungsknoten

„QF-Test“ unterstützt folgende Ablaufsteuerungen: If, Elseif, Else (siehe Abbildung 9), Try, Catch, Finally (siehe Abbildung 10), While-Schleife, Normale Schleifen, Break (siehe Abbildung 11). Zu den Ablaufsteuerungsknoten zählen auch die Skriptknoten (siehe Abbildung 12). Hier stellt „QF-Test“ zwei zur Verfügung, den Serverskriptknoten und SUT Skriptknoten. (QFS, 2014). Mit diesen Kontrollstrukturen können komplexe Sequenzen realisiert werden.

Abbildung 9: Symbolbild If-Konstrukte (If, Elseif, Else)

Abbildung 10: Symbolbild Try, Catch, Finally

Abbildung 11: Symbolbild Schleifen (While, Normal) und Break

Abbildung 12: Symbol Skript in „QF-Test“

7.1.7 Programmknoten

Der Programmknoten ermöglicht das Ausführen von Programmen, Shellkommando´s oder auch speziellen Warten Knoten (siehe Abbildung 13). Der SUT Client Knoten beispielsweise startet die zu testende Software. Des Weiteren ist es möglich, weitere Programme zu starten.

Abbildung 13: Symbolbild Programmknoten

7.1.8 Nützliche Funktionen

In „QF-Test“ gibt es die Möglichkeit, einzelne Knoten zu konvertieren. So kann eine Sequenz bequem in eine Prozedur umgewandelt werden. Des Weiteren ist es möglich, durch markieren von einzelnen Testschritten, diese in eine Sequenz einzupacken. Somit lässt sich ein Testszenario leichter strukturieren. Diese Funktionen erhöhen den Bedienkomfort beträchtlich.

7.1.9 Protokoll

Das Testprotokoll wird von „QF-Test“ am Ende des Testdurchlaufs automatisch erstellt (siehe Abbildung 14). Das Protokoll ist in zwei Spalten gegliedert. Die rechte Spalte zeigt allgemeine Informationen zum Testdurchlauf an. Zum Beispiel die Testlauf ID, Uhrzeit und Dauer des Testdurchlaufs, Anzahl Exceptions, Fehler und Warnungen, Umgebungsvariablen: Hostname, Betriebssystem Version, Benutzer, Java Version, „QF-Test“ Version und welche Testsuite ausgeführt wurde. Die linke Spalte des Protokolls enthält die einzelnen Knoten des jeweiligen Testdurchlaufs. Fehlerhafte Knoten werden durch ein rotes Kästchen gekennzeichnet. Warnungen werden durch ein gelbes Kästchen angezeigt. Das Protokoll ist wie die Testfälle aufgebaut und ermöglicht somit jeden Knoten zu überprüfen.

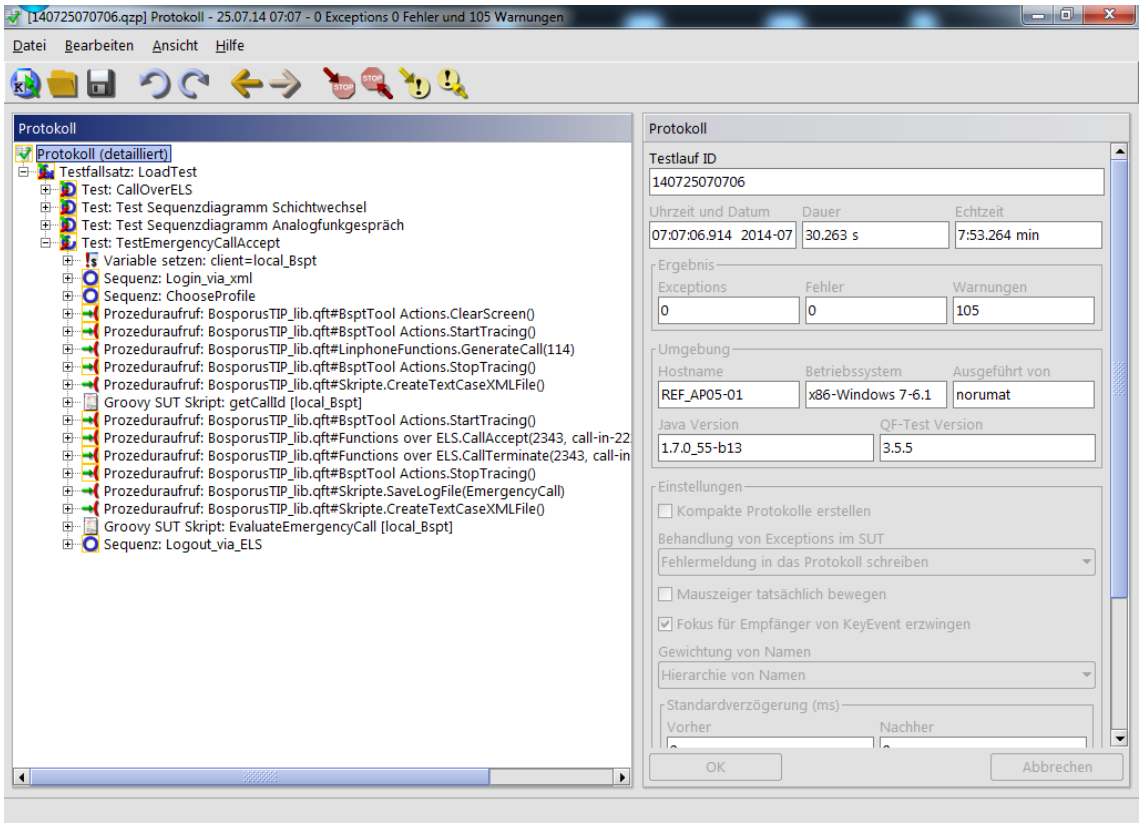

Abbildung 14: „QF-Test“ Protokoll Übersicht

Über das Runcontext (rc) Objekt stellt „QF-Test“ eine Möglichkeit zur Verfügung, Nachrichten in das Protokoll zu schreiben. Die Methoden rc.logMessage(String), rc.logWarning(String), rc.logError(String) stehen zur Verfügung. Mit rc.logMessage(String) können einfache Nachrichten in das Protokoll geschrieben werden. Die Methode rc.logError(String) hingegen erzeugt eine Fehlernachricht im Protokoll. Jede Nachricht erhält ein spezifisches Icon entsprechend des Nachrichtentyps. Anhand dieser Icons lassen sich Fehler, Warnungen und einfache Meldungen unterscheiden.

7.2 BsptTool

Das BsptTool wurde von THALES entwickelt. Es dient in erster Linie dazu, das Einsatzleitsystem zu simulieren. Das Programm besteht im Wesentlichen aus zwei Hälften (siehe Abbildung 15). Der Anzeige Bereich (Abbildung 15 / (1)) enthält alle Nachrichten, die zwischen BsptTool und „NORUMAT TIP“ ausgetauscht werden. In der unteren Hälfte liegt der Eingabe Bereich (Abbildung 15 / (2)). Dort können XML Request eingefügt und gesendet werden. Über das Programm kann man sich mit dem „NORUMAT TIP“-Server verbinden. Das Programm beherrscht mehrere nützliche Funktionen. Eine Funktion davon ist, dass sämtlicher Datenverkehr zwischen „BOSPORUS TIP“ Schnittstelle und „NORUMAT TIP“ protokolliert wird. Des Weiteren kann über ein Menü jede benötigte Nachricht abonniert werden. Der entscheidende Vorteil dieses Werkzeugs ist, dass es in Java programmiert wurde und daher von „QF-Test“ aus gesteuert werden kann.

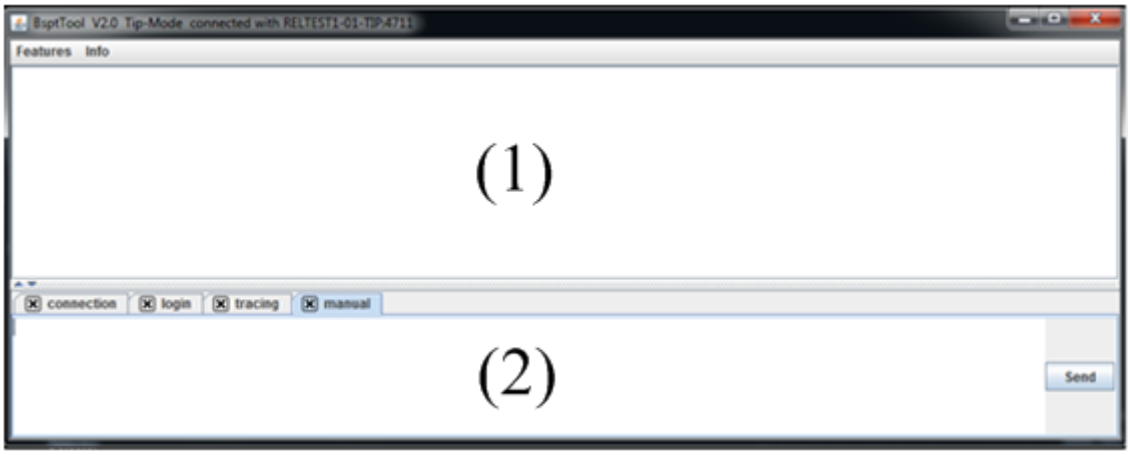

Abbildung 15: BsptTool Anzeige Bereich (1), Manuelle Eingabe Bereich (2)

7.3 Groovy Console

7.3.1 Was ist Groovy?

Die Skriptsprache Groovy erweitert Java um einige nützliche Funktionen, die beispielsweise den Umgang mit Strings wesentlich angenehmer gestaltet (siehe dazu Abschnitt 7.3.2). Beim Schreiben von Skripts mit Groovy liegt der Fokus auf kurzen, aber effektiven Programmcode. Damit die Skripts schnell zur Ausführung gebracht werden können, sind die Standard Bibliotheken schon integriert und müssen nicht explizit angegeben werden. Es muss auch keine Main-Klasse angelegt werden. Beispielsweise kann mit dem XML Parser von Groovy eine XML Datei in einer Zeile geparst werden (siehe Abbildung 16).

```
def rootObject = new XmlParser().parse(fileToParse)
```

Abbildung 16: Codezeile zum parsen einer Datei

Des Weiteren müssen in Groovy die Datentypen für Variablen nicht angegeben werden. Diese werden dynamisch von Groovy an die jeweilige Situation angepasst. Dazu muss nur vor die Variable das Schlüsselwort „def“ geschrieben werden. Wie oben erwähnt wird Java nur um Funktionalität erweitert, d.h. der volle Funktionsumfang von Java steht jederzeit zur Verfügung.

7.3.2 Closure und each-Methode

Eine Closure ist ein als Objekt verpacktes Codestück. Sie verhält sich wie eine Methode. Sie kann Parameter entgegennehmen und einen Wert zurückgeben und sie ist auch ein Objekt, sie kann wie jedes Objekt als Referenz übergeben werden. (König, Glover, King, Laforge, & Skeet, 2007). Eine Closure beginnt und endet mit einer geschweiften Klammer, ähnlich einem Anweisungsblock. Closures sind besonders nützlich bei der Arbeit mit Arrays, Listen und Maps. Das folgende einfache Beispiel soll dies verdeutlichen. (Abbildung 17)

```
Closure mit expliziter Parameterangabe
log = ''
(1..5).each{counter -> log+= counter}
assert log == '12345'

Closure mit Standardparameter "it"
log =''
(1..5).each{log+= it}
assert log == '12345'
```

Abbildung 17: Beispiel Closure deklarieren (König, Glover, King, Laforge, & Skeet, 2007)

Groovy fügt die zusätzliche Methoden (each, find, findAll, collect usw.) den vorhandenen Collection-Klassen hinzu, damit diese ständig zur Verfügung stehen.

Die im Beispiel verwendete each-Methode iteriert über das gesamte Array. Bei jedem Durchlauf wird der Inhalt an die „counter“-Variable übergeben. Im ersten Durchlauf wird die Eins an die „counter“-Variable übergeben, im zweiten Durchlauf die Zwei, im Dritten die Drei usw. In der Closure werden die einzelnen Variablenwerte verarbeitet, hier einfach der Variablen „log“ hinzugefügt. Die Trennung zwischen Iteration (each-Methode) und Verarbeitung (Closure) erleichtert den Umgang mit Arrays, Listen und Maps.

7.3.3 Programmbeschreibung

Groovy Console erleichtert das Erstellen der Testskripte. Das Programm ist so aufgebaut, dass im oberen Teil (1) der Programmcode eingegeben werden kann(siehe Abbildung 18). Im unteren Teil des Programms (2) werden die Compiler Ausgaben angezeigt. Die Tastenkombinationen Strg+S (Speichern), Strg+W (Ausgabe löschen), Strg+R (Skript ausführen) haben sich als sehr nützlich erwiesen. Die Dokumentation zu den Groovy Klassen steht online zur Verfügung. Das Programm unterstützt durch Highlighting und durch einfachen Einstieg.

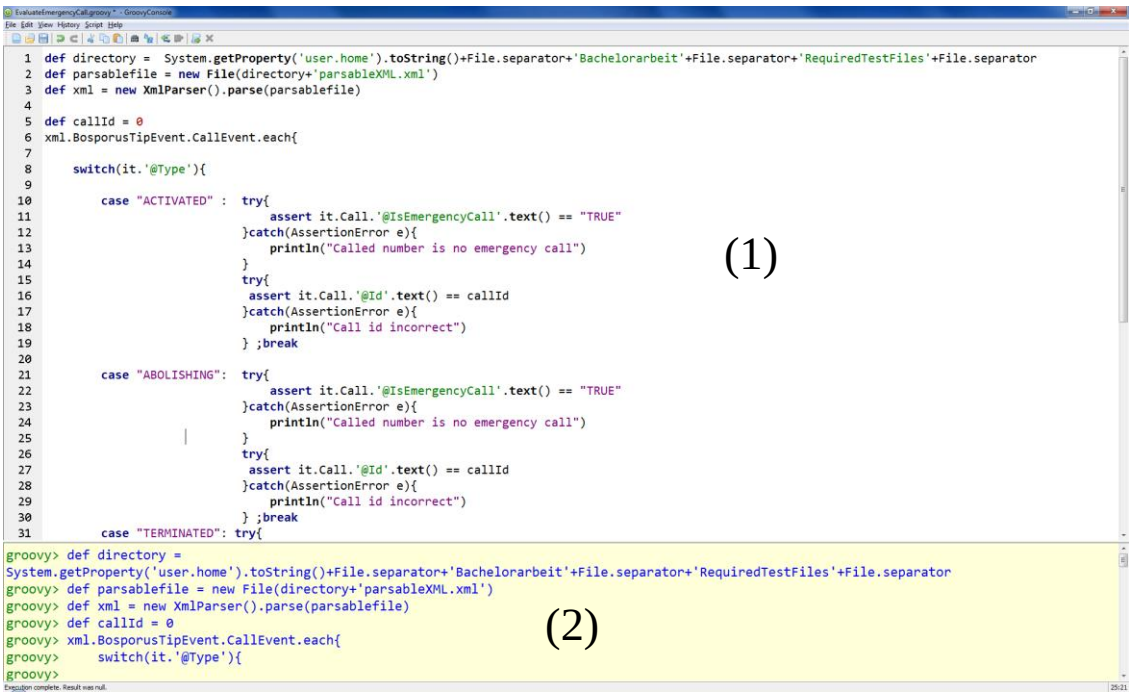

Abbildung 18: Screenshot Groovy Console

7.4 Linphone

Mit Linphone lassen sich Telefonanrufe über VoIP generieren. Dazu registriert man sich an der Telefonanlage des „NORUMAT TIP“ Servers. Damit das funktioniert muss in der Telefonanlage ein Benutzer angelegt sein. Unter den Einstellungen in der Linphone GUI muss die Firewall angepasst werden. Die Vorteile von Linphone sind, dass es frei verfügbar ist und über Command Befehle gesteuert werden kann

Nach dem Registrieren kann das Programm wie ein normales Telefon benutzt werden.

Im Testszenario übernimmt Linphone die Aufgabe, Notrufe und Amtsanrufe zu simulieren. Somit können auch Testszenarien erstellt werden, die mit Telefongesprächen in Verbindung stehen.

8 Konzeptdesign

Die Anforderungen bestimmen maßgeblich das Design. Die oben genannten Programme „QF-Test“, BsptTool wurden festgelegt und mussten verwendet werden. Dabei lag das Hauptaugenmerk auf „QF-Test“. Dieses Programm sollte die zentrale Rolle einnehmen. Somit ergibt sich das Konzept mit den zwei Hauptkomponenten Server und Testautomat (siehe Abbildung 19). Die Kommunikation zwischen den beiden Komponenten besteht aus einzelnen XML Nachrichten. Jeder Request wird mit entsprechenden Events beantwortet.

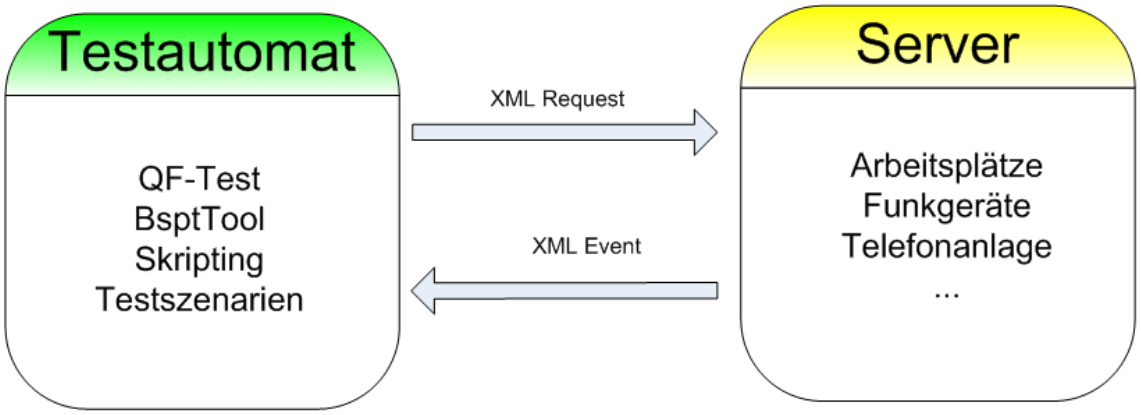

Abbildung 19: Konzeptdesign

Als nächstes wird der interne Aufbau des Testautomaten diskutiert (siehe Abbildung 20). Die Beschreibungen beziehen sich nur auf die Software des Servers. Alle Hardwaresysteme werden erklärt, wenn sie für das Konzept relevant sind. Zuerst betrachten wir den Serverteil. Dieser besteht aus den zwei Wolken und der „BOSPORUS TIP“ Schnittstelle. Die zwei Wolken verdeutlichen, dass die XML Requests und die resultierenden Events nicht von derselben Schicht bearbeitet werden. Dies ist von Bedeutung, denn dadurch können sich Zeitunterschiede ergeben, die im Test berücksichtigt werden müssen. Die „BOSPORUS TIP“ Schnittstelle liest die XML Requests und reicht die Daten an die dahinterliegenden Schichten weiter (siehe Abbildung 20 Wolke 1). Es kann daher keine Aussage gemacht werden, wie lange es dauern kann, bis ein Event gesendet wird. Damit Nachrichten empfangen werden können, müssen diese abonniert werden. Dies geschieht über das BsptTool (siehe Abschnitt 7.2). Sobald neue Informationen zur Verfügung stehen, werden diese, per Broadcast an die Abonnenten gesendet (siehe Abbildung 20 Wolke 2). In diesem Konzept werden die Nachrichten vom BsptTool empfangen, angezeigt und in eine Datei geschrieben. Die Logdatei enthält alle Requests und Events, die während eines Testszenarios über die Schnittstelle gesendet und empfangen wurden. Diese Datei lässt sich mit einem XML Parser nicht auslesen

und muss deshalb umgewandelt werden. Die Auswertung der XML Datei erfolgt am Ende jedes Testszenarios. Das Protokoll enthält alle Ergebnisse des Testszenarios in Form von Ereignismeldungen, Warnungen und Fehlern. Das Konzept sieht vor, dass das gesamte Testszenario über einen Startknopf gestartet wird. Der Tester muss am Ende des Tests nur bei einem Fehler das Protokoll analysieren. Für die Analyse von Fehlern wird jedem Testszenario zusätzlich die Logdatei angehängt.

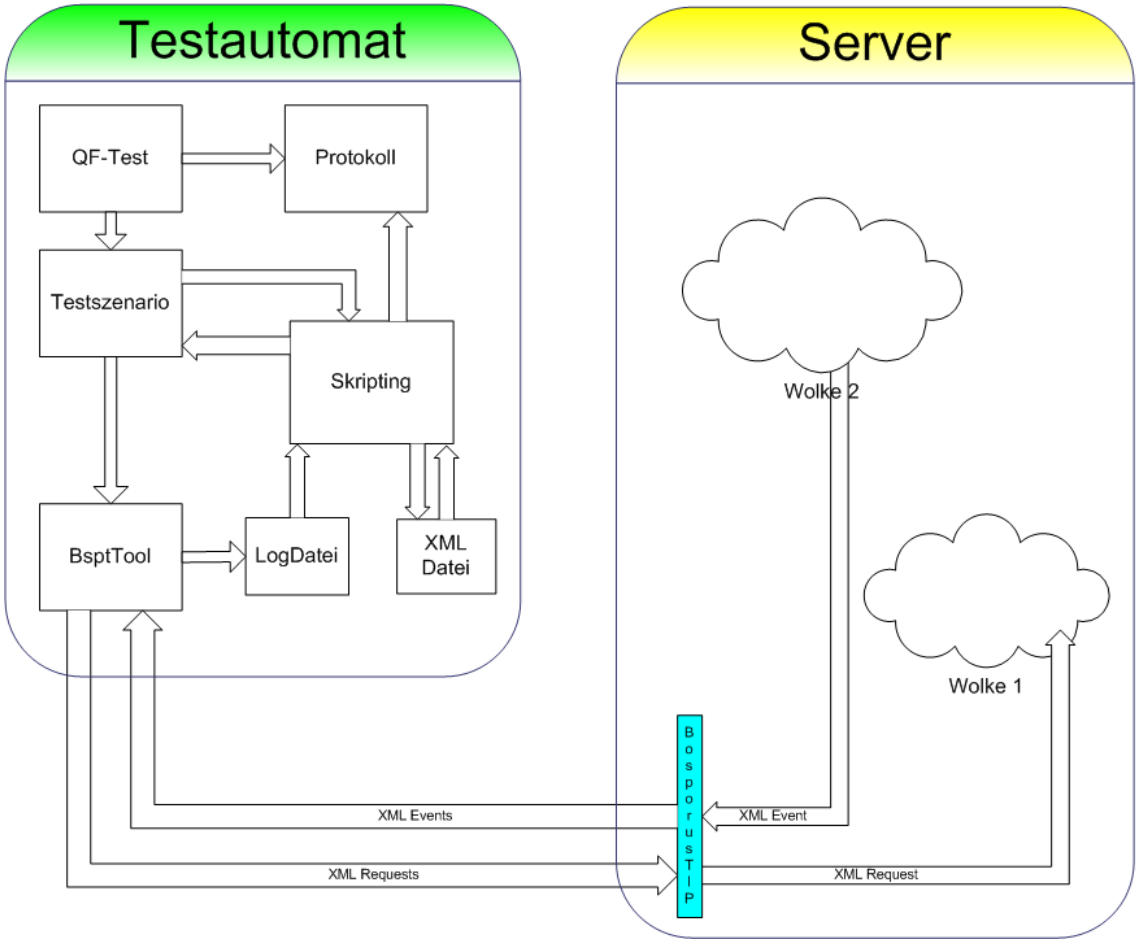

Abbildung 20: Konzeptdesign interner Aufbau

8.1 Risikoanalyse

Da nicht alle Funktionen getestet werden können, muss bewertet werden welche Funktionalitäten wichtig sind. Des Weiteren muss geklärt werden, welche Tests einfach und kostengünstig implementiert werden können und welche Tests kompliziert und teuer sind. Die Risikoanalyse hilft zu entscheiden, welche Funktionen getestet werden müssen und welche nicht (siehe Abbildung 21).

Um das Risiko zu verringern wurde das Prinzip ALARP[3] angewendet. Dabei wird die Funktion nach zwei Kriterien bewertet. Die Eintrittswahrscheinlichkeit gibt an, wie häufig eine Fehlersituation vermutlich eintritt. Das Schadensausmaß gibt an, wie hoch der Schaden im Fehlerfall ist. Für die Eintrittswahrscheinlichkeit ergeben sich sechs Klassen:

- Häufig
- Wahrscheinlich
- Gelegentlich
- Entfernt vorstellbar
- Unwahrscheinlich
- Unvorstellbar

Das Schadensausmaß wird in vier Klassen eingeteilt:

- Unwesentlich
- Geringfügig
- Kritisch
- Katastrophal

Die Funktionen „Profil anmelden“ und „Funkkreis belegen“ befinden sich im ALARP-Bereich, d.h. es müssen keine besonderen Maßnahmen getroffen werden. Die Funktion „Notruf“ befindet sich im inaktzeptablen Bereich. Es ist in diesem komplexen System nicht ausgeschlossen, dass Notrufe immer zu 100% abgearbeitet werden können. Tritt dieser Fall ein, so muss ein redundant aufgebautes Ersatzsystem diese Aufgabe übernehmen.

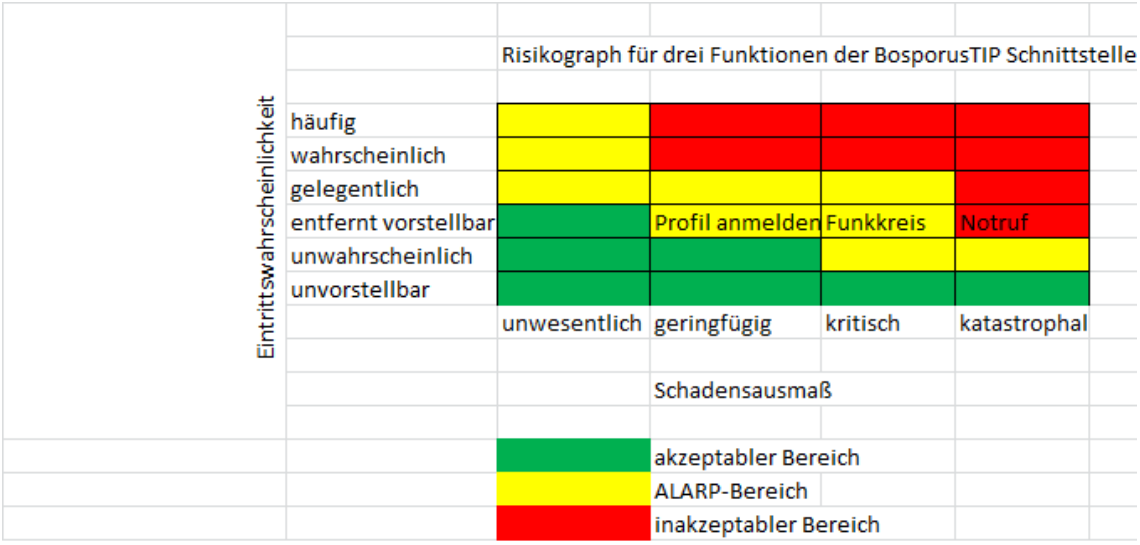

Abbildung 21: Risikograph nach ALARP: Die Risikoanalyse wurde für die drei Funktionen: Notruf annehmen, Funkkreis belegen und Profil anmelden durchgeführt

[3] **ALARP** ist ein englisches Akronym und bedeutet ***A**s **L**ow **A**s **R**easonably **P**racticable* (*so niedrig, wie vernünftigerweise praktikabel*). (Wikipedia, 2013)

8.2 Funktionsbibliothek in QF-Test

Damit die Testszenarien einfacher in „QF-Test“ implementiert werden können, wurde eine Funktionsbibliothek (auch Prozeduren-Bibliothek) erstellt. Diese Funktionsbibliothek ist in Packages organisiert (siehe Abbildung 22). Die einzelnen Packages enthalten so viele Prozeduren, wie in den Testszenarien benötigt werden (siehe Abbildung 23). Jede Prozedur übernimmt dabei eine kleine Aufgabe. Beispielsweise übernimmt die Prozedur „Dispatcher einloggen“ die Aufgabe einen Dispatcher am Server anzumelden. Funktionen die in den Testszenarien häufiger Verwendung finden, werden als Prozeduren in der Bibliothek angelegt.

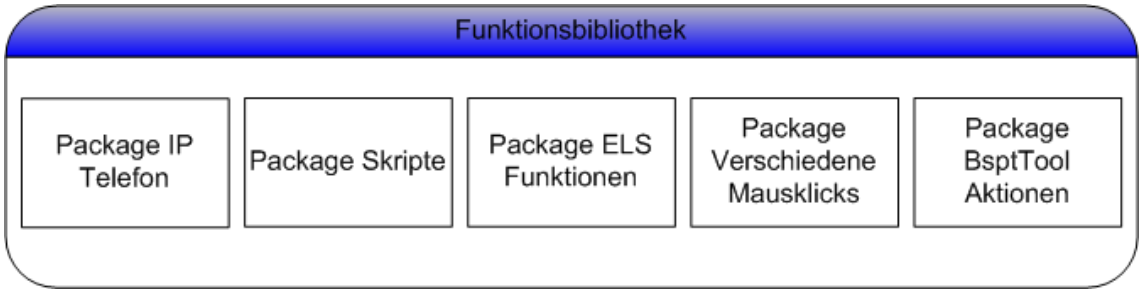

Abbildung 22: Funktionsbibliothek Packages

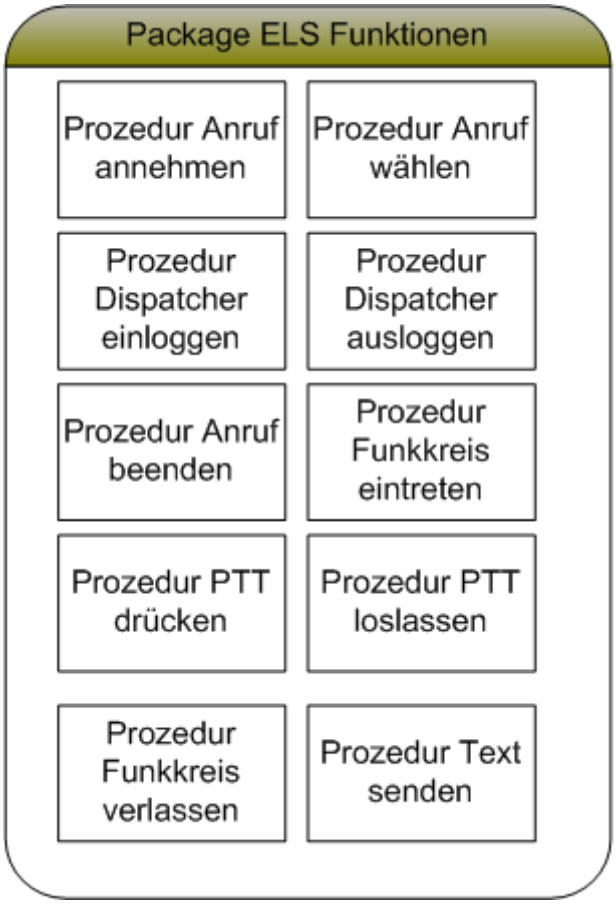

Abbildung 23: Package der Funktionsbibliothek

8.2.1 Aufbau einer Prozedur

Es gibt keine festgelegten Regeln, wie eine Prozedur aufgebaut sein muss. Jede Prozedur benötigt einen Namen. Dieser Name sollte so gewählt werden, dass sofort ersichtlich ist, welche Aufgabe durch die Prozedur übernommen wird. Fall nötig, können zu jeder Prozedur Übergabeparameter angelegt werden. Ihre Aufgabe ist es, die Kommunikation zwischen den Prozeduren und „QF-Test“ zu ermöglichen. Somit können Variablen, die später im Testszenario gebraucht werden, an die Prozedur übergeben werden. Der schematische Aufbau einer Prozedur beschreibt die Vorgehensweise beim Erstellen (siehe Abbildung 24). Die Aufgabe dieser Prozedur ist die Implementierung einer Funktion, der „BOSPORUS TIP“ Schnittstelle.

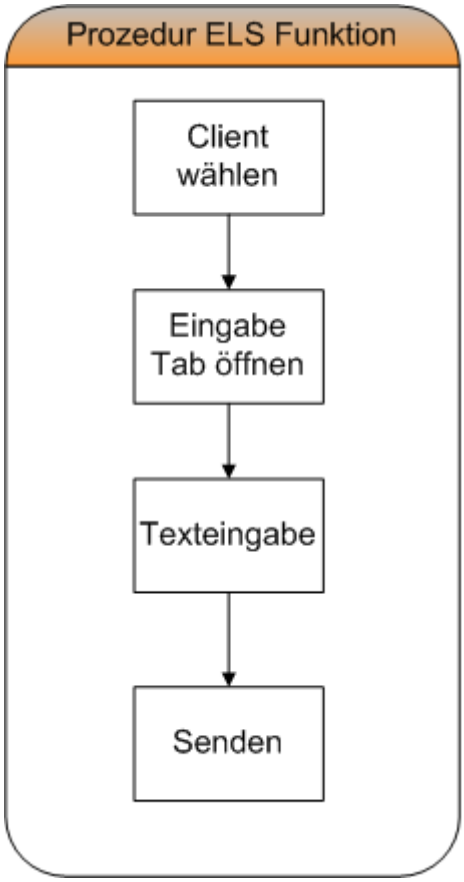

Abbildung 24: Allgemeiner Prozeduraufbau für ELS Funktionen

Die erste Anweisung legt die Anwendung (Client) fest, auf dem die Prozedur ausgeführt werden soll. Jede zu testende Anwendung, die über „QF-Test“ gestartet wird, benötigt einen eindeutigen Namen. Dieser Name identifiziert die Anwendung in „QF-Test“. Die Zuweisung einer Variablen innerhalb einer Prozedur erfolgt mit einem speziellen „Variablen-Knoten“.

Die Anweisungen „Eingabe Tab öffnen“ und „Senden“ sind ebenfalls Prozeduraufrufe. Diese liegen unter dem Package „BsptTool Aktionen“. Daran lässt sich erkennen, dass die Prozeduren untereinander verschachtelt werden können. Deshalb muss beim Erstellen darauf geachtet werden, dass Änderungen in den Prozeduren zu Fehlern führen können, die nicht sofort ersichtlich sind. Der Knoten „Texteingabe“ erledigt die eigentliche Aufgabe und enthält den XML Request. Die Parameter des XML Requests werden über die Parameter der Prozedur nach außen weitergeleitet. Somit kann die Prozedur flexibel eingesetzt werden.

8.3 Testszenarien entwickeln

8.3.1 Testkriterien festlegen

Damit die Qualität der „BOSPORUS TIP“-Schnittstelle festgestellt werden kann, muss definiert sein, wie sich die Schnittstelle bei bestimmten Anfragen verhält. Dazu dienen sogenannte Sequenzdiagramme. Diese Sequenzdiagramme bilden die Kommunikation zwischen „NORUMAT TIP“ und Einsatzleitsystem (ELS) ab (siehe Abbildung 25). Die Sequenzdiagramme werden aus den Anwenderbedienabläufen abgeleitet. Diese liegen teilweise als Use-Case Diagramme vor oder in rein textueller Form. Anhand dieser Informationen kann geprüft werden, ob die Schnittstelle das tut was sie soll.

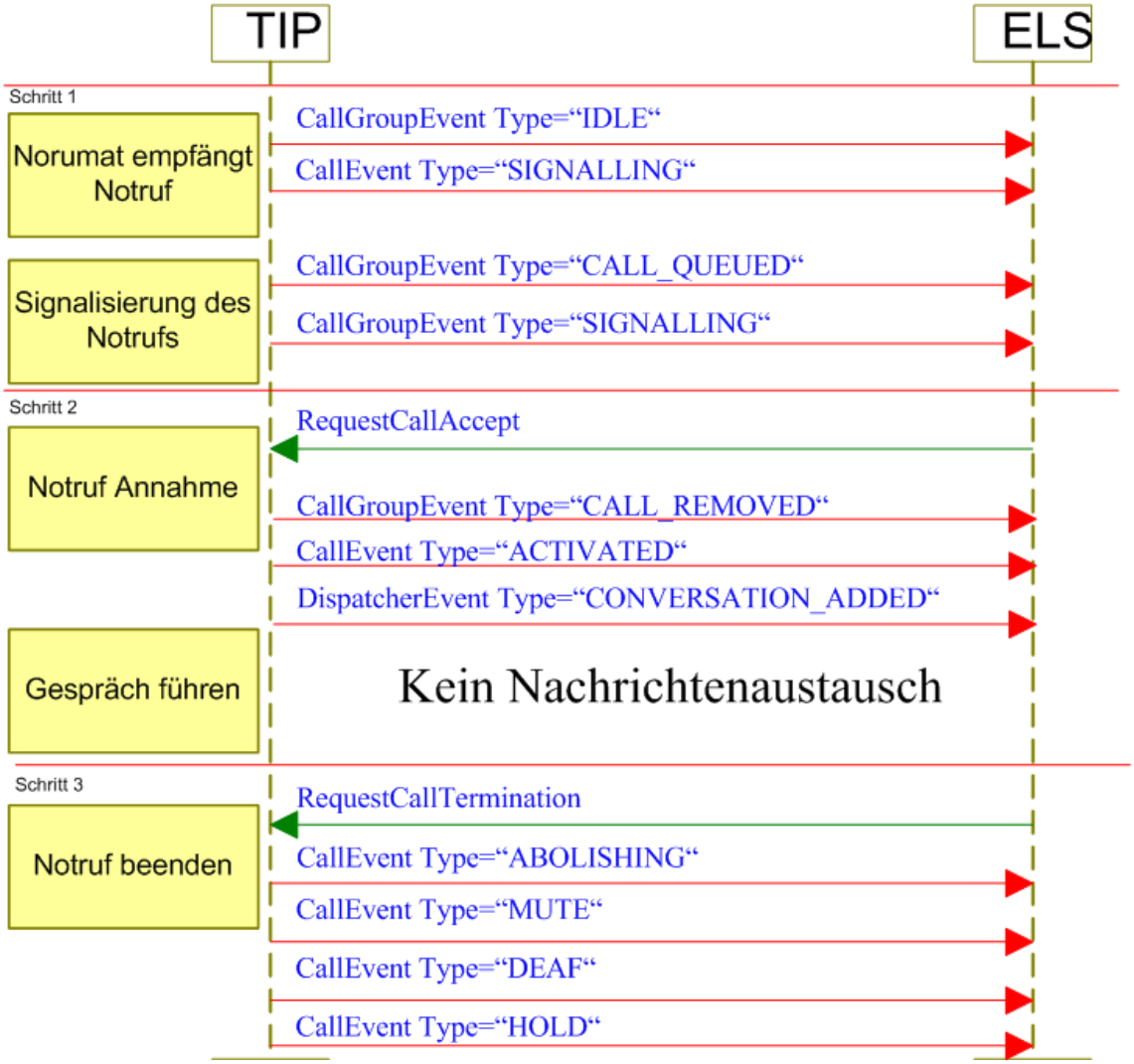

Abbildung 25: Sequenzdiagramm Notruf annehmen (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013)

Die Anzahl der Nachrichten, die gesendet werden, hängen von der Konfiguration des Systems ab. Auf die Konfiguration des Systems wird nicht näher eingegangen.

Das Testkonzept sieht vor, die Typen der Events zu prüfen und gegebenenfalls auch die Attribute. Ob Attribute mit getestet werden hängt von der jeweiligen Sequenz ab. Bei einem Notruf muss beispielsweise neben dem Typ des Events auch das Attribut „IsEmergency=true“ geprüft werden. Jeder Event hat eine Vielzahl von Attributen. Es ist nicht genau spezifiziert, welche Attribute benötigt werden. Falls die Spezifikation nachträglich ergänzt wird, muss die Testauswertung überarbeitet werden. Die Anzahl der empfangenen Events wird nicht berücksichtigt. Sobald ein Event des jeweiligen Typs vorhanden ist, wird der Test mit „OK“ bewertet. Fehlt ein Event läuft der Test weiter und es wird im Protokoll eine Fehlermeldung erzeugt.

Die konkrete Umsetzung dieses Sequenzdiagramms, wird in Abschnitt 9.1 genauer beschrieben.

8.4 Auswertung Testszenario

Die Testszenarien sind so aufgebaut, dass während eines Tests sämtlicher Datenverkehr zwischen der „BOSPORUS TIP“ Schnittstelle und dem „NORUMAT TIP“ Server in eine Datei geschrieben wird. Diese Datei wird mit der Endung .xml angelegt. Allerdings lässt sich diese Datei nicht mit einem XML Parser einlesen, da die inhaltliche Struktur nicht passt. XML Dateien müssen bestimmte Regeln einhalten, damit sie gelesen werden können. Im Abschnitt 8.4.1 wird erklärt, wie der interne Aufbau einer solchen XML Datei aussehen muss und im Abschnitt 8.4.2 wird gezeigt, wie mit Hilfe eines Groovy-Skripts die Datei zur Auswertung aufbereitet wird.

Liegt die XML Datei aufbereitet vor, kann mit Hilfe des Groovy XML Parsers die Datei eingelesen werden. Die XML Datei enthält alle Events die durch das Testszenario gesendet wurden. Die Auswertung einzelner Events geschieht mit einer Kombination aus each-Schleife und closure (siehe Abbildung 26, in diesem Beispiel wurde die XML Datei schon eingelesen und liegt in der Variable „testDatei“ vor). Die Schleife enthält die vordefinierte Iteratorvariable „it“. Diese Variable erlaubt es, auf den Inhalt zu zugreifen. Der Inhalt ist immer abhängig vom jeweiligen Kontext, hier enthält die Variable alle Informationen des CallEvents.

```
testDatei.BosporusTipEvent.CallEvent.each{

    switch(it.'@Type'){

        case "ACTIVATED": rc.logMessage("Test ok")
        case "ABOLISHING":rc.logMessage("Test ok")
        default:          rc.logError("Test not ok")
    }
}
```

Abbildung 26: Groovy Konstrukt überprüft CallEvents mit dem Typ "ACTIVATED" und "ABOLISHING"

In der Iteratorvariable „it“ liegt pro Schleifendurchlauf ein CallEvent. Über die Switch-Anweisung kann dann einzeln auf die Typen der CallEvents zugegriffen werden.

Der Aufwand beim Erstellen der Auswerteskripte wird durch diese Vorgehensweise reduziert. Des Weiteren ist der Code leichter zu lesen und zu warten.

Wenn die Anforderung nur voraussetzt, dass ein Event vorhanden sein muss, kann direkt im Case-Zweig die Ausgabe in das Protokoll erfolgen.

Im Default-Zweig werden fehlende Events als Fehlermeldung erzeugt. Es kann nicht genau erkannt werden, welche Events fehlen. Deshalb werden für die Analyse von

Fehlern zusätzlich zu jedem Testdurchlauf separate Logdateien angelegt. Für jedes Testszenario muss ein eigenes Auswerteskript erstellt werden. Einzelne Inhalte sind so aufgebaut, dass sie in anderen Skripten wiederverwendet werden können.

8.4.1 Aufbau XML Dokument

Der Aufbau einer regelkonformen XML Datei (siehe Abbildung 28) ist wichtig und wird daher im folgenden Abschnitt an einem Beispiel (siehe Abbildung 27) erklärt. Die ersten zwei Zeilen haben eine besondere Bedeutung, deshalb werden diese Zeilen genauer betrachtet. Die erste Zeile gibt an, welche XML Version verwendet wird. Diese Zeile darf im gesamten Dokument nur einmal auftauchen. Die Angabe des „Encoding“ ist optional, empfiehlt sich aber, denn UTF-8 unterstützt viele Sonderzeichen beispielsweise „öäü“. Die zweite Zeile „XML DTD“ ist auch wie das „Encoding“ optional. DTD steht für „Document Type Definition“ und beschreibt die Regeln der XML Datei. Damit lässt sich beispielsweise überprüfen, wie viele Elemente die XML Datei benutzen darf und ob der Name der einzelnen Elemente richtig ist etc. Die DTD wird für die Auswertung der Testszenarien nicht benötigt. Die dritte Zeile beschreibt das Wurzel Element. Dieses Element dient dem XML-Parser als Startknoten. Von dort aus hangelt man sich zu den einzelnen Kind-Knoten (Elementen). Jedes Element (auch das Wurzel Element) beginnt mit <Name und endet mit </Name (Abbildung 27 Zeile 3 und 8). Die Namen der Elemente sind nicht festgelegt und können vom Entwickler frei vergeben werden.

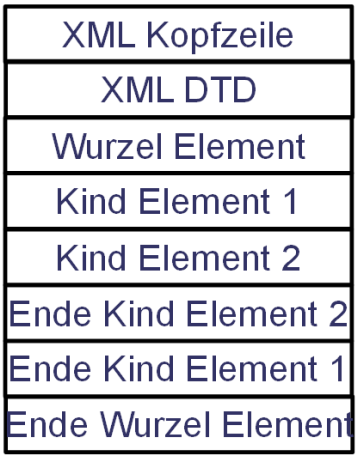

Abbildung 28: Allgemeiner XML Aufbau

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE BosporusTipRequest SYSTEM "bosporus-tip-core.dtd">
3 <BosporusTipRequest Sender="ELSNord" Version="5.0.0">
4     <RadioCircuitRequest Occurrence="01.01.1970 00:00:00 +0000">
5         <RequestRadioCircuitSelection DispatcherId="user55"
6                                       RadioCircuitId="rc144"/>
7     </RadioCircuitRequest>
8 </BosporusTipRequest>
```

Abbildung 27: Konkrete XML Umsetzung

8.4.2 Aufbau eines Groovy Skripts

Nach jedem Testszenario werden zwei Skripte gestartet. Ein Skript wandelt die Logdatei in eine für den XML Parser lesbare Datei. Ein Ausschnitt der Logdatei ist in Abbildung 29 zu sehen. Dabei spielt der Inhalt der einzelnen Nachrichten keine Rolle. Problematisch für den XML Parser sind die Logzeitpunkte (in Abbildung 29 grün dargestellt) und die jeweiligen ersten beide Zeilen der Nachrichten (Nummer 2/3). Wie in Abschnitt 8.4.1 bereits erwähnt, darf das XML Dokument diese Zeilen nur einmal beinhalten.

```
<!-- 09:05:00.191 msg received -->                                    (1)

<?xml version="1.0" encoding="UTF-8"?>                                (2)
<!DOCTYPE BosporusTipEvent SYSTEM "bosporus-tip-core.dtd">            (3)
<BosporusTipEvent Sender="2229" Version="5.1.0">
   <CallGroupEvent Occurrence="26.06.2014 09:04:58 +0200" Type="IDLE">
      <CallGroup CurrentState="IDLE" DisplayName="SIM S2m"
         GroupCapacity="30" Id="6619" RemainingCapacity="30" SignallingCalls="0"/>
   </CallGroupEvent>
</BosporusTipEvent>

<!-- 09:05:00.441 msg sent -->                                        (4)

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BosporusTipRequest SYSTEM "bosporus-tip-core.dtd">
<BosporusTipRequest Sender="ELSNord" Version="5.0.0">
   <DispatcherRequest Occurrence="01.01.1970 00:00:00 +0000">
      <RequestDispatcherLoginProfile DispatcherId="2343" ProfileId="2352"/>
   </DispatcherRequest>
</BosporusTipRequest>
```

Abbildung 29: Log Datei Beispiel

Das Skript übernimmt die Aufgabe, diese drei Zeilen zu entfernen. Die Nummer (1) und Nummer (4) enthält die Uhrzeit mit Millisekunden Angaben sowie Text. Die Uhrzeit sowie der Text (entweder „msg received“ oder „msg sent“) kann unterschiedlich sein. Damit diese Zeilen zuverlässig erkannt werden, müssen reguläre Ausdrücke benutzt werden.

Reguläre Ausdrücke

In Groovy wird ein regulärer Ausdruck durch zwei Slashes (/Ausdruck/) gekennzeichnet.

Um die Zeile (Abbildung 29 Nummer (1)) zu entfernen, könnte man auf die Idee kommen, einfach den exakten Wert als Ausdruck zu formulieren. (siehe Abbildung 30).

```
ausdruck = /<!-- 09:05:00.191 msg received --> /
```

Abbildung 30: Groovy Codezeile statischer Ausdruck

Dies würde für die Zeile (Abbildung 29 Nummer (1)) tatsächlich funktionieren. Wenn sich allerdings die Zeit um nur eine Millisekunde ändert, funktioniert es schon nicht mehr. Damit nicht für jede Millisekunde ein neuer Ausdruck erstellt werden muss, gibt es Operatoren, die diese Aufgabe übernehmen.

Dazu bietet Java die Klasse „Pattern“ an, die die Operatoren bereitstellt. Der komplette Ausdruck ist zu kompliziert, daher beschränken wir uns auf einen Teilausschnitt. Damit die Zeit immer zuverlässig erkannt wird, benutzen wir Operatoren anstatt der exakten Werte. Der Operator „\d“ beispielsweise entspricht einer Zahl zwischen 0-9. Der Operator „\d+“ sagt aus, dass bis zum nächsten Operator, beliebig viele Zahlen mit den Werten 0-9 enthalten sein können. Die Millisekunden können bis zu drei Stellen lang sein. Der Operator „\p{Punct}“ erkennt ein Sonderzeichen. In diesem Ausdruck (Abbildung 31) wird er verwendet um die Doppelpunkte zwischen den Stunden und Minuten zu erkennen, sowie den normalen Punkt zwischen den Sekunden und den Millisekunden.

```
zeit_erkennen = /\d\d\p{Punct}\d\d\p{Punct}\d\d\p{Punct}\d+/
```

Abbildung 31: Groovy Codezeile regulärer Ausdruck

Das zweite Skript liest die erstellte XML Datei ein und vergleicht die darin enthaltenen Angaben mit vordefinierten Werten. Stimmen die enthaltenen Werte mit den vorgegebenen überein, wird eine Notiz in das Protokoll geschrieben. Diese soll dem Tester bestätigen, dass der Testfall in Ordnung war. Enthält die XML Datei Werte, die nicht mit den vorgegebenen übereinstimmen, wird eine Fehlermeldung erzeugt und in das Protokoll geschrieben. Der Test läuft unbeirrt weiter. Am Ende des Testlaufs signalisiert eine Meldung, wie viele Fehler, Warnungen oder auch Exceptions in dem Testlauf aufgetreten sind.

8.4.3 Darstellung der Testergebnisse im Protokoll

Für die Darstellung der Testergebnisse wird das Protokoll von „QF-Test“ verwendet. Dazu wird im Auswerteskript über das rc-Objekt Nachrichten in das Protokoll geschrieben. Die Nachrichten müssen anzeigen, um welche Events es sich handelt und ob der Test in Ordnung war oder nicht. In den folgenden Abbildungen sind zwei Protokollausgaben zu sehen. Im oberen Bild (Abbildung 32) sind die Protokollausgaben ohne Fehler zu sehen, im unteren Bild (Abbildung 33) mit Fehler.

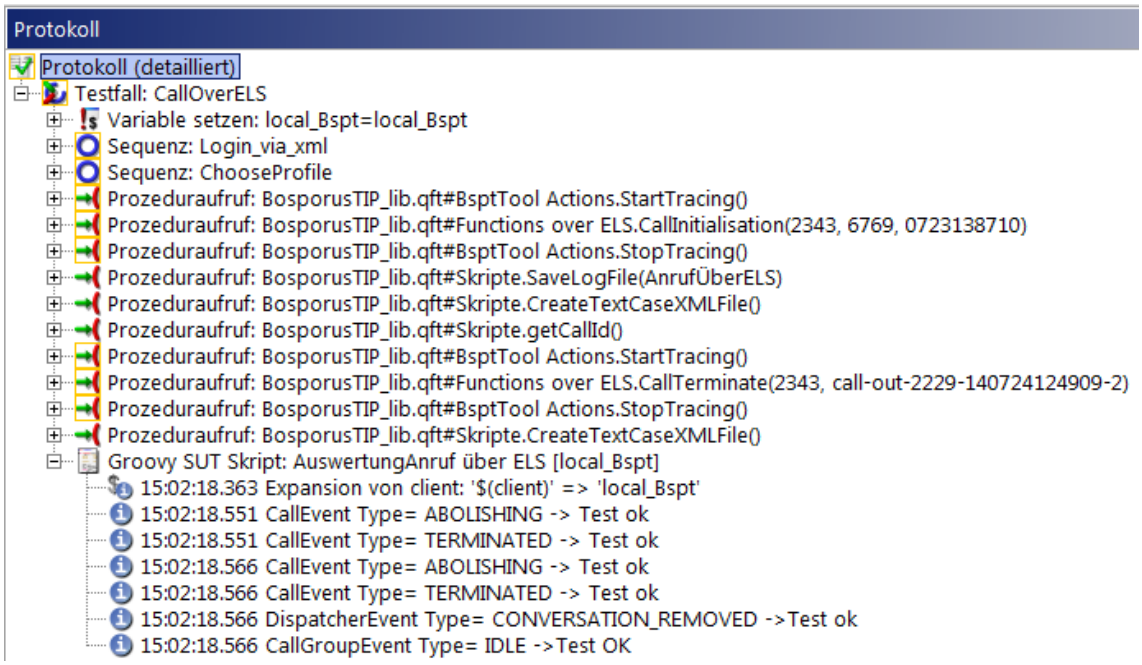

Abbildung 32: „QF-Test“ Protokoll ohne Fehler

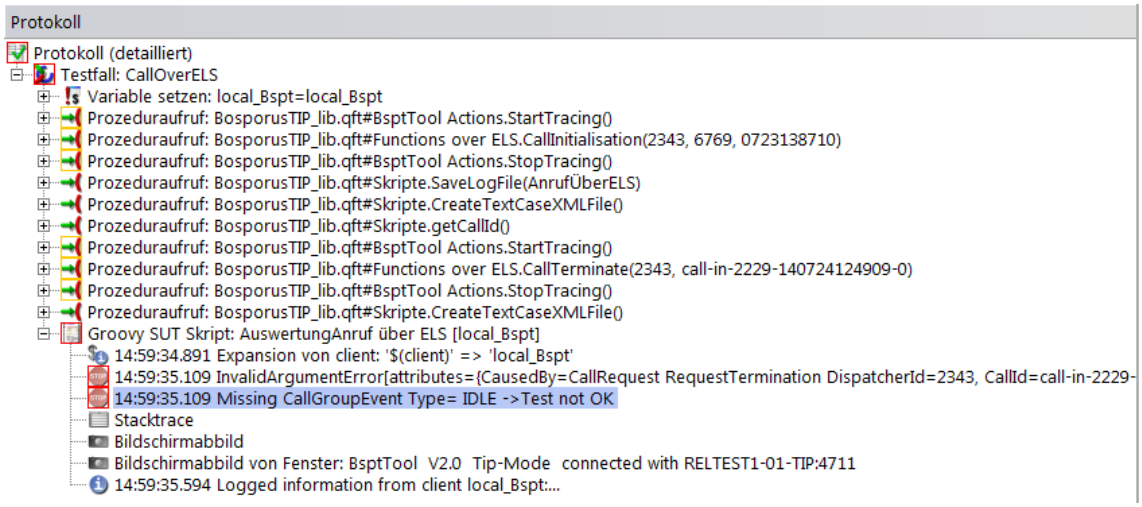

Abbildung 33: „QF-Test“ Protokoll mit Fehler

9 Vorgehensweise für die Entwicklung von Testszenarien mit QF-Test

Der Entwicklungsablauf lässt sich wie folgt beschreiben:

Zuerst wird das Sequenzdiagramm analysiert. Anschließend wird geprüft, ob alle benötigten Requests als Prozeduren in „QF-Test“ vorhanden sind. Falls nicht, müssen diese zuerst implementiert werden.

Im nächsten Schritt wird in „QF-Test“ ein neuer Testfall angelegt. Die allererste Prozedur startet das Tracing, damit alle nachfolgenden Ereignisse geloggt werden. Danach wird das Sequenzdiagramm eins zu eins in „QF-Test“ abgebildet. Am Ende der Sequenz wird das Tracing gestoppt. Durch das Stoppen des Tracings wird die Logdatei erstellt.

9.1 Konkrete Entwicklungsbeschreibung an einem Testszenario Beispiel Notruf

Bevor der eigentliche Test beginnen kann, müssen bestimmte Vorbedingungen erfüllt werden. Zuallererst muss das BsptTool über „QF-Test“ gestartet werden. Damit erhält „QF-Test“ die komplette Kontrolle über das Programm. Dazu ruft man die Prozedur „StartBosporusAndLogin“ auf. Anschließend wird das Softwaretelefon „Linphone“ registriert. Dieses Werkzeug wird benötigt, um über „QF-Test“ Notrufe simulieren zu können. Neben den Schnittstellentests ist es auch wichtig, die korrekte Darstellung auf dem „NORUMAT TIP“-GUI zu überprüfen. Sämtliche Aktionen, die über die „BOSPORUS TIP“ Schnittstelle gemacht werden, lösen Aktionen auf dieser GUI aus und müssen kontrolliert werden. Deshalb ist es hilfreich, das „NORUMAT TIP“-GUI über „QF-Test“ ebenfalls zu starten.

Als nächstes wird das BsptTool konfiguriert. Dazu sind folgende Schritte notwendig.

1. Verbindung zum „NORUMAT TIP“-Server herstellen
2. Anmelden mit Username und Passwort
3. Pfad für die Logdatei eingeben
4. Fenster für das Senden von XML Requests öffnen

Zuerst muss eine Socket Verbindung mit dem „NORUMAT TIP“-Server hergestellt werden. Die IP-Adresse und der Port sind am Testsystem standardmäßig vergeben und müssen nur einmal eingegeben werden. Die Anmeldung erfolgt anschließend mit Eingabe des Usernamen und des Passwortes. Für jede Aktion wurde in „QF-Test“ eine Prozedur angelegt.

Aus dem Sequenzdiagramm (siehe Abbildung 25) lässt sich erkennen, dass genau drei unterschiedliche Arten von XML Events gesendet werden. Es gibt „CallEvents“, „CallGroupEvent“ und „DispatcherEvent“.

Ein CallEvent ist ein Ereignis, das direkt mit einem Anruf in Verbindung steht. Es zeigt an, dass beispielsweise ein neuer Notruf ansteht oder dass ein Dispatcher aufgelegt hat. Es gibt insgesamt achtzehn unterschiedliche Typen dieses Events.

Das Attribut „Type“ zeigt in jedem Event-Tag den Auslöser des Events an (siehe Abbildung 25). Entspricht das Attribut „Type“ = SIGNALLING bedeutet dies, dass der Ruf signalisiert wird. Dieser Event mit dem „Type“ SIGNALLING wird bei eingehenden sowie bei ausgehenden Rufen gesendet.

Ein „CallGroupEvent“ ist ein Ereignis, das mit einer Anrufgruppe in Zusammenhang steht.

Die häufigsten Ereignisse bei Anrufgruppen sind ankommende Anrufe. Für diese Art von Event gibt es neun unterschiedliche Typen. Eine „CallGroup“ ist eine logische Gruppe von Leitungen und Rufnummern, über die Anrufe hereinkommen können oder abgehende Anrufe getätigt werden dürfen.

Ein Dispatcher Event wird gesendet, sobald irgendeine Aktion mit einem Benutzer zu tun hat, zum Beispiel bei An- und Abmelden am System oder geführte Gespräche. Für diese Art von Event gibt es elf unterschiedliche Typen.

Diese drei Events haben zusammen noch über dreißig Attribute, die gegebenenfalls geprüft werden müssen.

Damit diese drei Arten vom BsptTool empfangen werden können, müssen sie abonniert werden. Dazu ruft man die Prozedur „SubscribeOneEvent“ dreimal hintereinander auf. Diese Prozedur erwartet einen Parameter, der dem Namen des Events entspricht, welches abonniert werden soll.

Die Erstellung des Testszenarios leitet sich aus dem Sequenzdiagramm (siehe Abbildung 25) ab. Die benötigten XML Requests stammen aus der IDD (Interface Design Description) der „BOSPORUS TIP“ Schnittstelle. Damit die XML Requests verwendet werden können, müssen diese in „QF-Test“ über Prozeduren abgebildet werden. Die einzelnen Prozeduren enthalten neben dem XML Request noch zwei weitere Aufrufe. Der Erste wechselt im BsptTool ins benötigte Fenster, der zweite führt einen Mausklick aus. Die Reihenfolge ist wie folgt definiert. Zuerst wird das entsprechende Fenster im BsptTool geöffnet, danach wird der XML Request, in Form von Text eingefügt. Über einen Mausklick auf den Send-Button wird der XML Request abgeschickt.

Durch die Verwendung von Prozeduren wird dem Tester die Erstellung neuer Testszenarien erleichtert. Fast jeder XML Request benötigt Parameter. Diese werden beispielsweise benötigt, um einen Disponenten anzumelden oder auch einen bestimmten Notruf/ Anruf annehmen zu können. Jede Prozedur ist so aufgebaut, dass jeder einzelne Parameter, der für den XML Request benötigt wird, an die Prozedur übergeben werden muss. Es besteht allerdings die Möglichkeit, Standard Parameter zu erstellen. Diese werden dann benutzt, falls der Tester keine Parameter angegeben hat.

Das Sequenzdiagramm wird anschließend in „QF-Test“ implementiert. Damit der „NORUMAT TIP“ einen Notruf empfangen kann (siehe Abbildung 25 Schritt 1), muss dieser generiert werden. Dies geschieht über das Softwaretelefon „Linphone“. Die dazu benötigten Prozeduren wurden in „QF-Test“ implementiert. Zuerst muss über die Prozedur „LinphoneInitRegister“ ein Linphone an dem VoIP-Gateway registriert werden. Diese Prozedur muss zu Beginn einmal gestartet werden. Anschließend kann man über „QF-Test“ mit diesem Softwaretelefon Telefonanrufe oder Notrufe generieren. Um einen Anruf auszulösen, wird die Prozedur „GenerateCall“ aufgerufen. Die Prozedur erwartet als Eingabeparameter die zu wählende Telefonnummer. Über die Prozedur „StopCall“ kann der Anruf beendet werden.

Nachdem der Notruf generiert wurde, sendet der „NORUMAT TIP“ die vier XML Events (siehe Abbildung 25 Schritt 1). Jeder Notruf erhält vom System eine eindeutige Nummer (Call-ID). Das XML Event „CallEvent“ enthält diese Call-ID. Über ein Groovy Skript kann man die Call-ID aus diesem „CallEvent“ auslesen. Die Call-ID wird (zurzeit) als globale Variable in „QF-Test“ angelegt. Über das Runcontext Objekt (rc) von „QF-Test“ kann auf diese globale Variable zugegriffen werden. Mit der Prozedur „CallAccept“ kann der Notruf angenommen werden (siehe Abbildung 25 Schritt 2). Die Prozedur benötigt zwei Parameter: Zum einen die Call-ID und zum anderen die Dispatcher-ID. Die Dispatcher-ID benötigt das System, um den Notruf einem Disponenten zuordnen zu können. Die Dispatcher-ID entsteht beim Anlegen eines Disponenten im System und ist vor jedem Test bekannt. Das Gespräch wird nach 5 Sekunden durch den Prozeduraufruf „CallTerminate“ beendet. Die Prozedur „CallTerminate“ benötigt dieselben Parameter wie die Prozedur „CallAccept“. Am Ende wird die Logdatei über ein Groovy Skript umgewandelt und danach über ein weiteres Groovy Skript ausgewertet. Dabei werden neben den Typen der XML Events auch die Attribute geprüft. Das XML Event CallEvent Type=“ACTIVATED“ enthält das Attribut „isEmergency“. War die gewählte Nummer ein Notruf, so ist dieses Attribut auf „true“ gesetzt. Jeder Event enthält neben dem Attribut „Type“, weitere Attribute die gegebenenfalls geprüft werden müssen.

Tests, die erfolgreich waren, werden im Protokoll von „QF-Test“ durch eine Nachricht kommentiert. Ist dagegen ein Fehler aufgetreten, wird im Protokoll eine Fehlermeldung erzeugt.

9.2 Realisierte Sequenzdiagramme

Im Rahmen der Bachelorarbeit sind folgende Sequenzdiagramme umgesetzt worden (Abbildung 34, Abbildung 35, Abbildung 36 , Abbildung 37). Der Aufwand der einzelnen Sequenzdiagramme kann stark variieren. So ist das Testszenario „Schichtwechsel“ (siehe Abbildung 35 und Abbildung 36) deutlich umfangreicher als das Testszenario „Funkkreis besprechen“ (siehe Abbildung 34).

Die roten Pfeile zeigen die Nachrichten an, die im Auswerteteil des Testszenarios berücksichtigt werden müssen. Die grüne Pfeile sind als Prozedur in QF-Test angelegt und können verwendet werden. Die Anfragen in den Sequenzdiagrammen enthalten nicht die benötigten Parameter, diese müssen zusätzlich aus der Spezifikation entnommen werden.

1. Testszenario Funkkreis eintreten, PTT betätigen, austreten

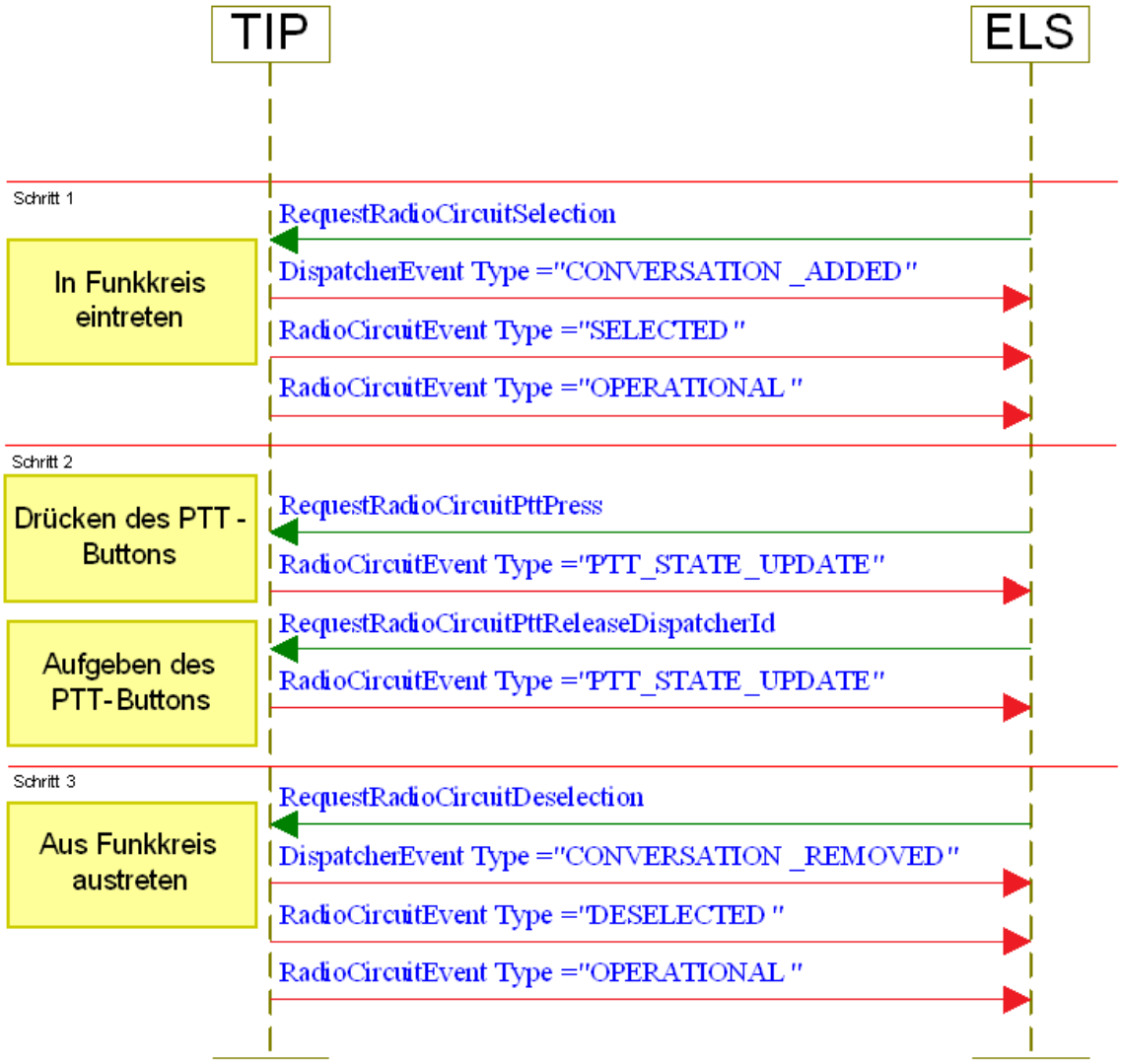

Abbildung 34: Sequenzdiagramm analoger Funkkreis besprechen (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013)

2. Testszenario „Schichtwechsel“ zwei Disponenten anmelden und abmelden

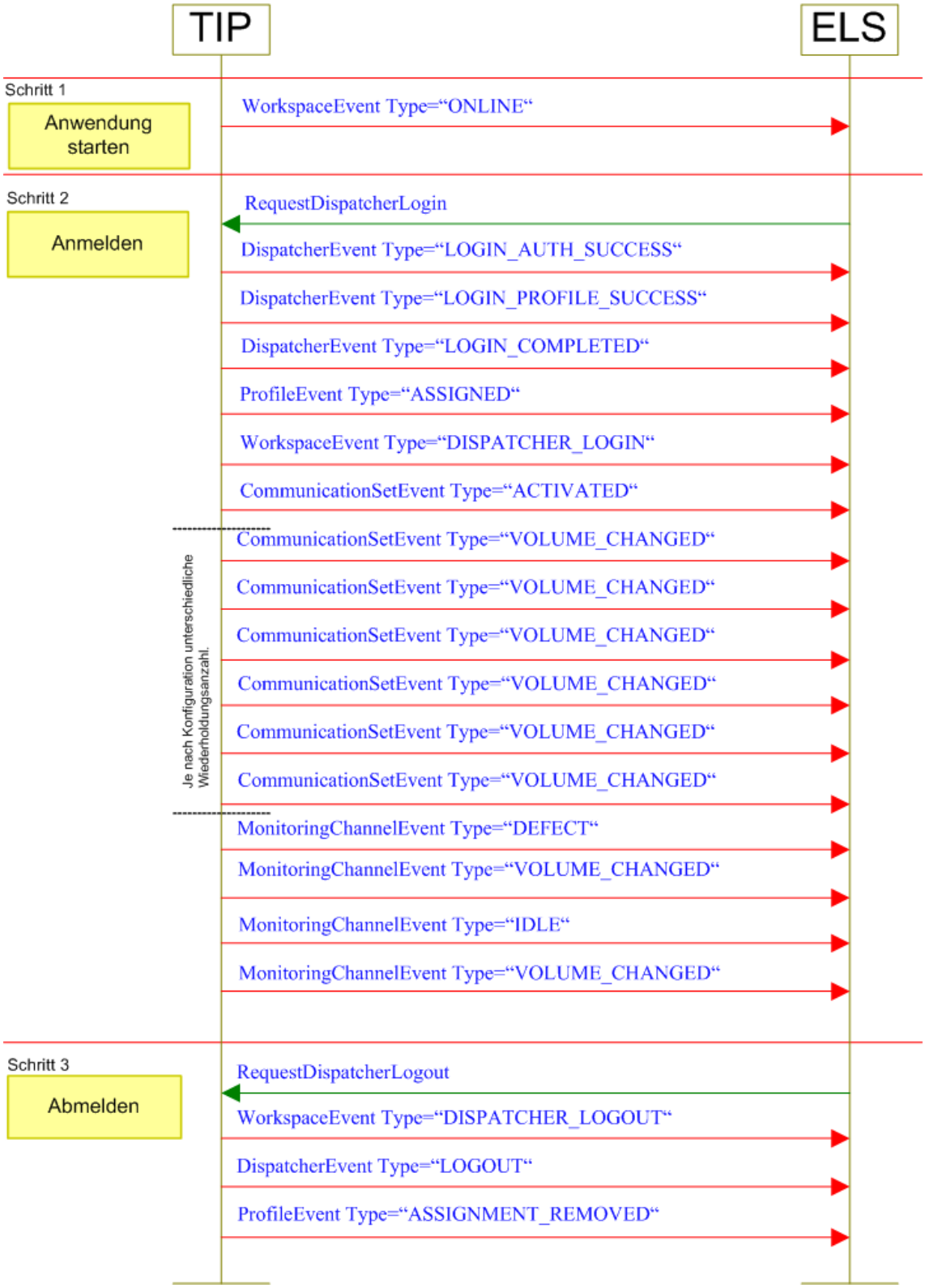

Abbildung 35: Sequenzdiagramm Schichtwechsel (1) (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013)

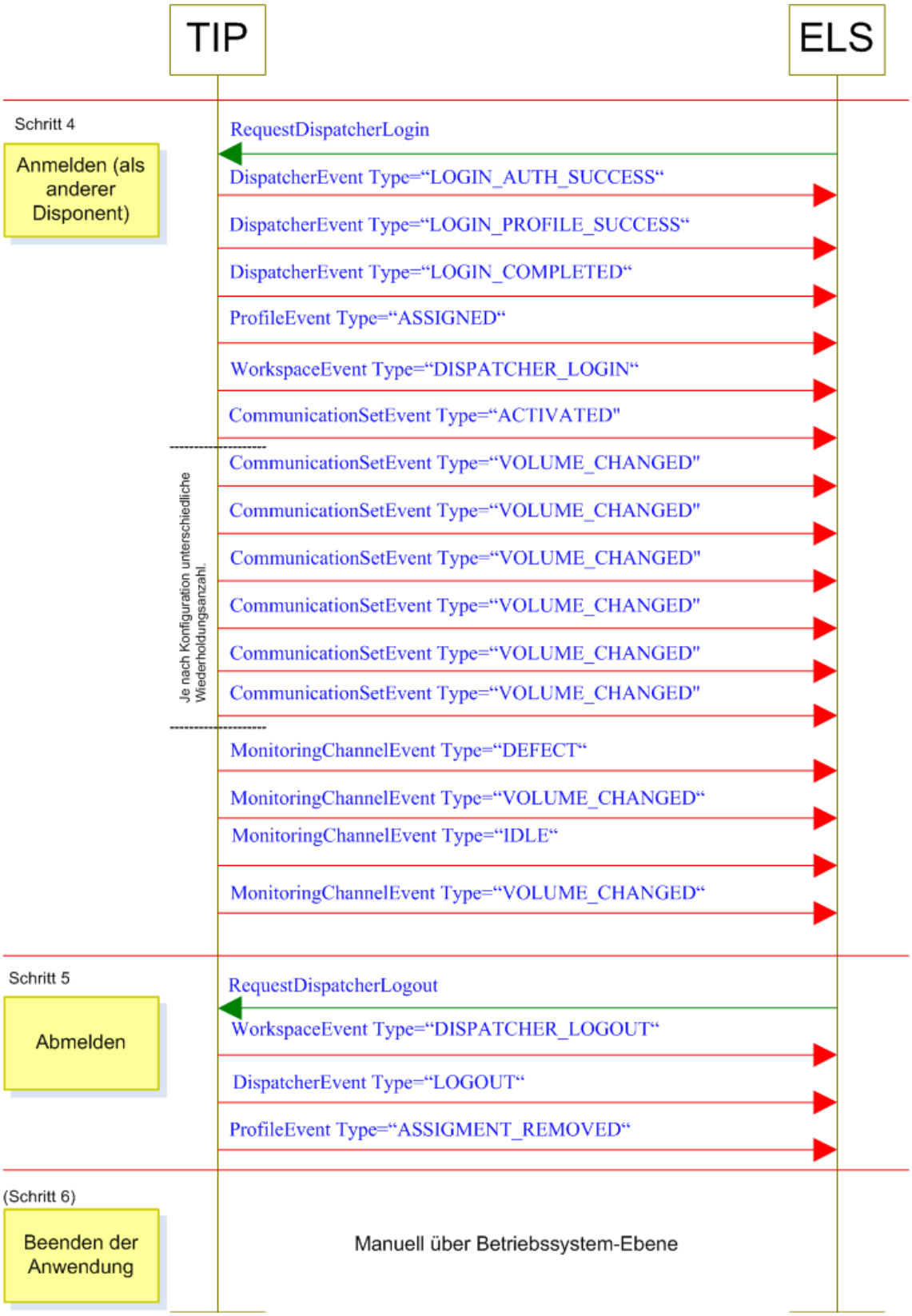

Abbildung 36: Sequenzdiagramm Schichtwechsel (2) (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013)

3. Testszenario Notruf annehmen Notruf generieren, annehmen, beenden

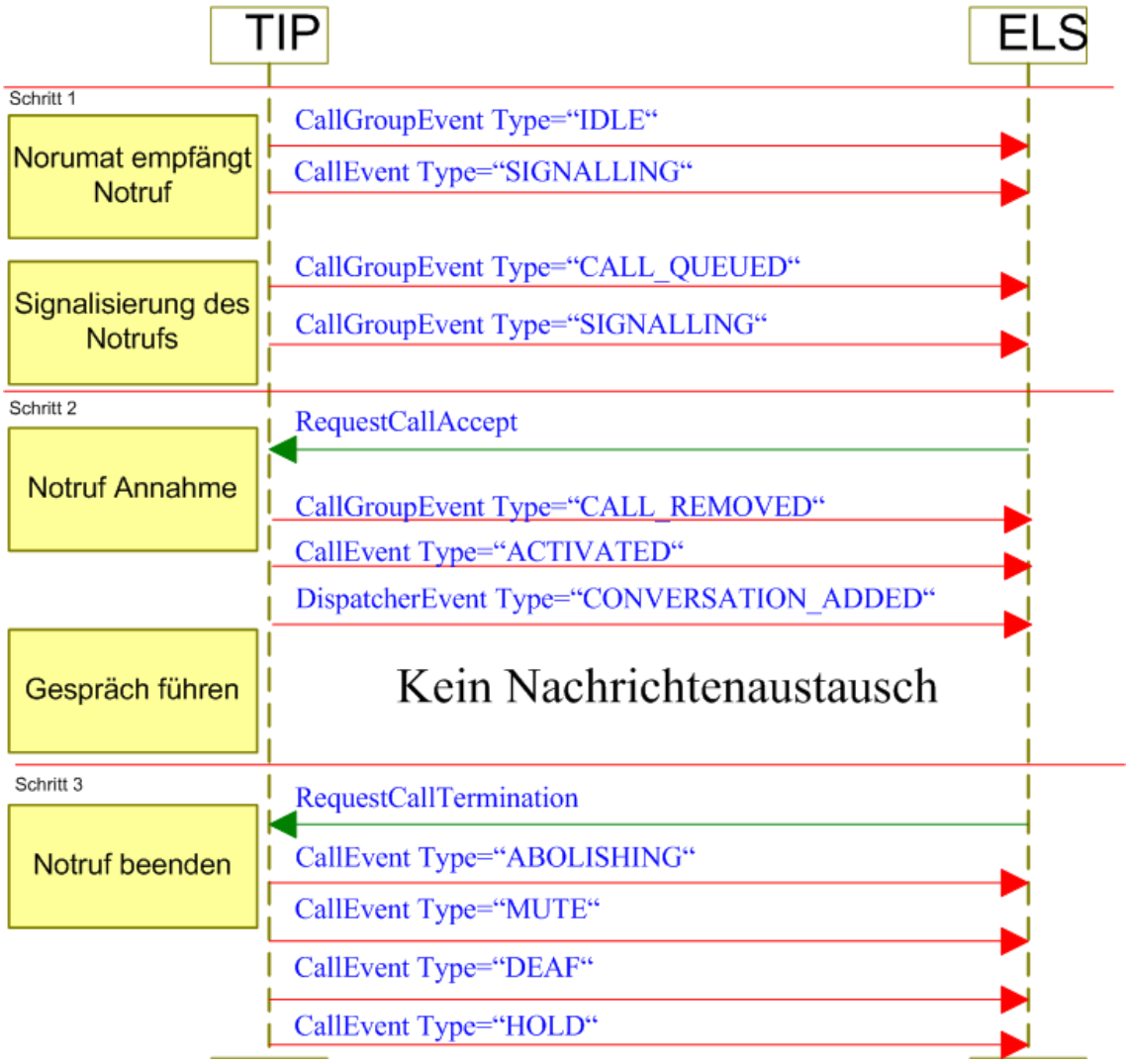

Abbildung 37: Sequenzdiagramm Notruf annehmen (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013)

10 Mögliche Testerweiterungen

Das Programm „QF-Test“ stellt viele Testerweiterungen zur Verfügung. Es bietet durch die bereits erwähnten Ablaufsteuerungen die Möglichkeit, zu den funktionalen Tests auch Dauertests durchzuführen. Damit lässt sich das Gesamtsystem über die Zeit belasten. Jeder funktionale Test kann damit automatisch auch als möglicher Dauertest verwendet werden. Dazu wird der Test in eine Schleife eingebettet. Die Anzahl der Wiederholungen kann beliebig eingegeben werden.

Des Weiteren unterstützt QF-Test folgende Testausführungsarten:

- Batch Modus
- Daemon Modus

10.1 Batch Modus

Im Batch Modus wird der Testlauf von der Kommandozeile aus gestartet. Dazu wird folgende Kommandozeile ausgeführt:

```
qftest –batch –run c:\mysuites\suiteA.qft
```

Nach dem Start werden alle Knoten der Testsuite nacheinander ausgeführt. Nach Beendigung des Testlaufs findet man im aktuellen Verzeichnis, unter dem gleichen Namen wie die Testsuite, eine Protokolldatei, der man das Testergebnis entnehmen kann. (QFS, 2014)

Es gibt zusätzlich viele Parameter die mehrere Einstellungen erlauben, diese können dem Handbuch von QF-Test entnommen werden (siehe Anhang A).

10.2 Daemon Modus

Der Daemon Modus ermöglicht die Ausführung von Testprozeduren auf anderen im Netzwerk verfügbaren Rechner. Um den Daemon Modus zu realisieren, genügt es auf dem Zielrechner eine „QF-Test“ Instanz im Batch Modus zu starten. Dazu wird beispielsweise die nachfolgende Zeile auf dem Zielrechner ausgeführt.

```
qftest –batch –daemon –daemonport 6666
```

Innerhalb des Testszenarios wird dann die zu testende Software über eine Prozedur auf dem Zielrechner gestartet. Nach dem Start des Testszenarios auf dem lokalen Rechner, führt „QF-Test“ alle Aktionen des Testszenarios auf dem Zielrechner aus.

11 Zusammenfassung und Ausblick

11.1 Zusammenfassung

Ein Ziel der vorliegenden Bachelorarbeit war es, ein Konzept zur Testautomatisierung der „BOSPORUS TIP“-Schnittstelle zu entwickeln. Das Konzept sollte einerseits die Testautomatisierung lösen, andererseits sollte die Erstellung neuer Testszenarien vereinfacht werden. Des Weiteren berücksichtigt das Konzept die automatisierte Auswertung der Testszenarien sowie die Erstellung der Testprotokolle.

Ein weiteres Ziel der Bachelorarbeit war die Realisierung des entwickelten Konzeptes. Die Umsetzung beinhaltet folgende Programme: QF-Test, BsptTool, Groovy Skripte, Linphone. Die Testszenarien werden in dem Programm „QF-Test“ erstellt. Nachdem Start eines Testszenarios werden die genannten Programme sinnvoll verwendet. Während die Testszenarien laufen, müssen keine Eingaben vom Tester gemacht werden. Am Ende eines Testszenarios wird ein Protokoll erzeugt. Das Protokoll enthält die Fehler, Warnungen oder Ereignismeldungen und erlaubt dem Tester die Überprüfung des Testszenarios.

Was die Überprüfung der Funktionalität der Schnittstelle betrifft, so konnte anhand der entwickelten Testszenarien gezeigt werden, dass Fehler in den einzelnen Nachrichten sicher festgestellt wurden.

Die umgesetzte Testautomatisierung deckt nur einen kleinen Teil der „BOSPORUS TIP“-Schnittstelle ab. Ein Grund dafür ist, dass die Spezifikation der „BOSPORUS TIP“-Schnittstelle nur wenige Sequenzdiagramme zur Verfügung stellt, anhand derer die Auswertung der Testszenarien entwickelt werden. Bevor die Testautomatisierung erweitert werden kann, müssen diese Sequenzdiagramme erstellt werden.

Abschließend kann festgestellt werden, dass die erfolgreiche Konzeptionierung und Implementierung der Testautomatisierung die Qualität der Software erhöht und die Dauer der Testdurchführung erheblich verkürzt. Des Weiteren steht über die „BOSPORUS TIP“-Schnittstelle eine weitere Möglichkeit zur Verfügung, das „NORUMAT TIP“-System zu prüfen und eventuelle neue Fehler aufzudecken.

Somit kann die „BOSPORUS TIP“ Schnittstelle zum Erfolg der nächsten Rettungseinsätze beitragen und dadurch Leben retten.

11.2 Ausblick

Das entwickelte Konzept bietet eine gute Basis, die „BOSPORUS TIP“-Schnittstelle automatisiert zu testen. Nun ist es die Aufgabe, diese Testautomatisierung zu erweitern, um die größtmögliche Testabdeckung zu erreichen. Um die Testautomatisierung zu verbessern, können folgende Punkte aufgegriffen werden:

- Dynamisches Einlesen der Paramater aus der Konfigurationsdatenbank
- Die verwendeten globalen Variablen, in lokalen Variablen verwalten
- Ein übergeordnetes Startskript anlegen, in dem Vorbedingungen erledigt werden
- Neue Testszenarien anlegen: ELS → NORUMAT TIP (drei schon implementiert)
- Neue Testszenarien anlegen und auswerten: NORUMAT TIP → ELS (Testszenarien teilweise vorhanden, die Auswertung der Nachrichten fehlt)

12 Literaturverzeichnis

König, D., Glover, A., King, P., Laforge, G., & Skeet, J. (2007). *Groovy im Einsatz.* (D. Heymann-Reder, Übers.) München Wien: Carl Hanser Verlag.

Myers, G., Sandler, C., & Badgett, T. (2012). *The art of software testing.* New Jersey: John Wiley & Sons, Inc., Hoboken.

Pätzold, M., & Seyfert, S. (26. Januar 2010). *Stufen des V-Modells.* Abgerufen am 21. Juli 2014 von Wikipedia: http://de.wikipedia.org/wiki/V-Modell

QFS. (12. Mai 2014). *12.05.2014.* Abgerufen am 5. Juni 2014 von http://www.qfs.de/de/qftest/manual.html

Richard Seidl, M. B. (2012). *Basiswissen Testautomatisierung.* Heidelberg: dpunkt.verlag GmbH.

Software Quality Lab GmbH. (25. März 2009). Abgerufen am 21. Juli 2014 von http://www.software-quality-lab.com/uploads/media/SWQL-Newsletter-200903_-_Testautomatisierung_-_Kosten_und__Nutzen_01.pdf

Spillner, A., & Linz, T. (2010). *Basiswissen Softwaretest.* Heidelberg: dpunkt.verlag GmbH.

Wikipedia. (12. Juli 2013). Abgerufen am 3. Juni 2014 von http://de.wikipedia.org/wiki/ALARP

12.1 Firmeninterne Quelle

THALES DEFENCE & SECURITY SYSTEMS GmbH. (8. Oktober 2013). Interface Design Description für BOSPORUS TIP CORE im NORUMAT TIP. Pforzheim, Baden-Württemberg, Deutschland.

Abbildungsverzeichnis

Abbildung 1: V-Modell nach Boehm 1979 (Pätzold & Seyfert, 2010) 3
Abbildung 2: „NORUMAT TIP“ System Übersicht (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013) 9
Abbildung 3: Allgemeine Kommunikation (THALES DEFENCE & SECURITY SYSTEMS GmbH, 2013) 11
Abbildung 4: Wurzelknoten einer „QF-Test“suite 16
Abbildung 5: Symbolbild Prozeduren 16
Abbildung 6: Symbolbild Extrasequenz 16
Abbildung 7: Symbolbild Fenster und Komponenten 17
Abbildung 8: Symbolbild Sequenzknoten 17
Abbildung 9: Symbolbild If-Konstrukte (If, Elseif, Else) 17
Abbildung 10: Symbolbild Try, Catch, Finally 17
Abbildung 11: Symbolbild Schleifen (While, Normal) und Break 17
Abbildung 12: Symbol Skript in „QF-Test“ 18
Abbildung 13: Symbolbild Programmknoten 18
Abbildung 14: „QF-Test“ Protokoll Übersicht 19
Abbildung 15: BsptTool Anzeige Bereich (1), Manuelle Eingabe Bereich (2) 20
Abbildung 16: Codezeile zum parsen einer Datei 21
Abbildung 17: Beispiel Closure deklarieren (König, Glover, King, Laforge, & Skeet, 2007) 22
Abbildung 18: Screenshot Groovy Console 23
Abbildung 19: Konzeptdesign 24
Abbildung 20: Konzeptdesign interner Aufbau 25
Abbildung 21: Risikograph nach ALARP: Die Risikoanalyse wurde für die drei Funktionen: Notruf annehmen, Funkkreis belegen und Profil anmelden durchgeführt 26
Abbildung 22: Funktionsbibliothek Packages 27
Abbildung 23: Package der Funktionsbibliothek 27
Abbildung 24: Allgemeiner Prozeduraufbau für ELS Funktionen 28
Abbildung 25: Sequenzdiagramm Notruf annehmen 30
Abbildung 26: Groovy Konstrukt überprüft CallEvents mit dem Typ "ACTIVATED" und "ABOLISHING" 31
Abbildung 27: Konkrete XML Umsetzung 32
Abbildung 28: Allgemeiner XML Aufbau 32
Abbildung 29: Log Datei Beispiel 33
Abbildung 30: Groovy Codezeile statischer Ausdruck 34
Abbildung 31: Groovy Codezeile regulärer Ausdruck 34
Abbildung 32: „QF-Test“ Protokoll ohne Fehler 35
Abbildung 33: „QF-Test“ Protokoll mit Fehler 35
Abbildung 34: Sequenzdiagramm analoger Funkkreis besprechen 39

Abbildung 35: Sequenzdiagramm Schichtwechsel (1) 40
Abbildung 36: Sequenzdiagramm Schichtwechsel (2) 41
Abbildung 37: Sequenzdiagramm Notruf 42

Glossar

Tracing

Im BsptTool wird über die Tracing Funktion sämtlicher Nachrichtenverkehr in eine Datei geschrieben.

BsptTool

Bezeichnet das „BOSPORUS TIP“ Werkzeug und ist Name der Anwendung.

Runcontext Objekt

Das Runcontext Objekt (rc) ermöglicht den Zugriff von „QF-Test“ zu den Komponenten der zu testenden Anwendung. Des Weiteren können Informationen in das Protokoll geschrieben werden. Das Variablenmanagement wird ebenfalls hierüber gesteuert.

Dispatcher/ Disponent

Bezeichnung für den Mitarbeiter der das System bedient. Dispatcher wird im Entwicklungskontext und Disponent im Projektkontext verwendet.

Prozedur/ Funktionen

Im „QF-Test“ Kontext ist eine Funktion, eine Prozedur. Einfach ein anderer Name für Funktion.

Testdurchlauf

Ein Testdurchlauf bezeichnet die Ausführung von Tests.

Testszenario

Beschreibt die Umsetzung eines Sequenzdiagramms in „QF-Test“.

BOSPORUS TIP

Name der Schnittstelle zwischen „NORUMAT TIP“ und Einsatzleitsystem.

QF-Test

Testautomatisierungswerkzeug der Firma QFS.

Scrum

Scrum ist ein Modell mit dem Software agil entwickelt werden kann.

Anhang A

Auf der CD enthalten: Erstellte Groovy Skripte

PDF von Internetquellen: ALARP, V-Modell, Testautomatisierung Software Quality Lab, Handbuch QF-Test