Bericht zu notwendigen Veränderungen und Ergänzungen bei den Issue-Trackern etc. für Entwickler (R 4.2.1)

Version 1.0 vom 15.11.2011
Arbeitspaket 4
verantwortlicher Partner TU Darmstadt / FH Worms

TextGrid

Vernetzte Forschungsumgebung in den eHumanities

Projekt: TextGrid - Vernetzte Forschungsumgebung in den eHumanities
BMBF Förderkennzeichen: 01UG0901A
Laufzeit: Juni 2009 bis Mai 2012

Dokumentstatus: Finale Fassung
Verfügbarkeit: öffentlich
Autor:

Yahya Al-Hajj (FH Worms)
Waldemar Artes (FH Worms)
Thomas Selig (FH Worms)

Revisionsverlauf:

Datum	Autoren	Kommentare
02.11.2011	Al-Hajj, Artes, Selig	Finaler Entwurf
15.11.2011	Köhlmann, Lohmeier	Layout, Korrekturen, Ausblick

Inhaltsverzeichnis:

1. Einleitung

TextGrid ist ein Verbundforschungsprojekt mit 10 institutionellen Partnern und ca. 20 Entwicklern, das in kooperativer Zusammenarbeit aller Beteiligten entwickelt wird. Für die verteilte Arbeit ist der Einsatz von Tools nötig, um die Komplexität beherrschen zu können und um frühzeitig Fehlerquellen zu entdecken und zu beheben.

Der Bericht stellt dar, wie die folgenden Tools von den Entwicklern und anderen Projektbeteiligten für die Entwicklung und zum Testen der Software eingesetzt werden.

2. Entwicklungs- und Projektmanagementtools

2.1. Subversion

Apache Subversion[1] (SVN) ist ein Versionsverwaltungssystem zur Versionierung von Dateien und Verzeichnissen in einem zentralen Projektarchiv (Repository), wobei bei der Erstellung einer neuen Version nur die Änderungen mit den dazugehörigen Metadaten (z.B. Autor, Datum, Kommentar) im Repository gespeichert werden. Subversion ist eine freie Software unter der Apache-Lizenz[2].

Path		Last modification		Log	SVN	RSS
branches/	11455	2h 57m	jdabbert	Log	SVN	RSS
info.textgrid.middleware.confclient/	11250	33d 03h	nassourou	Log	SVN	RSS
tags/	11271	25d 02h	fugu	Log	SVN	RSS
trunk/	11443	3d 22h	haase	Log	SVN	RSS
admin/	992	1324d 14h	ludwig	Log	SVN	RSS
doc/	5672	598d 07h	haase	Log	SVN	RSS
lab/	11419	6d 03h	jdabbert	Log	SVN	RSS
authn/	11099	48d 03h	queens	Log	SVN	RSS
build/	11032	62d 10h	vitt	Log	SVN	RSS
core/	11144	42d 03h	vitt	Log	SVN	RSS
de.uni-wuerzburg.informatik.www1.fnquery.plugin/	6106	539d 01h	vitt	Log	SVN	RSS
digilib/	11379	12d 01h	vitt	Log	SVN	RSS
features/	11213	38d 05h	vitt	Log	SVN	RSS
help/	10673	105d 20h	queens	Log	SVN	RSS
info.textgrid.lab.conf/	11043	61d 08h	vitt	Log	SVN	RSS
info.textgrid.lab.core.metadataeditor/	11270	25d 04h	alhajj	Log	SVN	RSS
info.textgrid.lab.cosmas/	8833	251d 06h	wick	Log	SVN	RSS

Abbildung 1: Anzeige des TextGrid-Repositories in Subversion via WebSvn

Es gibt viele Tools von verschiedenen Herstellern und für die verschiedenen Betriebssystemen, die als SVN-Client (z.B. TortoiseSVN für MS-Windows) oder als Repository-Browser (z.B. WebSVN[3] - Abbildung 1) agieren.

SVN ist in vielen Tools und Entwicklungsumgebungen integriert. Für die Entwicklung von TextGrid-Tools benutzen wir hauptsächlich das eclipse-Framework[4] mit SVN-Integration.

[1] http://subversion.apache.org/
[2] http://svn.apache.org/repos/asf/subversion/trunk/LICENSE
[3] http://dev.digital-humanities.de/websvn/wsvn/TextGrid/

Die SVN-Funktionalität in eclipse können durch die Subclipse-Plugins[5] nachinstalliert werden.

In der folgenden Abbildung (Abbildung 2) sehen Sie die integrierte SVN-Funktionalität innerhalb des eclipse-Frameworks. Links in der Abbildung ist ein View, der den Inhalt des SVN-Repositories anzeigt. In der Mitte oben ist der Compare-View, der den Inhalt von zwei ausgewählten Versionen vergleicht und Unterschiede markiert und unten ist der History-View, der die Historie für eine selektierte Datei als Tabelle (alle Revisionen mit ihren Beschreibungen) anzeigt. Rechts ist der Navigator, der die lokalen Kopien von Repository-Projekten anzeigt - Alle Änderungen sollen in diesen lokalen Dateien gemacht werden und dann kann man die Änderungen ins Repository senden (check-in) - Diese Operation erzeugt dann eine neue Version im Repository.

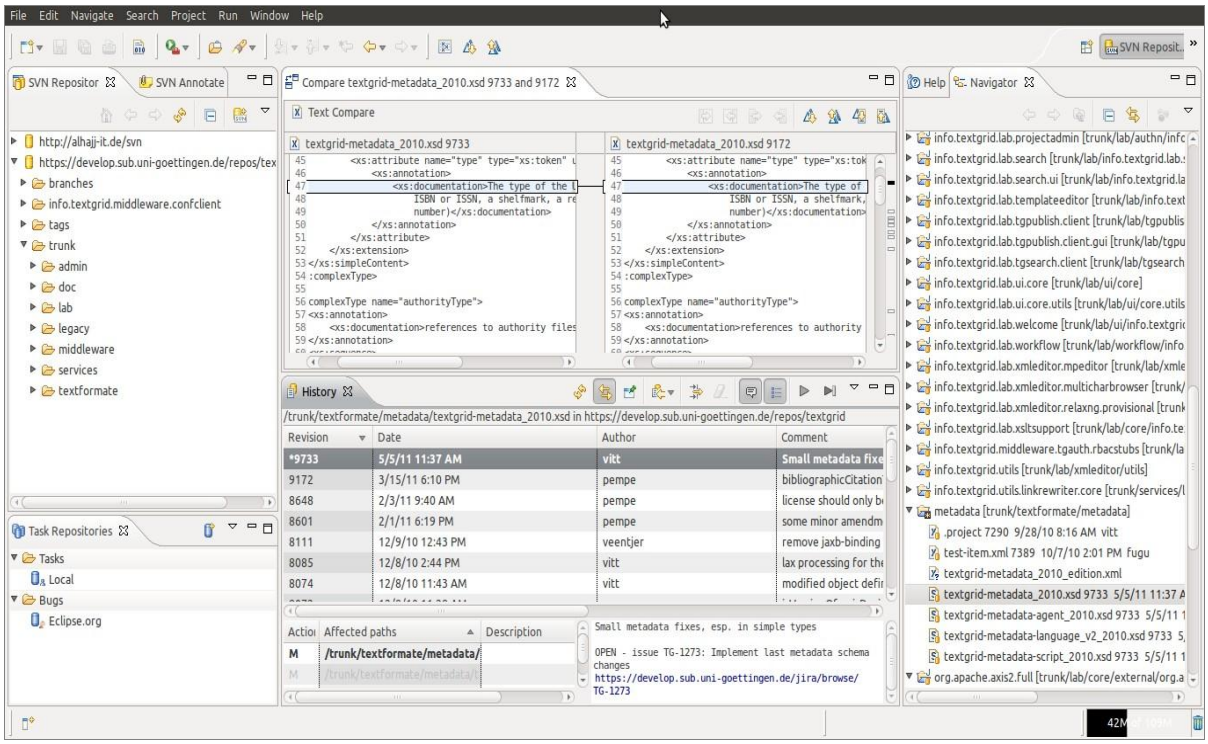


Abbildung 2: subclipse in eclipse

2.2. Jenkins

Jenkins[6] ist ein freies und quell-offenes webbasiertes System unter der MIT-Lizenz zur kontinuierlichen Integration (continuous integration) in Softwareprojekten. Das System wurde hauptsächlich in der Programmiersprache Java entwickelt und kann in einem beliebigen Servlet-Container (z.B. Apache Tomcat) laufen.

[4] http://www.eclipse.org
[5] http://subclipse.tigris.org
[6] http://jenkins-ci.org/

Jenkins bietet die Möglichkeit zur Erweiterung der System-Funktionalität durch die Anbindung einer Vielzahl von Plugins, die es erlauben, andere Tools wie CVS, SVN, Ant, Maven, usw. im System zu integrieren.

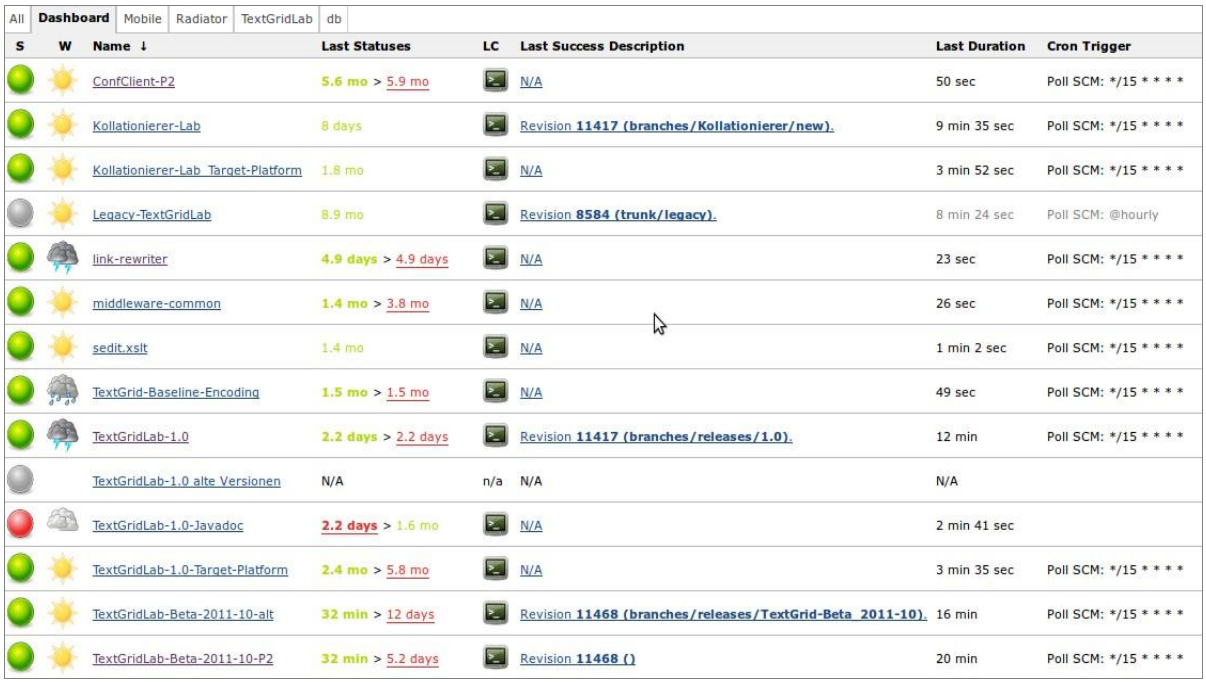
Es ist auch möglich eigene Erweiterungen für Jenkins mit wenigem Aufwand durch Plugins zu erstellen - Jenkins definiert dafür die sogenannten „extensibility points“, die als Schnittstellen oder abstrakte Klassen dienen. Im Wiki[7] von Jenkins ist dieses Thema unter dem Punkt „Extend Jenkins“ mit Beispielen dokumentiert.

Für TextGrid hat Jenkins die Aufgabe, sowohl die verschiedenen Releases (Beta und Produktiv) von TextGrid-Komponenten (insbesondere des TextGridLabs) als auch die kontinuierlich erstellten Nightly-Builds zu erstellen. Der Erfolg oder Misserfolg des Erstellungsprozesses wird grafisch im Web-Browser dargestellt und Fehler werden lokalisiert und an die Entwickler per E-Mail geschickt. Die erfolgreichen Builds und Releases werden dann zum Herunterladen zur Verfügung gestellt. Der Erstellungsprozess wird automatisch gestartet, aber nur wenn es Änderungen im Quellcode der betreffenden Komponente gibt; d.h. es gibt einen Job (Prozess), der periodisch abgefragt, ob jemand Änderungen ins Repository eingecheckt hat und zwar in einen Bereich, der die relevanten Komponente betrifft - Nur in diesem Fall wird ein Build-Prozess für diese Komponente gestartet.

In der folgenden Abbildung (Abbildung 3) sehen Sie einen Abschnitt der Jenkins-Webseite von TextGrid. Die erfolgreichen Builds werden in der ersten Spalte mit grünen Kreisen visualisiert und die erfolglosen mit den roten. Transparente Kreise bedeuten, dass der Erstellungsprozess für diese Komponenten deaktiviert ist.

In der zweiten Spalte sehen Sie Symbole, die man vom Wetterbericht kennt - sonnig bedeutet, dass die Erstellungsprozesse dieser Komponente in der letzten Zeit immer erfolgreich waren und die anderen Wettersymbole (bewölkt, regnerisch) werden entsprechend der Fehlerquote angezeigt. Die anderen Spalten zeigen andere Informationen über die Builds (Zeitpunkt, Beschreibung und Dauer des letzten Builds).

[7] https://wiki.jenkins-ci.org/display/JENKINS/Home



All | **Dashboard** | Mobile | Radiator | TextGridLab | db

S	W	Name ↓	Last Statuses	LC	Last Success Description	Last Duration	Cron Trigger
		ConfClient-P2	**5.6 mo** > 5.9 mo		N/A	50 sec	Poll SCM: */15 * * * *
		Kollationierer-Lab	8 days		Revision **11417 (branches/Kollationierer/new)**.	9 min 35 sec	Poll SCM: */15 * * * *
		Kollationierer-Lab_Target-Platform	1.8 mo		N/A	3 min 52 sec	Poll SCM: */15 * * * *
		Legacy-TextGridLab	8.9 mo		Revision **8584 (trunk/legacy)**.	8 min 24 sec	Poll SCM: @hourly
		link-rewriter	**4.9 days** > 4.9 days		N/A	23 sec	Poll SCM: */15 * * * *
		middleware-common	**1.4 mo** > 3.8 mo		N/A	26 sec	Poll SCM: */15 * * * *
		sedit.xslt	1.4 mo		N/A	1 min 2 sec	Poll SCM: */15 * * * *
		TextGrid-Baseline-Encoding	**1.5 mo** > 1.5 mo		N/A	49 sec	Poll SCM: */15 * * * *
		TextGridLab-1.0	**2.2 days** > 2.2 days		Revision **11417 (branches/releases/1.0)**.	12 min	Poll SCM: */15 * * * *
		TextGridLab-1.0 alte Versionen	N/A	n/a	N/A	N/A	
		TextGridLab-1.0-Javadoc	**2.2 days** > 1.6 mo		N/A	2 min 41 sec	
		TextGridLab-1.0-Target-Platform	**2.4 mo** > 5.8 mo		N/A	3 min 35 sec	Poll SCM: */15 * * * *
		TextGridLab-Beta-2011-10-alt	**32 min** > 12 days		Revision **11468 (branches/releases/TextGrid-Beta_2011-10)**.	16 min	Poll SCM: */15 * * * *
		TextGridLab-Beta-2011-10-P2	**32 min** > 5.2 days		Revision **11468 ()**	20 min	Poll SCM: */15 * * * *

Abbildung 3: Jenkins als Continuous-Integration-System von TextGrid[8]

Da die durch Jenkins erstellten Komponenten voneinander abhängig sein können, ist es wichtig, dass aller von einer Komponente abhängigen Build-Prozesse nach einem erfolgreichen Prozess automatisch gestartet werden - Dafür kann man in Jenkins die Abhängigkeiten zwischen den Komponenten definieren und diese Abhängigkeiten kann Jenkins grafisch darstellen.

In der folgenden Abbildung sehen Sie die grafische Darstellung der Abhängigkeiten zwischen den TextGrid-Komponenten, die von Jenkins verwaltet werden.

[8] http://dev.digital-humanities.de/ci/

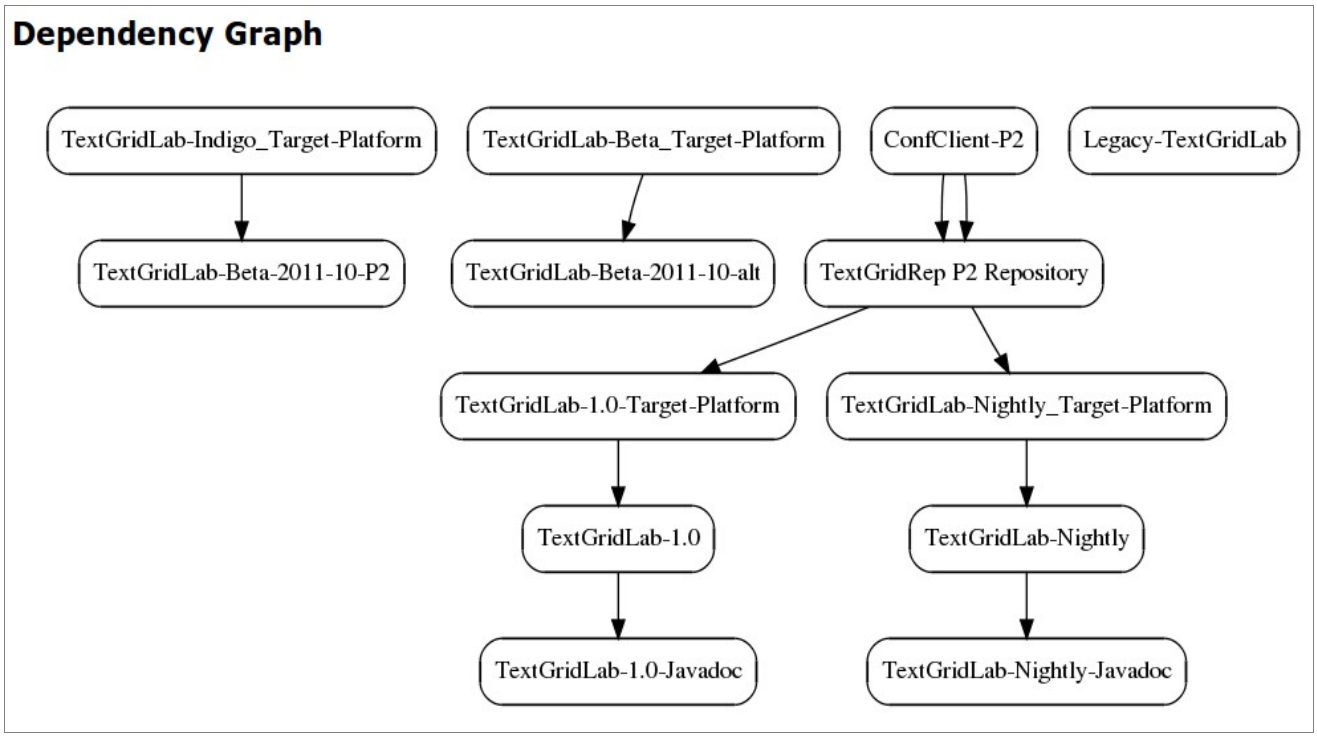


Abbildung 4: Die Abhängigkeiten zwischen den Komponenten in Jenkins

Die erfolgreichen Builds von TextGridLab stehen dann als Archive für die unterschiedlichen Betriebssysteme zum Herunterladen bereit. Hier in der folgenden Abbildung stehen 5 unterschiedliche Archive des TextGridLabs zur Auswahl, die nach dem Herunterladen und Entpacken sofort einsatzbereit sind.

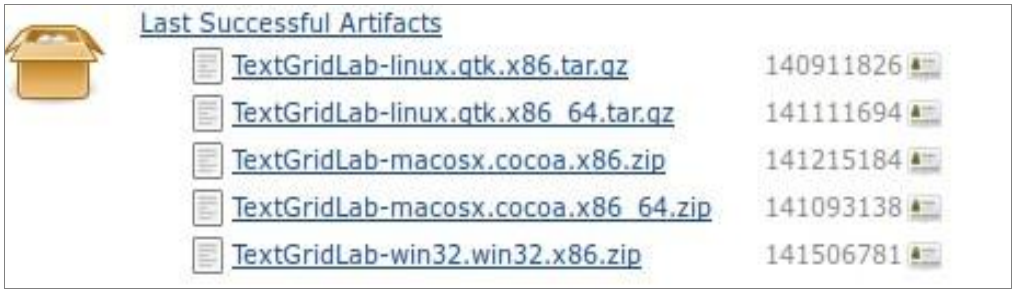


Abbildung 5: TextGridLab-Archiv in Jenkins

2.3. Jira

Jira[9] ist ein webbasiertes Issue-Tracking- und Projektmanagement-System, das auf der JEE-Technologie basiert und primär in Software-Projekten eingesetzt wird. Es wurde von der Firma „Atlassian“ entwickelt und ist ein kommerzielles Produkt. Open Source-Projekte oder gemeinnützige Einrichtungen und Organisationen dürfen Jira kostenfrei einsetzen.

[9] http://www.atlassian.com/software/jira/overview

Wie auch bei Jenkins kann man für Jira eigene Erweiterungen (Extensions und Plugins) erstellen, die dann der Community über die „Jira extension library“[10] zur Verfügung gestellt werden können.

In einer Jira-Anwendung können mehrere Projekte parallel verwaltet werden. In diesem Fall stehen diese Projekte auf der Startseite der Jira-Anwendung zur Auswahl bereit.

In TextGrid erfüllt Jira die folgenden Zwecke:

Issue-Tracking:

Vorgänge (Issues) können in TextGrid von jedem (solange die Zugriffsbeschränkung nicht explizit eingeschränkt wird) online gelesen werden aber nur von den berechtigten Personen erstellt und bearbeitet werden, die einen Jira-Account besitzen.

In TextGrid kann ein Issue einen der drei möglichen Typen haben:

1. Bug (Fehler)
2. Task (Aufgabe)
3. Improvement (Verbesserung)

Zu einem Vorgang kann der Ersteller mehrere Informationen angeben, die das Issue beschreiben. Die wichtigsten Eingaben sind dabei die folgenden:

- Der Titel des Vorgangs
- Die Beschreibung
- Die betroffene(n) Komponente(n)
- Das betroffene Release oder die betroffene Version
- Der Bearbeiter, zu dem der Vorgang zugewiesen werden soll (Kann ggf. mit Hilfe der betroffenen Komponente automatisch erkannt werden)

Nach der Erstellung des Vorgangs wird eine E-Mail an den zugewiesenen Bearbeiter geschickt. Der Bearbeiter kann dann den Vorgang selbst bearbeiten oder weiter an eine andere Person delegieren.

In der folgenden Abbildung sehen Sie einen Abschnitt der Jira-Startseite für das Projekt TextGrid. Hier stehen die fälligen und aktiven Vorgänge und ein Diagramm, das die Entwicklung der erstellten (rot) und erledigten (grün) Vorgänge in den letzten 30 Tagen anzeigt.

[10] https://plugins.atlassian.com/search/by/jira

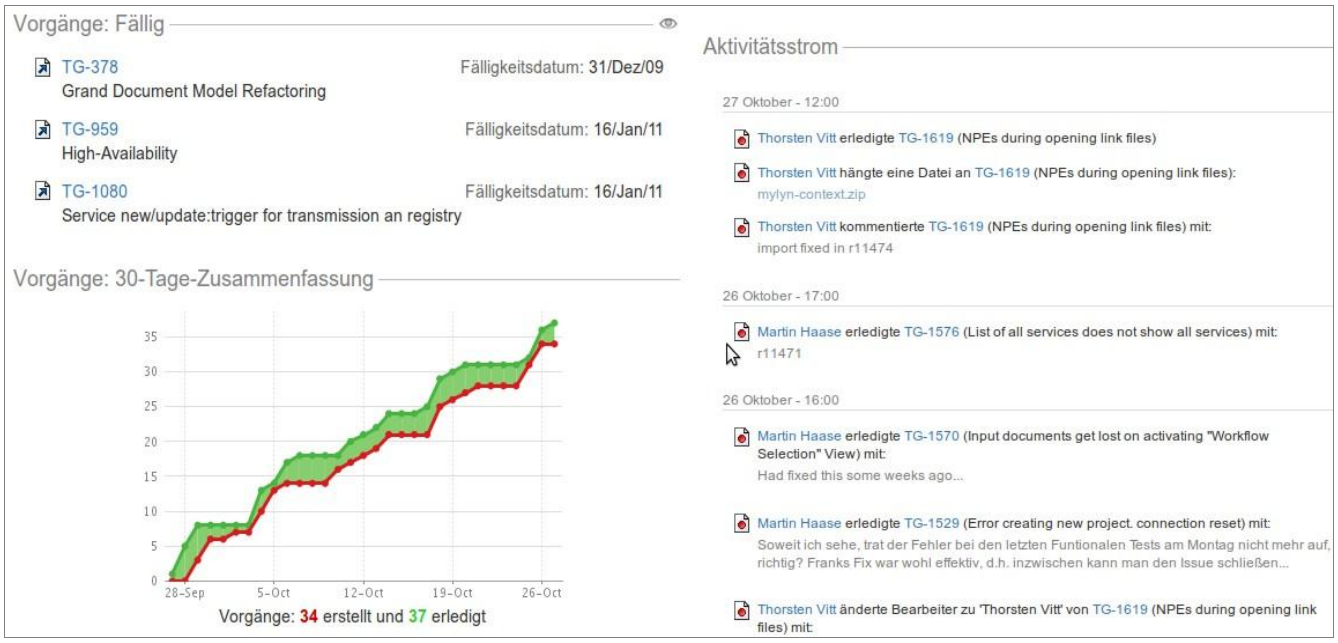


Abbildung 6: TextGrid in Jira

Release-Planung:

Ein neues Release enthält eine Gruppe von neuen Features, die implementiert werden sollen und Bug-Fixes, die zu einem bestimmten Zeitpunkt erledigt sein sollen. Genau das (die Erstellung von Features und Bug-Fixes) wird durch die Erstellung von Vorgängen (Issues) dokumentiert und es ist deswegen sehr sinnvoll, die Issue-Verwaltung und die Release-Planung in einem System zusammen zu bearbeiten.

In Abbildung 7 ist ein Beispiel für ein Release mit Namen und Beschreibung und unter der Beschreibung sehen Sie die Issues, die zu diesem Release gehören, wobei die ersten zwei Issues noch offen (nicht erledigt) sind. Eine grafische Darstellung sehen Sie oben rechts als Balken, der die geschlossenen Issues mit grün und die noch offenen mit rot anzeigt.

Abbildung 7: Release-Planung in JIRA

3. Test-Tools

Automatisierte Tests helfen frühzeitig Fehler aufzudecken und verbessern die Qualität der entwickelten Software. In TextGrid werden zwei Tools für automatisierte Tests eingesetzt: Sikuli für automatisierte funktionale Tests der TextGridLab-Tools und JMeter für Lasttests der Services und der Middleware.

3.1. Sikuli

Sikuli[11] ermöglicht GUI-Elemente automatisiert zu testen. Dadurch kann Benutzerverhalten imitiert und regelmäßig automatisiert ausgeführt werden.

Sikuli-Skripte sind in Python geschrieben und arbeiten mit Bilderkennung. So kann man nach einem Bildausschnitt auf dem Bildschirm suchen, um bestimmte Aktionen darauf auszuführen.

Die Entwicklung eines Tests erfolgt durch die Sikuli-IDE. Am häufigsten benutze Funktionen wie click, doubleClick oder type, findet man auf der linken Seite. So kann man z.B. mittels der Funktion *click()* einen Bildschirmausschnitt suchen und anklicken (Abbildung 8).

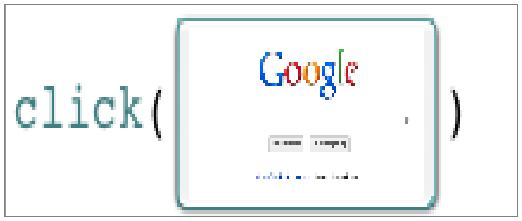


Abbildung 8

Durch erneutes Klicken auf das Bild kann man das Ähnlichkeitsmaß (Abbildung 9) und den Klickpunkt (Abbildung 10) anpassen.

[11] http://www.sikuli.org

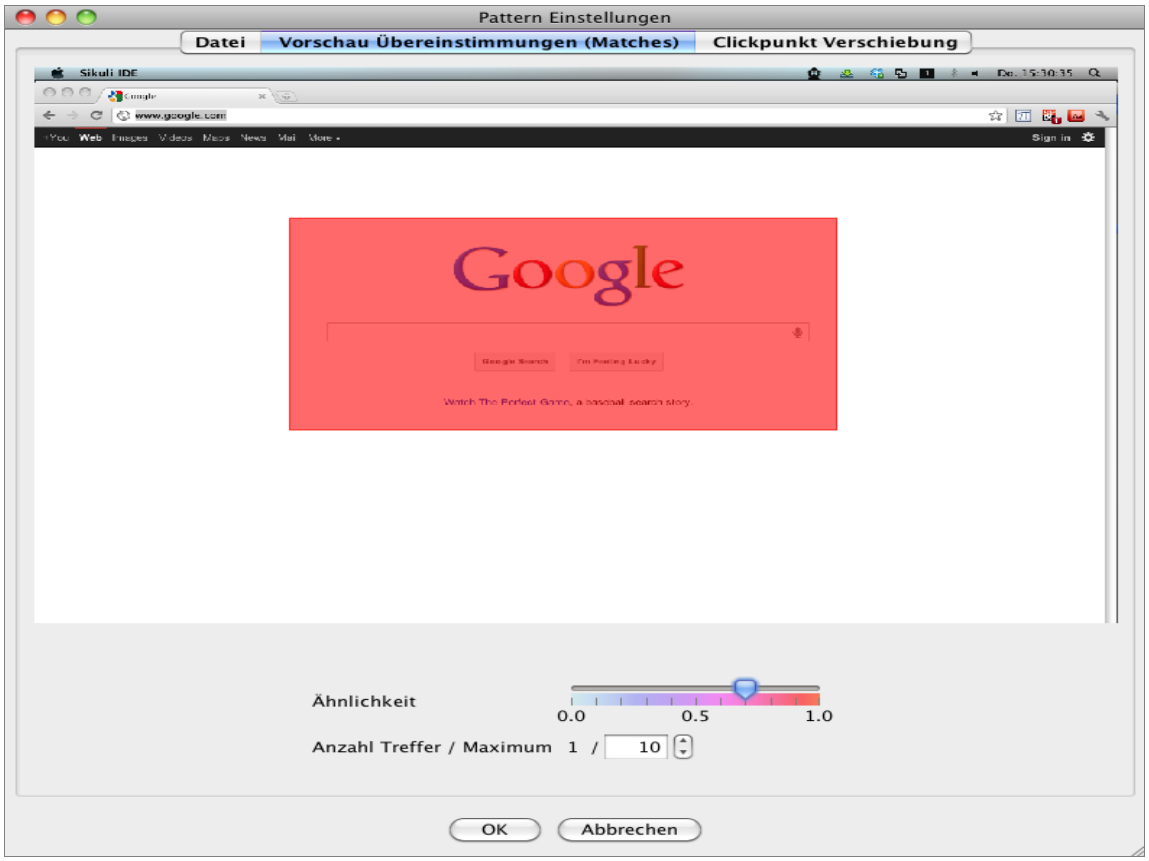


Abbildung 9

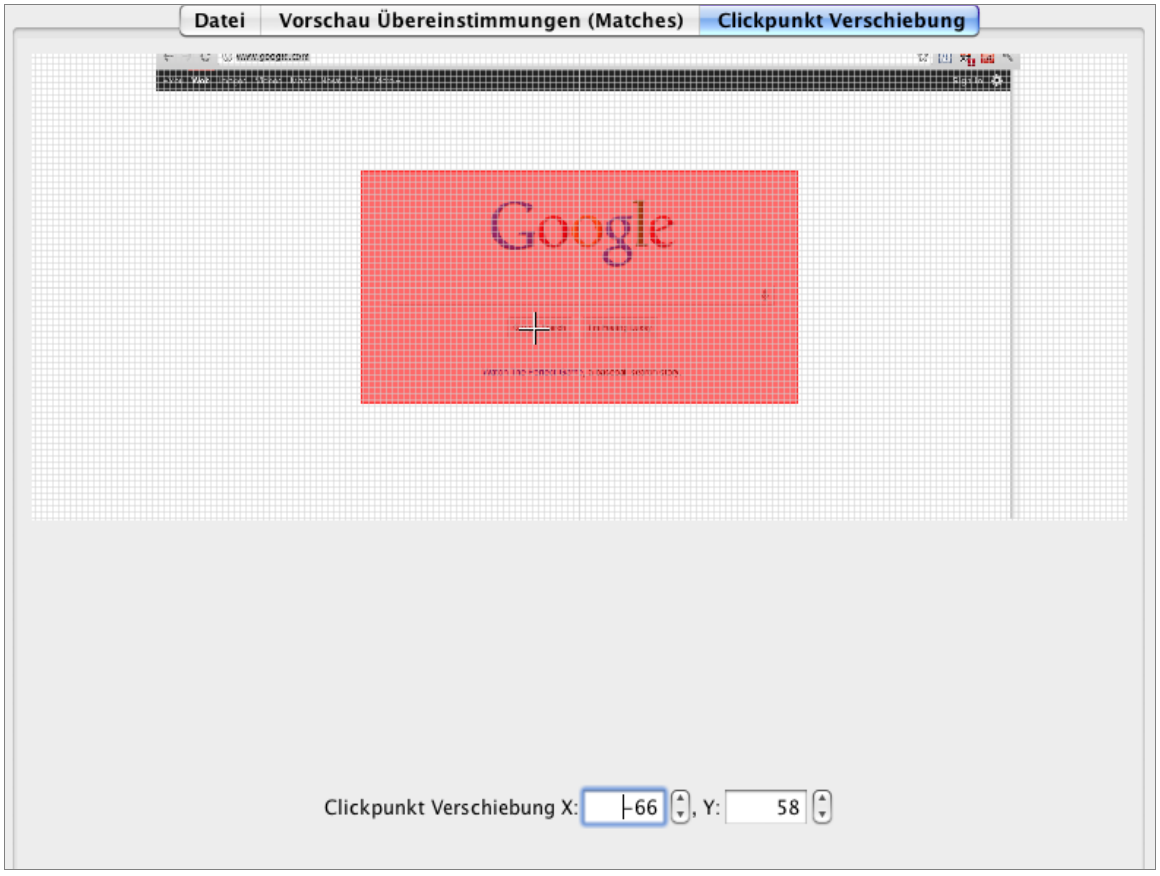


Abbildung 10

Im Verlauf der Entwicklung der Version 1.0 des TextGridLabs wurden viele solcher Tests parallel auf bis zu 21 Rechnern ausgeführt. So konnten 21 unterschiedliche Benutzer simuliert werden, die gleichzeitig mit dem TextGridLab arbeiten. Dabei konnten mehrere GUI-Bugs gefunden werden, die beim manuellen Testen nicht aufgefallen sind.

So wurde in seltenen Fällen ein Button ohne einen Grund ausgegraut. Auch wurde durch eine zwingende Autovervollständigung im Metadaten-Editor so viel Last erzeugt, dass das TextGridLab nicht mehr reagiert hat. Diese Fehler waren nicht durch einzelne Tests, sondern nur durch die Simulation von vielen gleichzeitigen Zugriffen zu erkennen.

Mängel in der Middleware konnten ebenfalls auf diese Weise aufgedeckt werden. Unter anderem konnte beim Importieren und Löschen großer Datenmengen beobachtet werden, dass die Stabilität des TG-Crud-Services (zuständig für die Dateiverwaltung im Grid) immer stärker abgenommen hat. Anschließende Lasttests zeigten, dass die eXist-Datenbank durch eine fehlerhafte Konfiguration instabil war.

In seltenen Fällen tauchte beim Erstellen eines Projektes eine Fehlermeldung auf mit dem wenig aussagekräftigen Inhalt: “1024” oder “4096” auf. Durch die Analyse der Logfiles konnte der Entwickler feststellen, dass eine fehlerhafte Version einer Fremdkomponente im TextGridLab verwendet wurde.

Durch einen Test, in dem ein TEI-Dokument editiert wurde, wurde gezeigt, dass die in TextGrid eingesetzte eXist-Datenbank bei großen Datenmengen in Verbindung mit vielen Update-Requests äußerst langsam lief. Die Testergebnisse konnten genutzt werden, um die Konfiguration von eXist zu optimieren, um die Performanz zu erhöhen.

Um zu verhindern, dass bei parallelen Testläufen nicht mehrere Clients mit demselben Benutzer arbeiten, wurde ein Webservice entwickelt, bei dem sich jeder Client die jeweiligen Logindaten abholen konnte. Abbildung 11 zeigt die Methode im Sikuli-Skript, die Logindaten vom Webservice abholt.

```
# This method downloads the credential data from the webservice
def getCredentials(self):
    url = urllib.urlopen(self.credentialsUrl)
    credentials = url.read()
    url.close()
    self.username, self.password = credentials.split("/")
    print "Info: credentials are ", self.username, " with ", self.password
    if self.password is None or self.password == "":
        print "Error: No credentials found"
        quit(1)
```

Abbildung 11

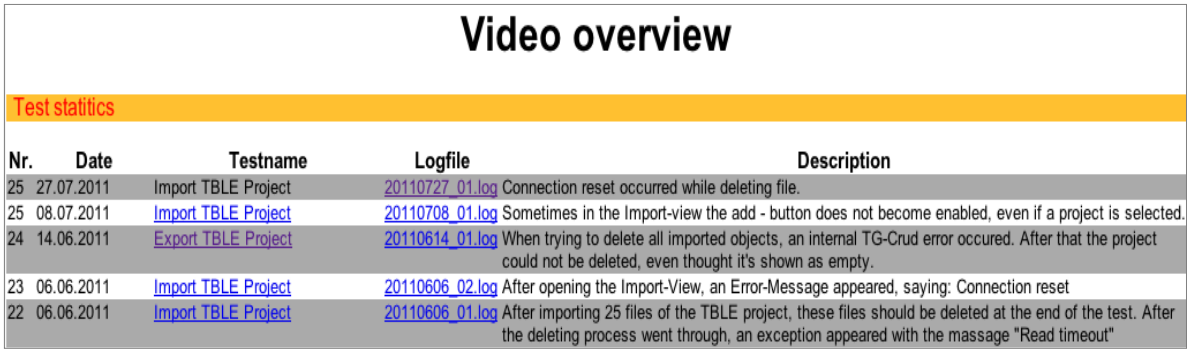
Im Verlauf der Entwicklung wurde die Testsuite so umfangreich, dass die Ausführung der gesamten Suite zu viel Zeit in Anspruch nahm. Um eine Auswahl der auszuführenden Tests zu ermöglichen, wurde die Suite in kleinere autonome Suites aufgeteilt. Durch einen weiteren Webservice konnte bestimmt werden, welche Suites auszuführen sind. Abbildung 12 zeigt die entsprechende Stelle im Sikuli-Skript.

```
class TestConfiguration(object):
    def __init__(self):
        self.configurationUrl = "http://unit3.ztt.fh-worms.de:8080/SikuliDataBroker/app/testConfiguration"
        self.configValue = 1
        self.getConfigurationValue()

    def getConfigurationValue(self):
        url = urllib.urlopen(self.configurationUrl)
        self.configValue = int(url.read())
        url.close()
```

Abbildung 12

Jeder Testlauf wurde mittels einer unabhängigen Software („Screenflow“ siehe unten) aufgezeichnet und Videos von Fehlern zusammen mit den Logfiles des TextGrigLabs wurden auf einer Webseite zusammengefasst (Abbildung 13).



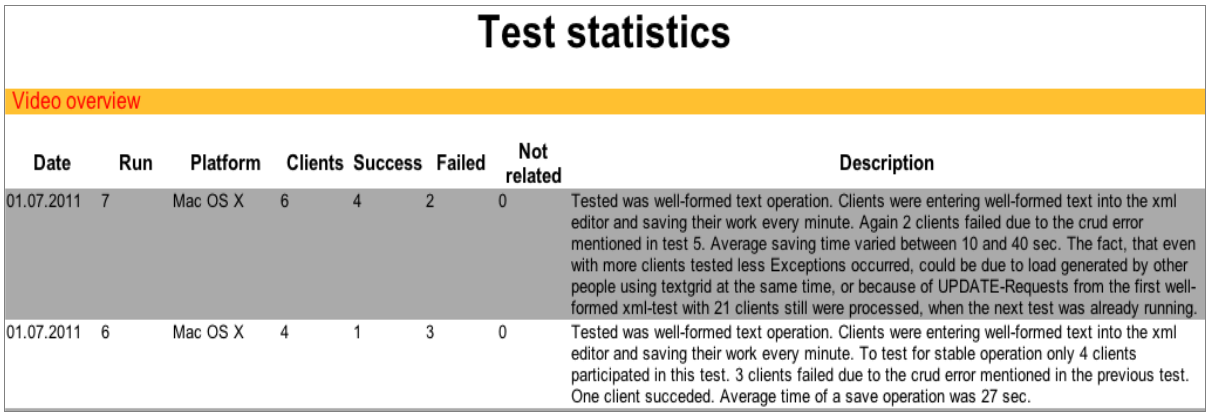
Video overview

Test statitics

Nr.	Date	Testname	Logfile	Description
25	27.07.2011	Import TBLE Project	20110727_01.log	Connection reset occurred while deleting file.
25	08.07.2011	Import TBLE Project	20110708_01.log	Sometimes in the Import-view the add - button does not become enabled, even if a project is selected.
24	14.06.2011	Export TBLE Project	20110614_01.log	When trying to delete all imported objects, an internal TG-Crud error occured. After that the project could not be deleted, even thought it's shown as empty.
23	06.06.2011	Import TBLE Project	20110606_02.log	After opening the Import-View, an Error-Message appeared, saying: Connection reset
22	06.06.2011	Import TBLE Project	20110606_01.log	After importing 25 files of the TBLE project, these files should be deleted at the end of the test. After the deleting process went through, an exception appeared with the massage "Read timeout"

Abbildung 13

Dazu wurden noch Statistiken über die einzelnen Testläufe ergänzt (Abbildung 14).



Test statistics

Video overview

Date	Run	Platform	Clients	Success	Failed	Not related	Description
01.07.2011	7	Mac OS X	6	4	2	0	Tested was well-formed text operation. Clients were entering well-formed text into the xml editor and saving their work every minute. Again 2 clients failed due to the crud error mentioned in test 5. Average saving time varied between 10 and 40 sec. The fact, that even with more clients tested less Exceptions occurred, could be due to load generated by other people using textgrid at the same time, or because of UPDATE-Requests from the first well-formed xml-test with 21 clients still were processed, when the next test was already running.
01.07.2011	6	Mac OS X	4	1	3	0	Tested was well-formed text operation. Clients were entering well-formed text into the xml editor and saving their work every minute. To test for stable operation only 4 clients participated in this test. 3 clients failed due to the crud error mentioned in the previous test. One client succeded. Average time of a save operation was 27 sec.

Abbildung 14

Zur Aufzeichnung der Tests wurde die Software „Screenflow“ benutzt. Screenflow[12] ist ein Tool für Mac OS X, mit dem Screencasts erstellt werden können. Benutzt wurde es im Rahmen von TextGrid zum Aufzeichnen der funktionalen Sikuli-Tests. Die Videos helfen den Entwicklern Fehler besser reproduzieren zu können. Oder diese zumindest sehen zu können, falls das Reproduzieren nicht möglich ist. So war in einem Fall, ein Button manchmal ausge-

[12] http://screenflow.softonic.de/mac

graut und manchmal nicht. Dieser Fehler trat nur bei den funktionalen Tests auf und ließ sich auf dem Rechner des Entwicklers nicht reproduzieren.

Die aufgezeichneten Videos lassen sich nur mit Screenflow öffnen, dort bearbeiten und dann in ein Flash-Video oder in andere Formate rendern. Mit Screenflow kann man unter anderem Schrift und Formen einfügen, um Fehler besser hervorzuheben. Außerdem kann man die Videos zurechtschneiden und skalieren. In der Trial-Version wird beim Rendern ein Wasserzeichen eingefügt.

Ein großer Vorteil von Screenflow gegenüber manchen anderen Screencast-Tools ist die Möglichkeit, es über die Kommandozeile zu starten und zu beenden. Diese Funktion wurde benötigt, um den gesamten Testprozess mittels eines Scripts zu starten. Die fertigen Dateien wurden nach Beenden des Tests ebenfalls mit einem Script eingesammelt, um sie später auswerten zu können.

3.2. JMeter

Da die Middleware von vielen TextGridLab-Clients angesprochen wird und dabei auch größere Lasten entstehen können, ist es wichtig zu testen, wie sich die einzelnen Komponenten unter Last verhalten. Für solche Lasttests wurde das Tool Apache JMeter[13] benutzt. Mit diesem Tool lassen sich unter anderem Lasttests auf Webservices über SOAP- und HTTP-Requests realisieren. Um die Last zu steigern, kann man neben einer beliebigen Anzahl von Threads, die Tests auch auf mehreren Rechnern parallel ausführen.

In unseren Tests benutzten wir 4 Server mit je 15 Threads.

Abbildung 15 zeigt, wie ein kompletter JMeter-Testplan aussehen kann.

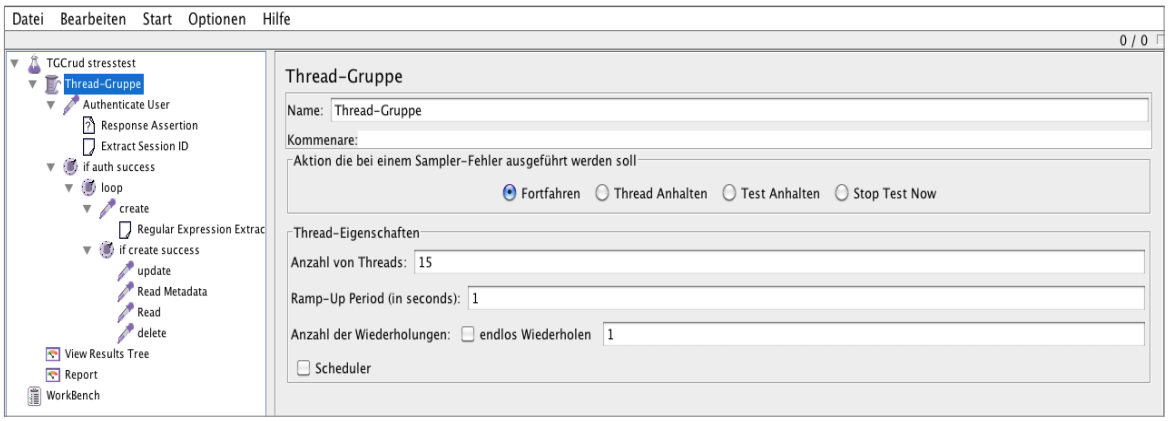


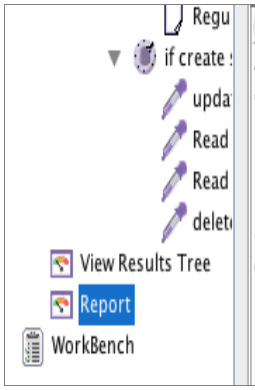
Abbildung 15

Zu einem Testplan kommt zunächst eine Thread-Group. Darin legt man fest, wie viele Threads laufen sollen. In diese Thread-Group kommen nun die einzelnen Requestsampler und Logikcontroller, die man für seine Tests verwenden will. So kann man mit einem HTTP-Requestsampler einfache HTTP-Anfragen verschicken und mit einer SOAP/XML-RPC-

[13] http://jakarta.apache.org/jmeter/

Anfrage kann man mit einem SOAP-Service kommunizieren.[14] Mittels des JavaRequest-Samplers können auch eigene Java-Klassen ausgeführt werden, um so Funktionalitäten zu implementieren, die JMeter nicht bereitstellt[15], beispielsweise die Kommunikation mit TG-Crud mittels eines Crud-Clients.

Die Statistiken des Testlaufs können in einem Report angezeigt werden (Abbildung 16).



Label	Anz. der Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% Fehler	Durchsatz
Authenticate User	1	1958	1958	1958	1958	1958	0,00%	30,6/min
create	1	3255	3255	3255	3255	3255	0,00%	18,4/min
update	1	1272	1272	1272	1272	1272	0,00%	47,2/min
Read Metadata	1	378	378	378	378	378	0,00%	2,6/sec
Read	1	293	293	293	293	293	0,00%	3,4/sec
delete	1	653	653	653	653	653	0,00%	1,5/sec
Gesamt	6	1301	653	1958	293	3255	0,00%	39,6/min

Abbildung 16

JMeter wurde benutzt, um die Performance der Middleware-Komponenten TG-Crud, TG-Search und TG-Auth sowie die eXist-Datenbank zu testen.

Am interessantesten waren dabei die Tests der eXist-Datenbank. In diesen Tests wurden Aktionen ausgeführt, die Daten in eXist speichern, aktualisieren, lesen und löschen. Dabei ist anfangs ein Deadlock aufgetreten. Als Reaktion darauf wurde die Testsuite mit den Logfiles an die Entwickler geschickt, um dort den Fehler reproduzieren zu können. Kurze Zeit später konnten die eXist-Entwickler den Fehler beheben, was jedoch erhebliche Performance-Verluste mit sich brachte.

Durch Lasttests auf TG-Crud konnte festgestellt werden, dass abgelaufene TCP-Verbindungen teilweise nicht mehr geschlossen wurden, woraufhin man keine neuen Verbindungen mehr öffnen konnte und im TextGridLab eine Meldung mit der Nachricht „Too many open files“ erhielt, wenn man versucht hat viele Objekte zu löschen.

Die Ergebnisse wurden zusammengefasst und im internen Wiki[16] von TextGrid für alle Entwickler zur Verfügung gestellt.

4. Kollaborations-Tools

Für die Abstimmung in der Zusammenarbeit wurden unterschiedliche Tools eingesetzt. Diese Tools kann man zweckmäßig in Kommunikationstools und Tools für die kollaborative Bearbeitung von Texten unterscheiden.

[14] http://jakarta.apache.org/jmeter/usermanual/index.html
[15] http://jakarta.apache.org/jmeter/usermanual/component_reference.html#Java_Request
[16] http://www.textgrid.de/intern/wiki1/wiki/Lasttests.html

4.1. Kommunikation

Für die Kommunikation werden neben der typischen asynchronen Kommunikation mittels Austausch von E-Mails zusätzlich die folgenden Kommunikationsanwendungen zur synchronen Kommunikation eingesetzt.

4.1.1. Skype

Es ist typisch, dass die TextGrid-Entwickler während ihrer Arbeitszeit in Skype online sind, sodass die direkte Kommunikation bei Bedarf erleichtert wird. Skype wird oft benutzt, um Audio-Konferenzen zwischen den Entwicklern zu ermöglichen.

4.1.2. Internet Relay Chat (IRC)

Eine andere Möglichkeit zur synchronen textbasierten Kommunikation ist der Internet Relay Chat (IRC). Diese Kommunikationsmöglichkeit wird im TextGrid-Projekt beispielsweise während des gemeinsamen Testens eingesetzt. Dabei laden alle Beteiligten eine aktuelle Version des TextGridLabs und testen bestimmte Tools zur gleichen Zeit. Beim Testen sind alle Beteiligten (inklusive Entwickler) im IRC-Kanal von TextGrid online, um Fragen zu beantworten und um Unklarheiten zu klären.

Neben den menschlichen Chat-Nutzern haben wir zwei virtuelle Nutzer, die immer im Chat online sind „cophi-ci“ als Bot für das Jenkins-System (siehe 1.2) und „subici“ für das Jenkins-System der SUB. Diese zwei Bots haben die Aufgabe, die Jenkins-Meldungen (z.B. Fehlermeldungen) im Chat durch eine Mitteilung bekannt zu machen.

4.2. Kollaborative Bearbeitung von Texten

Im Laufe des Projekts entstehen große Mengen von Texten, die für die Koordination der Entwicklung und für die Dokumentation interessant sein können. Deswegen ist es sinnvoll, diese Texte für alle Beteiligten auf eine einfachen Art und Weise zur Verfügung zu stellen. Es ist auch effizient, allen Beteiligten die Möglichkeit der aktiven Texterstellung und -bearbeitung zu gewähren. Für die kollaborative Bearbeitung von Texten haben wir vor allem die folgenden zwei Web-Anwendungen eingesetzt.

4.2.1. Wiki

Das projektinterne TextGrid-Wiki ist der Ort, in dem alle projektspezifischen Daten zur Verfügung gestellt werden. Im Wiki stehen die Anträge, Dokumentationen, Folien und Protokolle von internen und externen Veranstaltungen, Berichte und Publikationen über das Projekt und vieles mehr. Die Inhalte des Wikis sind nur für die Projekt-Beteiligten zugänglich.

4.2.2. EtherPad

EtherPad[17] ist ein webbasierter Editor zur synchronen und kollaborativen Bearbeitung von Texten in Echtzeit. Das bedeutet: mehrere Personen können ein Textdokument gleichzeitig ohne Verzögerungen oder Synchronisationskonflikte bearbeiten. Neben der Möglichkeit der

[17] http://etherpad.org/

kollaborativen Textbearbeitung hat EtherPad ein eingebettetes Chat-Fenster zur Kommunikation zwischen den Bearbeitern. Der kollaborativ bearbeitete Text kann dann als Dokument in verschiedenen Formaten (plain text, HTML, PDF, Open Document, MS Word) lokal gespeichert werden.

In der folgenden Abbildung sehen Sie ein Beispiel für ein Dokument in EtherPad. In diesem Beispiel sind drei Benutzer online und jeder Benutzer hat eine bestimmte Farbe, damit man unterscheiden kann, was jeder Benutzer geschrieben hat. Unten rechts sehen Sie den Chat-Bereich.

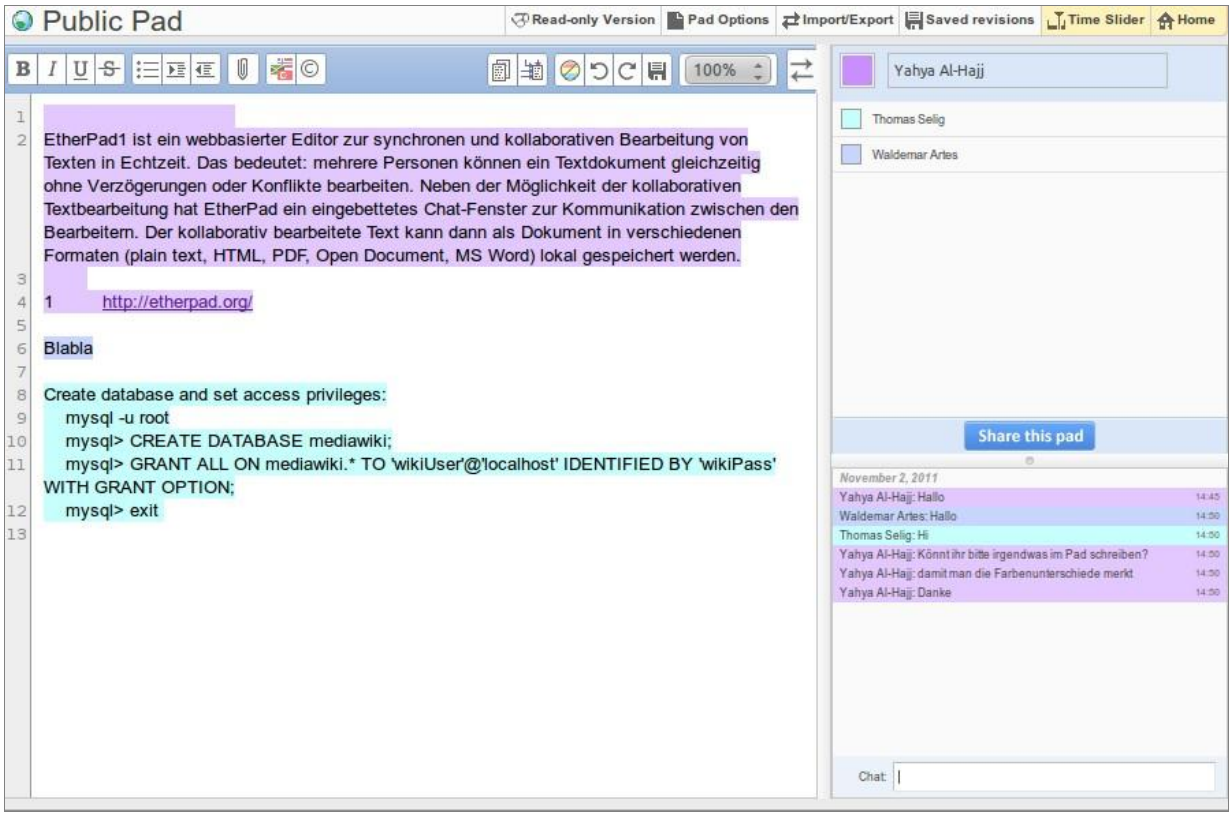


Abbildung 17: EtherPad in Interaktion

In TextGrid wird EtherPad hauptsächlich zur kollaborativen Protokollierung und zum Informationsaustausch in Jour-Fixe-Sitzungen und Entwickler-Programmiersprints eingesetzt.

5. Zusammenfassung und Ausblick

TextGrid verfügt über einige Entwicklungs-, Test- und Kollaborations-Tools, die speziell für das Projekt angepasst und konfiguriert wurden und an verschiedenen Standorten (z.B. Würzburg, Worms und Göttingen) gepflegt werden. Mit Hilfe dieser Tools lässt sich die Entwicklung augenblicklich sehr effizient gestalten und gut dokumentieren. Zum Projektabschluss steht die Entwicklungsumgebung in TextGrid vor zwei Herausforderungen:

1. Die Öffnung und Dokumentation der Tools für externe Entwickler, um die technischen Voraussetzungen für eine Open Source Developer Community zu schaffen.

2. Die Gewährleistung einer nachhaltigen Verfügbarkeit der Tools und der Dokumentation auch über das Projektende hinaus.

Für (2) ist eine enge Zusammenarbeit mit dem DARIAH-Projekt[18] geplant, die mittelfristig eine Entwicklungsinfrastruktur für Open Source Projekte zur Verfügung stellen werden.

[18] http://de.dariah.eu