HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

ita Institute for Internet Technologies and Applications

Vertrauenswürdiges und robustes Bulletin Board für E-Voting

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2010

Autoren:	Marco Hofstetter Marco Schälle
Betreuer:	Prof. Dr. Andreas Steffen
Projektpartner:	ITA, Rapperswil
Experte:	-
Gegenleser:	-

Datum: 22. Dezember 2010

Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.

Rapperswil, 22. Dezember 2010

Marco Hofstetter Marco Schälle

Inhaltsverzeichnis

1 Einführung 5
1.1 Aufgabenstellung 5
1.2 Aufbau der Arbeit 5
2 Konzepte 6
2.1 Verteiltes Web-Bulletin-Board nach R. Krummenacher 6
2.1.1 Anforderungen 6
2.1.2 Lösungsvariante 6
2.1.3 Ablauf 7
2.1.4 Verkettung der Historyeinträge 7
2.2 Practical Threshold Signatures nach Shoup 8
2.2.1 Anwendung auf's Web-Bulletin Board 9
3 Umsetzung 10
3.1 Vorausgehende Überlegungen 10
3.2 Web-Bulletin-Board 10
4 Deployment 11
4.1 Single-Machine-Simulation 11
4.2 Multi-Machine-Simulation 11
4.3 Single-Component-Machine 12
5 Architektur 13
5.1 JAR-Abhängigkeiten 13
5.2 Package-Overview 13
5.3 Packagebeschreibungen 14
5.3.1 simulation 14
5.3.2 wbb 15
5.3.3 implementation 16
5.3.4 verification 18
5.3.5 publishing 19
5.3.6 environment 20
5.3.7 aspects 21
5.3.8 script 23
5.3.9 messages 23
5.3.10 configuration 24
5.3.11 crypto 24
5.3.12 persister 25
5.3.13 logging 26
6 Testszenarien 27
6.1 Feststellen eines falschen Nachrichten-Hashes von einem Writer 27
6.2 Nichtzustandekommen der verteilten Threshold-Signatur über ein Threshold-Set 28
6.3 Feststellen einer falschen verteilten Threshold-Signatur eines Boards 29
6.4 Feststellen einer falschen partiellen Threshold-Signatur eines Boards 30
6.5 Nichterreichen des Threshold bei BoardPublikationen 31
6.6 Versuch ein Board mit ReadForAppending-Requests zu überlasten 32
6.7 Mehrere korrupte Writer blockieren das Threshold-Set 34
6.8 Reader stellt nachträglich geänderte Historyentries auf Boards fest 35
6.9 Die Boards besitzen eine zu kleine Anzahl an Histories für die Anzahl Writer im WBB-Environment 36
7 Installation 37
7.1 WBB Studio (& Postgres) 37
7.2 Einrichten der Entwicklungsumgebung 38

7.3 ANT Auto-Deployment ... 40
7.3.1 Standard Build-Tasks ... 40
7.3.2 Remote Build-Tasks ... 41
7.4 Verwendete Komponenten ... 41
8 Benutzerhilfe ... 42
8.1 Konsole ... 42
8.1.1 Absetzten eines Kommandos ... 42
8.1.2 Ausgabe ... 43
8.2 Skriptmodul ... 43
8.2.1 Kontext ... 43
8.2.2 Hilfe ... 43
8.2.3 Globale Kommandos ... 43
8.2.4 Alias ... 43
8.2.5 Simulation ... 44
8.2.5.1 Szenario ... 44
8.2.5.2 Aspekt ... 44
8.2.5.3 Referenz ... 44
8.2.6 Komponentenkonfiguration ... 49
8.2.7 Aspektkonfiguration ... 49
8.2.8 Simulationskonfiguration ... 49
8.3 Histories ... 50
8.3.1 Einträge aktualisieren ... 50
8.3.2 Filtern von Einträgen ... 50
8.3.3 Verändern von Einträgen ... 50
8.4 Zugriff per SSH ... 51
8.5 Zugriff per Webservice ... 51
9 Offene Punkte ... 52
10 Glossar ... 54
11 Projektmanagement ... 55
11.1 Erfahrungsberichte ... 55
11.1.1 Marco Hofstetter ... 55
11.1.2 Marco Schälle ... 56
11.2 Projektplan ... 57
12 Referenzen ... 58
13 Anhänge ... 59

1 Einführung

1.1 Aufgabenstellung

Moderne E-Voting Verfahren erlauben es dem Wähler jederzeit zu verifizieren, dass seine Stimme richtig zum Gesamtresultat beigetragen hat. Dies wird über ein öffentliches, meist Web-basiertes Bulletin-Board (WBB) erreicht, das alle individuellen Wählerstimmen in verschlüsselter Form registriert und auch die Aufsummierung zum Gesamtresultat protokolliert. Dabei ist es wichtig, dass sämtliche Einträge auf dem Bulletin-Board weder modifiziert noch gelöscht werden können. Aus Gründen der Verfügbarkeit soll das Bulletin-Board redundant ausgebildet sein, wobei die Konsistenz der Einträge gewährleistet werden muss.

Im Rahmen einer Seminararbeit hat der MSE Student Roland Krummenacher ein praktisches Verfahren entwickelt, das alle Anforderungen an ein vertrauenswürdiges und robustes Bulletin-Board erfüllt. Ausgehend von diesem Konzept soll in der Studienarbeit ein verteiltes WBB praktisch umgesetzt werden. Die zu wählende Web-Technologie ist dabei offen gestellt, es sollen aber Open Source Komponenten verwendet werden, um dem WBB eine möglichst grosse und schnelle Verbreitung zu ermöglichen, wie es z.B. vom U.S. National Institute of Standards and Technology (NIST) gewünscht wird.

Die Ausgestaltung der Benutzeroberfläche ist in dieser Arbeit völlig sekundär und kann unter Umständen sogar völlig weggelassen werden. Es geht primär um die zuverlässige und fälschungssichere Replizierung von Daten auf verteilten Servern.

1.2 Aufbau der Arbeit

Dieses Dokument befasst sich mit der gesamten Dokumentation des Projektes ‚Vertrauenswürdiges und robustes Bulletin Board für E-Voting'. Im ersten Abschnitt geht dieses Dokument auf die Grundlagen des Themas und in das Konzept welches dahinterliegt ein. In einem zweiten Teil die Architektur der Software im Detail erläutert. Danach werden die Szenarien aus Herr Krummenacher's Paper mit unserer Implementation auf ihre Korrektheit geprüft, währen am Schluss auf die Bedienung des WBB Studio eingegangen wird.

2 Konzepte

2.1 Verteiltes Web-Bulletin-Board nach R. Krummenacher

2.1.1 Anforderungen

Ein robustes Web-Bulletin-Board muss nach Heather & Lundin folgende Anforderungen erfüllen.

- Eine publizierte Nachricht darf im Nachhinein nicht mehr editiert werden.
- Eine publizierte Nachricht darf im Nachhinein nicht mehr aus der History entfernt werden.
- Neue Nachrichten können nur an die bestehende Historie des Boards angehängt und nicht dazwischen eingefügt werden.
- Das Board ist öffentlich, es kann somit jeder die publizierten Nachrichten lesen.
- Das Board selber darf keine Nachrichten generieren.
- Der Zustand des Boards kann jeder Zeit geprüft werden.
- Das Board ist zuständig für die Überprüfung der Nachrichten bei der Publikation.

Gemäss der Erweiterung durch R. Krummenacher muss ein *verteiltes* Web-Bulletin-Board zusätzlich die folgenden Anforderungen erfüllen:

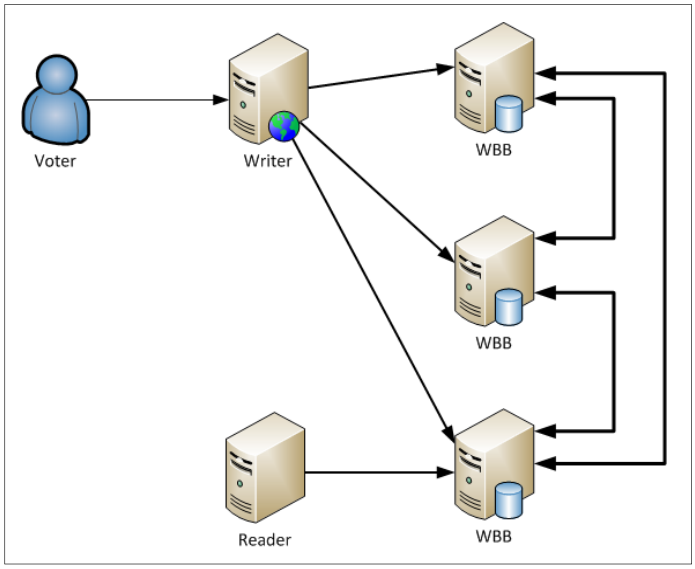
- Eine Nachricht muss parallel auf mehrere WBB's publiziert werden.
- Korrupte Boards dürfen keinen negativen Einfluss auf den Betrieb des Web-Bulletin-Boards haben.

2.1.2 Lösungsvariante

Basierend auf den Überlegungen von R. Krummenacher haben wir uns dazu entschieden die Variante ‚Asynchron verteiltes Web-Bulletin-Board' mit mehreren Histories pro WBB umzusetzen. Die im Folgenden beschriebene Variante besteht aus den Komponenten ‚WBB', ‚Writer' und ‚Reader'.

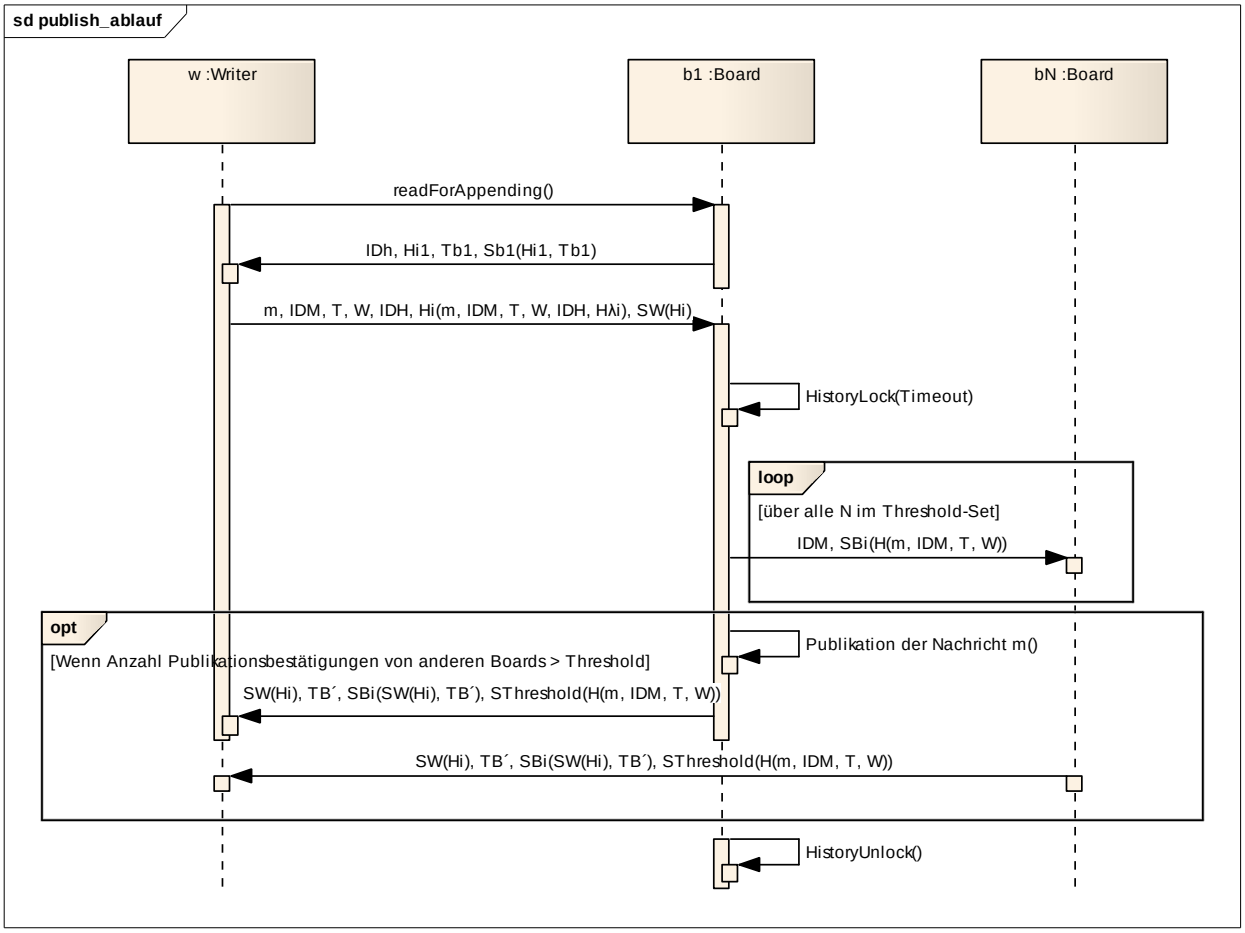
Bei dieser Variante wird vorgesehen, dass ein Writer seine Nachricht unabhängig auf alle WBB's innerhalb seines dazugehörigen Threshold-Sets verteilt. Zur Sicherstellung, dass die Nachricht auch auf genügend vielen WBB's publiziert wurde, benötigt der Writer die gültige verteilte Threshold-Signatur von mindestens einem Board. Damit ein WBB wiederum eine solch gültige verteilte Threshold-Signatur ausstellen kann, benötigt es die Bestätigung von mindestens k-1 anderen WBB's, dass diese eine Nachricht ebenfalls auf ihrer History publizieren werden. Somit kann der Writer mit nur einer korrekt erhaltenen verteilten Threshold-Signatur sicher sein, dass seine Nachricht auf genügend Boards publiziert wurde.

Hat sich das einzelne WBB dazu entschlossen eine Nachricht eines Writers zu publizieren, so wird die Meldung wie beim einfachen Web-Bulletin-Board mit der letzten Meldung auf der History verknüpft um ein nachträgliches Ändern oder Herauslöschen einer Nachricht aus der History zu verhindern.

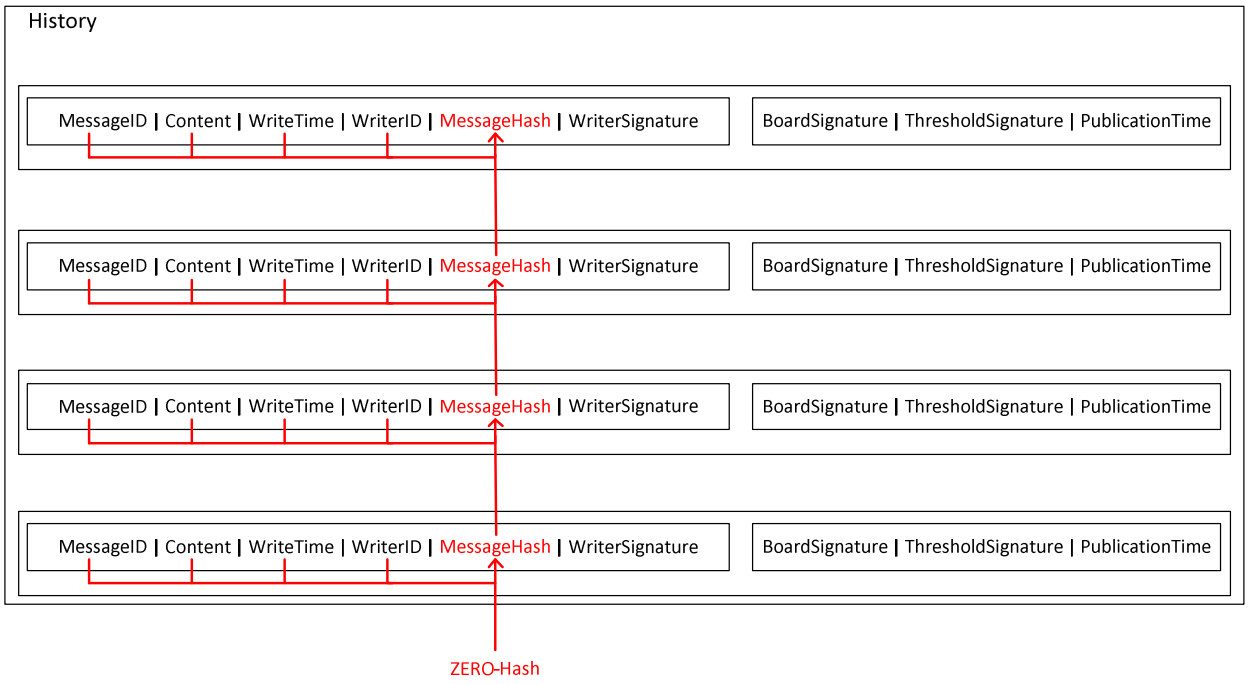


Ein WBB verwaltet zudem mehrere Histories, was dazu führt, dass das System viel besser skaliert wenn mehrere Writer zeitgleich eine Nachricht publizieren wollen. Mit nur einer History pro WBB würde das System blockieren, da die entsprechende History jeweils nur eine Publikation einer Nachricht übernehmen kann und während dieser Zeit gelockt werden muss. Mit mehreren Histories kann eine History gelockt werden und einem nachfolgenden Writer die nächste History zugewiesen werden.

2.1.3 Ablauf



2.1.4 Verkettung der Historyeinträge



2.2 Practical Threshold Signatures nach Shoup

Die Arbeit von R. Krummenacher über das Verteilte Web-Bulletin-Board setzt eine Thresholdsignatur Verfahren voraus. Die aktuelle Arbeit enthält eine Implementation des Thresholdsignatur-Verfahren wie es von Victor Shoup in Practical Threshold Signatures beschrieben wurde.

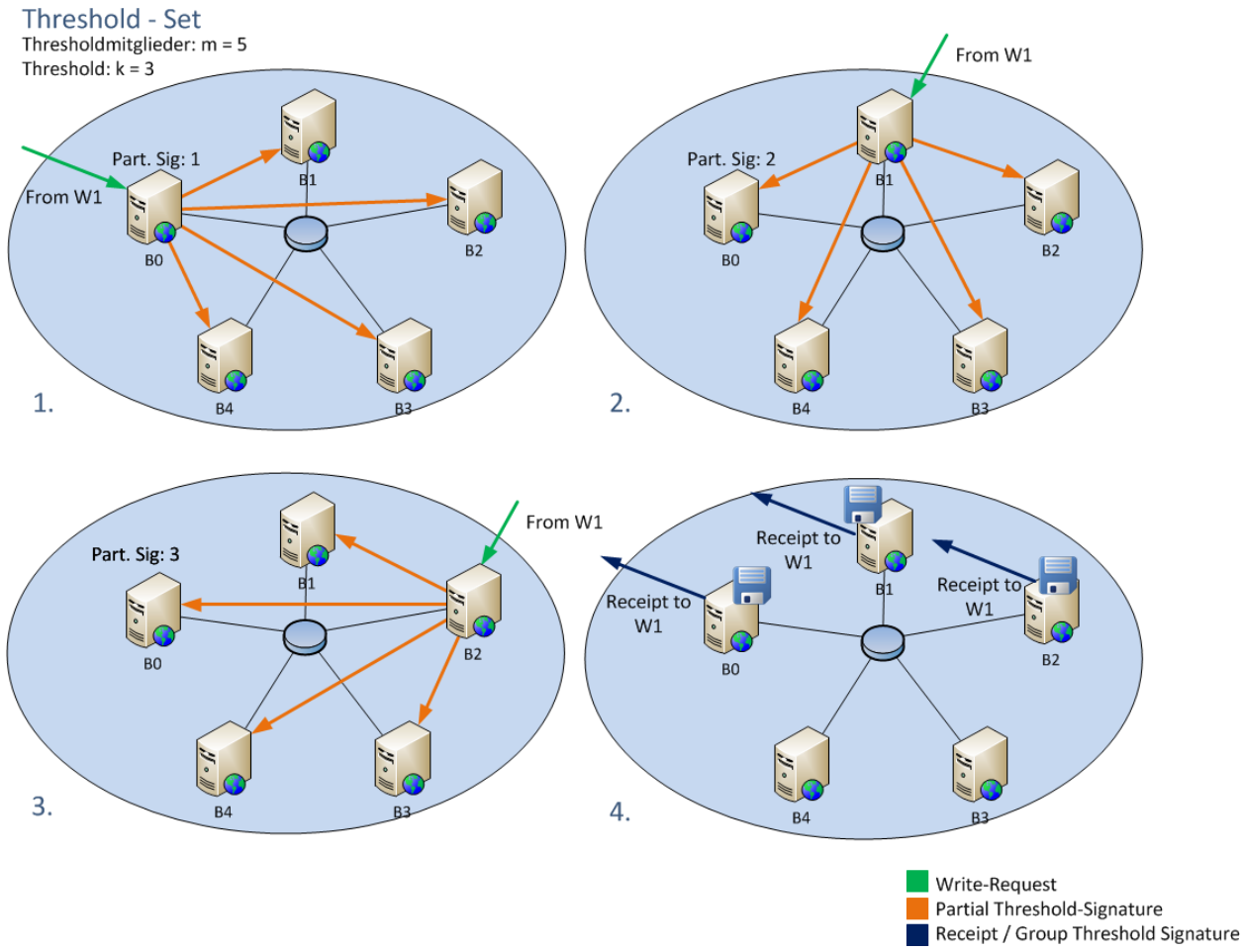
Dieses erlaubt es eine gültige verteilte RSA Signatur zu erstellen, wenn genügend Parteien eine partielle Signatur ausgestellt haben. Diese Teilsignatur kann von jeder Partei mittels Zero-Knowledge-Proof überprüft werden. Dies ist erforderlich, um die einzelnen Teilsignaturen zu prüfen, bevor die verteilte Threshold-Signatur erstellt wird.

Die erstellte Threshold-Signatur beweist, dass mindestens eine Anzahl Teilnehmer grösser oder gleich dem Threshold an der Generierung der Threshold-Signatur teilgenommen hat. Die Validierung der verteilten Threshold-Signatur geschieht danach analog der Validierung einer normalen RSA-Signatur, wobei der entsprechende RSA PublicKey verwendet wird.

> *„A k out of l threshold signature scheme is a protocol that allows any subset of k players out of l to generate a signature, but that disallows the creation of a valid signature if fewer than k players participate in the protocol. This non-forgeability property should hold even if some subset of less than k players are corrupted and work together. For a threshold scheme to be useful when some players are corrupted, it should should also be robust, meaning that corrupted players should not be able to prevent uncorrupted players from generating signatures.“*
>
> *Victor Shoup - Practical Threshold Signatures*

Für genauere Infos sei hier auf das Dokument „Practical Threshold Signatures“ von Victor Shoup verwiesen.

2.2.1 Anwendung aufs Web-Bulletin Board



In der obigen Illustration will ein Writer W1 eine Nachricht innerhalb des angehörigen Threshold-Sets publizieren. Das Threshold-Set besteht aus 5 Web-Bulletin Boards (B0, B1, B2, B3 & B4) wobei der Threshold bei drei im Voraus definiert wurde. Empfohlen ist hier als Threshold mindestens eine Zahl grösser der Hälfte des Threshold-Sets zu wählen. Nach dem Empfang des Write-Request's verteilt das Board jeweils die dazugehörige partielle Threshold-Signatur innerhalb des Threshold-Sets an die anderen Boards. (Abb. 1-3)

Nach Erreichen des Threshold in Abbildung 3, ist das WebBulletin-Board B0 in der Lage, mit der eigenen und den 2 erhaltenen partiellen Threshold-Signaturen (B1 & B2) dem Writer die verteilte Threshold-Signatur für dessen Nachricht als Quittung auszustellen (Abb. 4). Die Nachricht kann bei Erreichen des Threshold zudem auf der History abgespeichert werden.

In dieser Illustration könnten jetzt zum Beispiel die Boards B3 und B4 korrupt sein und nicht an der verteilten Threshold-Signatur-Generierung teilnehmen. Sogar ein Verteilen einer falschen partiellen Threshold-Signatur an die anderen Boards könnte vom jeweiligen Empfänger mittels Zero-Knowledge-Proof detektiert werden

3 Umsetzung

3.1 Vorausgehende Überlegungen

Im Zuge dieser Studienarbeit sollten die theoretischen Konzeptvorschläge von Roland Krummenacher in die Praxis umgesetzt werden. Es wurde sich darauf geeignet die ganze Arbeit in der Programmiersprache Java durchzuführen, da dies die Vorgabe OpenSource – Komponenten zu verwenden gut abdeckt. Nebst der eigentlichen Implementation der verschiedenen Kernkomponenten einer robusten und verteilten Web-Bulletin Umgebung ging es auch darum die Reaktionen des Systems bei verschiedenen Betriebsszenarien zu überprüfen. Dabei entschieden wir uns für die Realisierung einer Simulation, in welcher wir unsere Implementation des Web-Bulletin-Board sehr flexibel und unter verschiedensten Bedingungen simulieren und austesten können. Ein späteres Deployment der Komponenten auf einzelne Hardwarelösungen soll mit möglichst geringem Zeitaufwand und einer geringen Anpassung der Laufzeitumgebung geschehen.

3.2 Web-Bulletin-Board

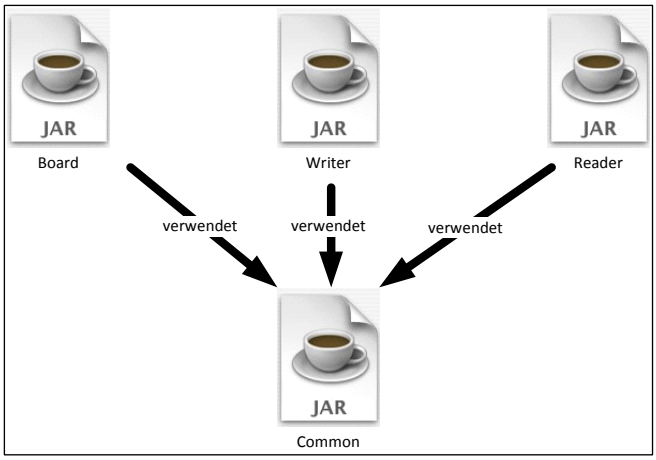
Eine vollständige Web-Bulletin-Board Umgebung wird durch den Einbezug der drei Hauptkomponenten Board, Writer und Reader definiert.

Dabei übernimmt das ‚Web Bulletin Board' die zentrale Rolle der Speicherung und Verwaltung der publizierten Nachrichten innerhalb einer WBB-Umgebung. Die eigentlichen Nachrichten werden dazu auf einer History abgespeichert, wovon jedes Board mehrere besitzt. Zu beachten ist, dass eine Nachricht von einem Writer nur dann auf eine History publiziert wird, wenn die Nachricht auch von der ‚Mehrheit' aller anderen Boards in der Umgebung auf ihre Korrektheit bezüglich Hash und Signatur geprüft und publiziert werden würde. Dieser Schwellwert wird Threshold genannt. Mittels einer verteilten Threshold-Signatur wird dem Writer eine Quittung ausgestellt, dass seine Nachricht korrekt auf mindestens einer Anzahl Boards grösser oder gleich dem Threshold unverändert publiziert wurde. Mittels dieser kann der Voter später jederzeit validieren, dass seine Stimme noch korrekt zum Wahlergebnis beiträgt.

Der ‚Writer' hat die Aufgabe eine Schnittstelle nach aussen zur Verfügung zu stellen, welche es ermöglicht eine neue Nachricht innerhalb einer WBB-Umgebung zu publizieren. Dabei wird die Nachricht von einem Writer redundant auf alle zur Verfügung stehenden Boards innerhalb der Umgebung publiziert.

Der ‚Reader' als Prüfkomponente hat die Aufgabe die Boards innerhalb der WBB-Umgebung darauf zu überprüfen, dass keine Nachricht im Nachhinein abgeändert, dazwischen gefügt oder gelöscht wurde.

Bei der Implementation der eigentlichen Komponenten einer Web-Bulletin-Board Umgebung soll darauf geachtet werden, dass die einzelnen Komponenten einzeln als Java Libraries in Form von JAR's (Java Archives) verwendet werden können. Dies ermöglicht eine vielseitige Benutzung des Web-Bulletin-Boards für verschiedenste Anwendungszwecke – sei dies nun für eine E-Voting Anwendung oder für eine andere Verwendung. Im Rahmen dieser Arbeit soll die Library in erster Linie in die Simulation eingebunden werden und später in einem zweiten Schritt jeweils die einzelnen Komponenten auf separaten Servern verteilt und betrieben werden. Es wird darauf geachtet, dass nebst den drei JAR-Dateien für die Hauptkomponenten noch ein JAR-File ‚Common' entwickelt wird, in welches jene Teilkomponenten eingezogen werden, welche von allen drei Komponenten benötigt werden. Darunter zählen vor allem Komponenten für die Kommunikation sowie für die Kryptographie.

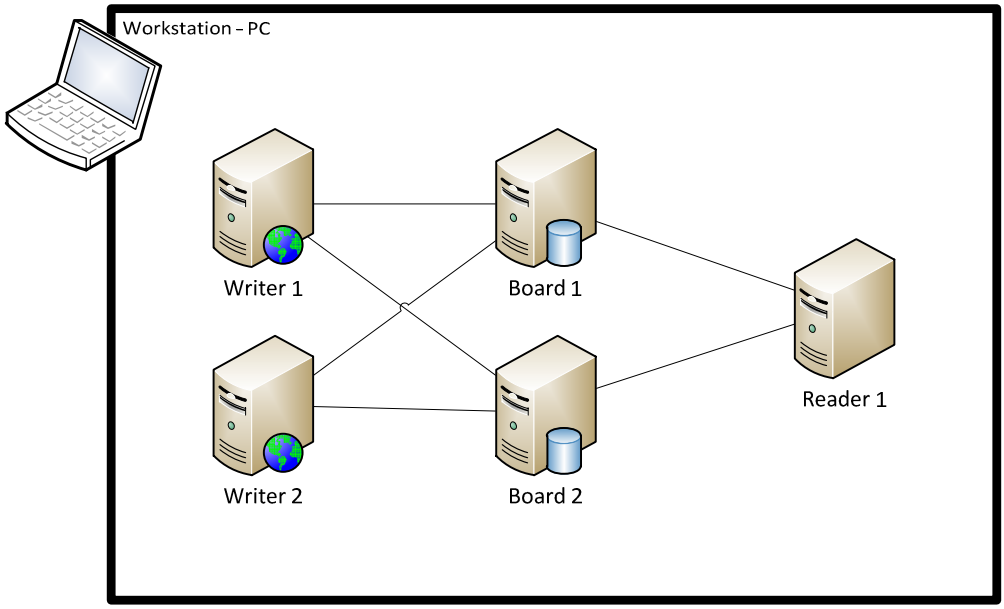


4 Deployment

Im Folgenden sind die möglichen Deployments für das WBB beschrieben. Dabei sind auch Deployments beschrieben, welche noch nicht unterstützt werden.

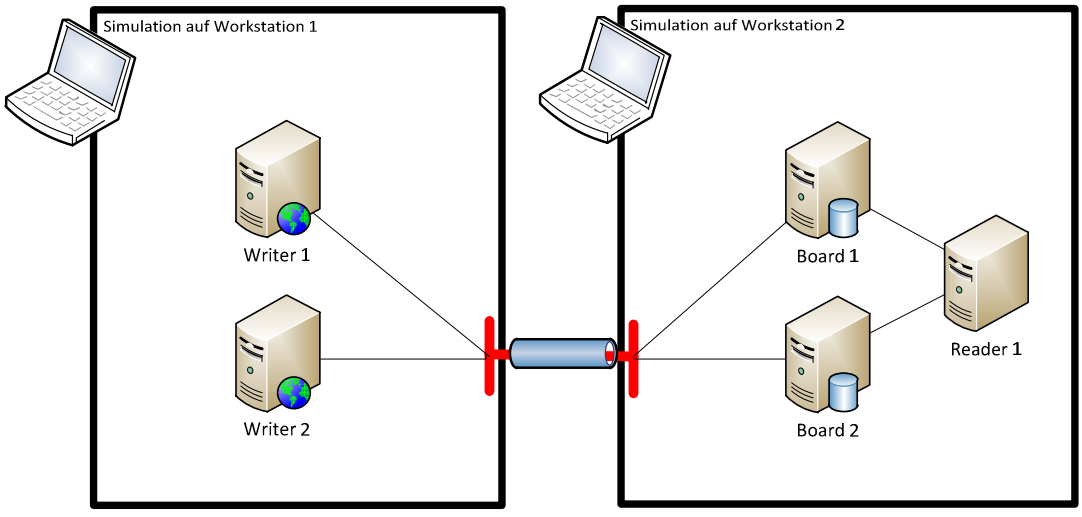
4.1 Single-Machine-Simulation

Die Simulation läuft auf einer einzelnen Maschine. Es findet also keine Netzwerkkommunikation zwischen den einzelnen Komponenten statt. Im Rahmen dieser Studienarbeit konzentrierten wir uns auf diese Deploymentart, welche uns eine gute Plattform bietet, die Funktionalität der WBB-Umgebung zu testen.



4.2 Multi-Machine-Simulation

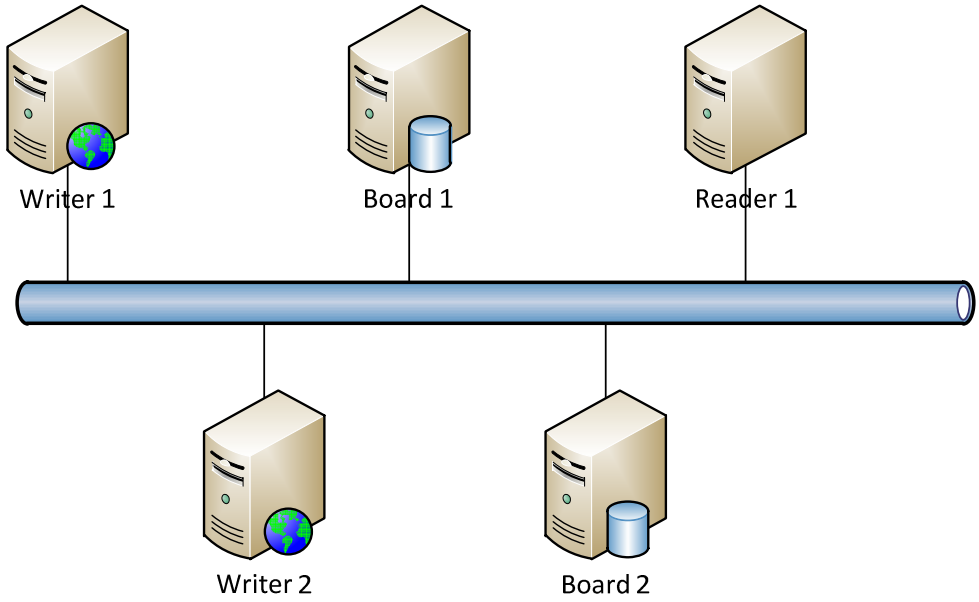
Auf mehreren Maschinen läuft ein Teil der Simulation. Das heisst es laufen auf den unterschiedlichen Maschinen unterschiedliche Komponenten. Diese Komponenten kommunizieren über das Netzwerk untereinander. Hauptvorteil dieser Art des Deployments ist es, die Vorzüge der Simulation auszunutzen und zugleich die Skalierbarkeit des Systems nicht zu vernachlässigen. Dieses Deployment wurde im Rahmen der Arbeit nicht implementiert.



4.3 Single-Component-Machine

Dieses Deployment ist für das Endprodukt gedacht. Dabei läuft eine einzelne Komponente auf einer eigenen Maschine. Die Funktionalität aus der Simulation ist nicht Bestandteil des Deployments.

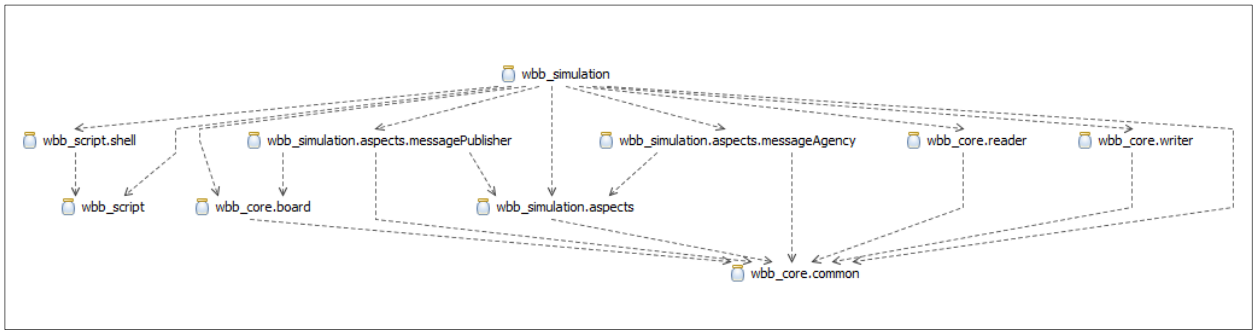
Dieses Deployment wurde im Rahmen der Arbeit nicht implementiert.



5 Architektur

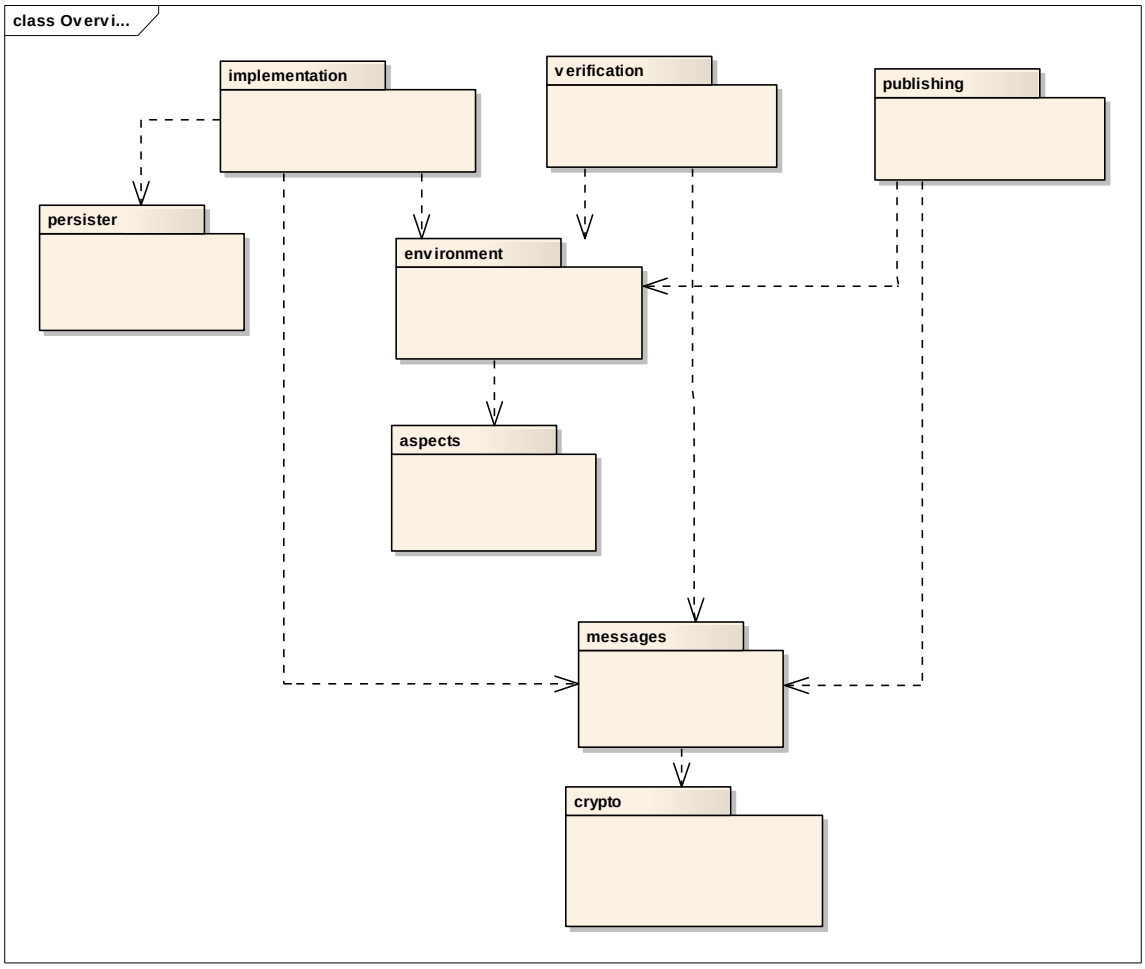
5.1 JAR-Abhängigkeiten

Das nachfolgende Diagramm zeigt die Abhängigkeiten zwischen den erstellten Jar-Dateien. Die Abhängigkeiten zeigen auf welche anderen Jar-Dateien benötigt werden, um die einzelne Jar-Datei zu verwenden. Dabei wurde darauf geachtet, dass die Kernkomponenten möglichst unabhängig von anderen Dateien sind. Somit können diese Komponenten später einfach wiederverwendet werden, ohne Abhängigkeiten auf die Simulationsumgebung zu haben.



5.2 Package-Overview

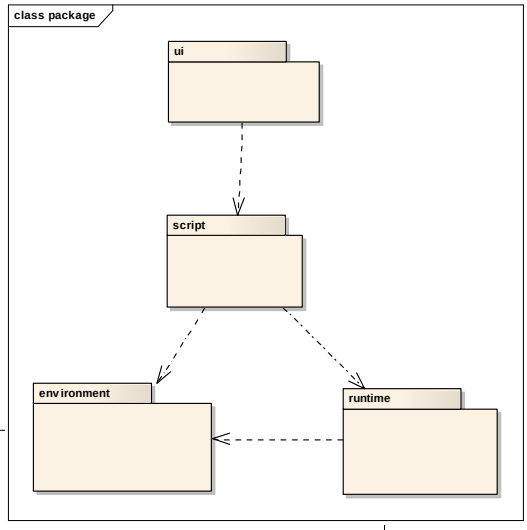
Dieser Abschnitt beschreibt die logische Sicht der Architektur. Dabei wird auf die zentralen Packages und ihre Verwendung eingegangen.

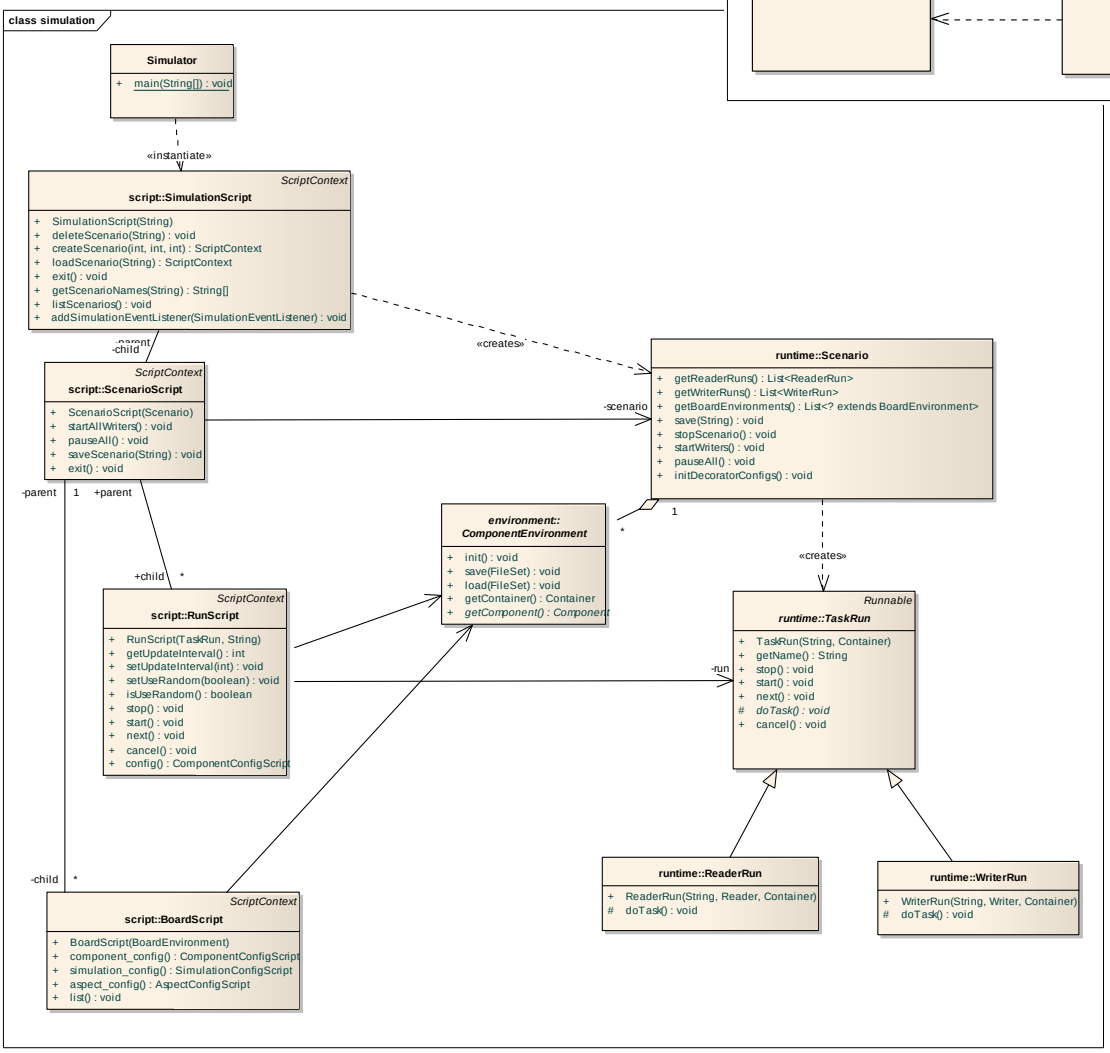


5.3 Packagebeschreibungen

5.3.1 simulation

Im Package ‚simulation' befinden sich alle für die Simulation notwendigen Klassen. Dies erstreckt sich von der grafischen Aus- und Eingabe, über die Skriptsprache zur Steuerung der Komponenten bis hin zur eigentlichen Simulationslaufzeitumgebung für die Komponenten. Die Eingaben aus dem UI werden an die Klassen im Package ‚script' weitergeleitet und es ist möglich, Konfigurationen welche sich im Package ‚environment' befinden, zu bearbeiten. Dieses wiederum kapselt vor allem die Funktionalität aus dem Package ‚runtime'. Das Package ‚runtime' beinhaltet die zentralen Klassen um Szenarien zu laden und in einer Simulation laufen zu lassen.

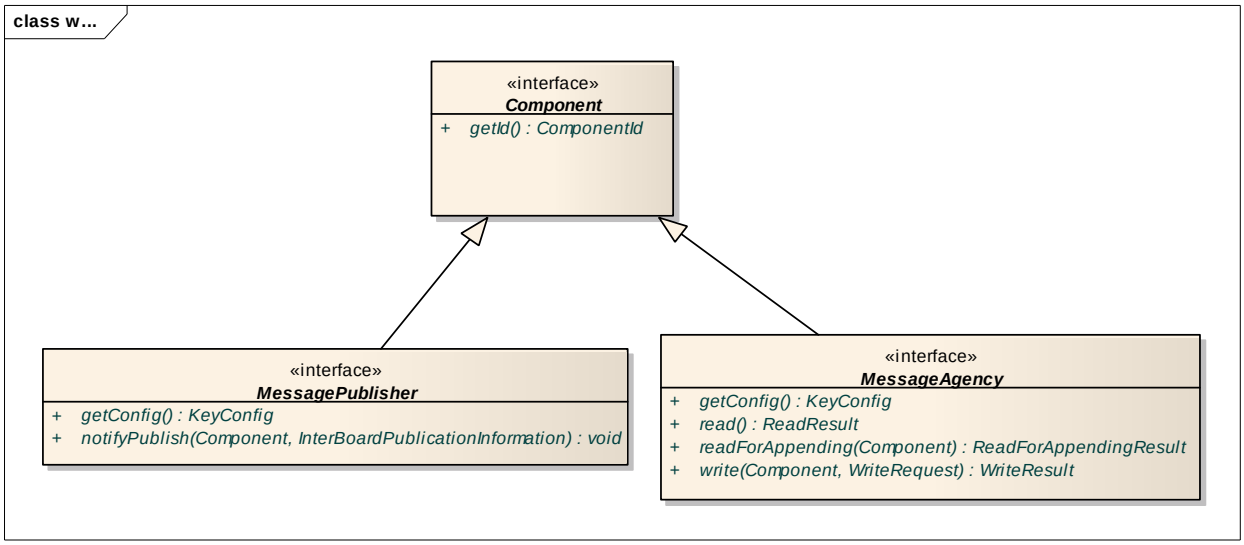




Wie das Diagramm zeigt, läuft innerhalb einer Simulation jeweils ein Szenario. Das Szenario beschreibt eine Simulationsanordnung. Dazu gehören die einzelnen Komponenten wie Board, Writer und Reader. Zusätzlich sind auch die Konfigurationen dieser Komponenten darin enthalten. Konkrete Implementationen der TaskRun Schnittstelle werden ebenfalls im Scenario definiert und von diesem beim Laden auch kreiert. Die abstrakte TaskRun Klassen erlaubt es eine Operation zyklisch aufzurufen. Dabei gibt es eine Implementation für den Reader, welche zyklisch die Boards prüft und eine Implementation für den Writer welche zufallsgenerierte Meldungen auf die Boards schreibt.

5.3.2 wbb

‚wbb' ist das oberste Package und beinhaltet die grundlegen Schnittstellen. Die Schnittstelle Component definiert eine Komponent aus einer WBB-Umgebung. Board, Writer und Reader implementieren diese Schnittstelle. Eine Ausprägung der Komponente ist der MessagePublisher und die MessageAgency. Der MessagePublisher definiert eine Schnittstelle für eine Komponente welche Nachrichten publizieren kann und von anderen Komponenten informiert wird, wenn diese eine Nachricht publizieren wollen. Konkret wird diese Schnittstelle vom DistributedBoard aus dem Package ‚implementation' implementiert. Eine andere Spezialisierung einer Komponente ist die MessageAgency. Die MessageAgency erlaubt es Nachrichten zu publizieren und bereits publizierte Nachrichten abzurufen. Das DistributedBoard implementiert dieses Interface ebenfalls.



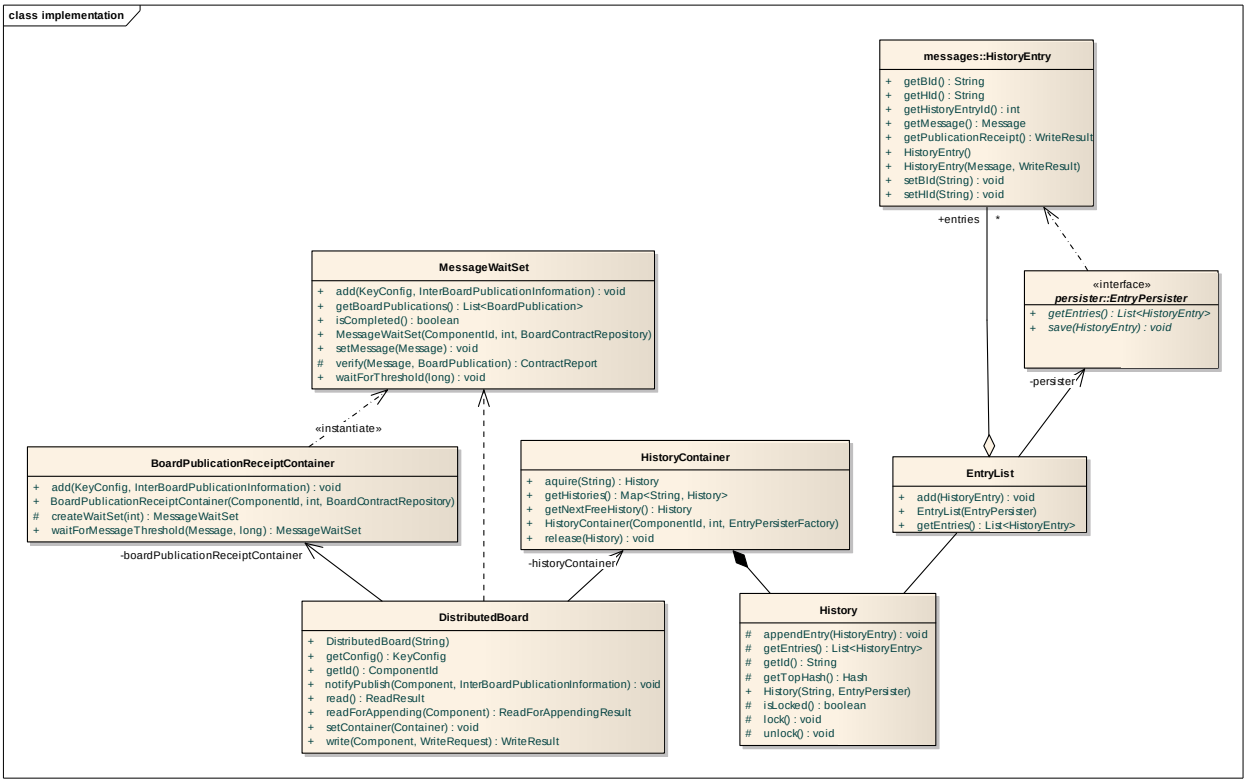
5.3.3 implementation

Das Package ‚implementation' enthält die konkrete Implementation des WebBulletin Boards. Dabei implementiert die DistributedBoard Klasse das verteilte Board, wie dies im Kapitel der Umsetzung beschrieben ist.

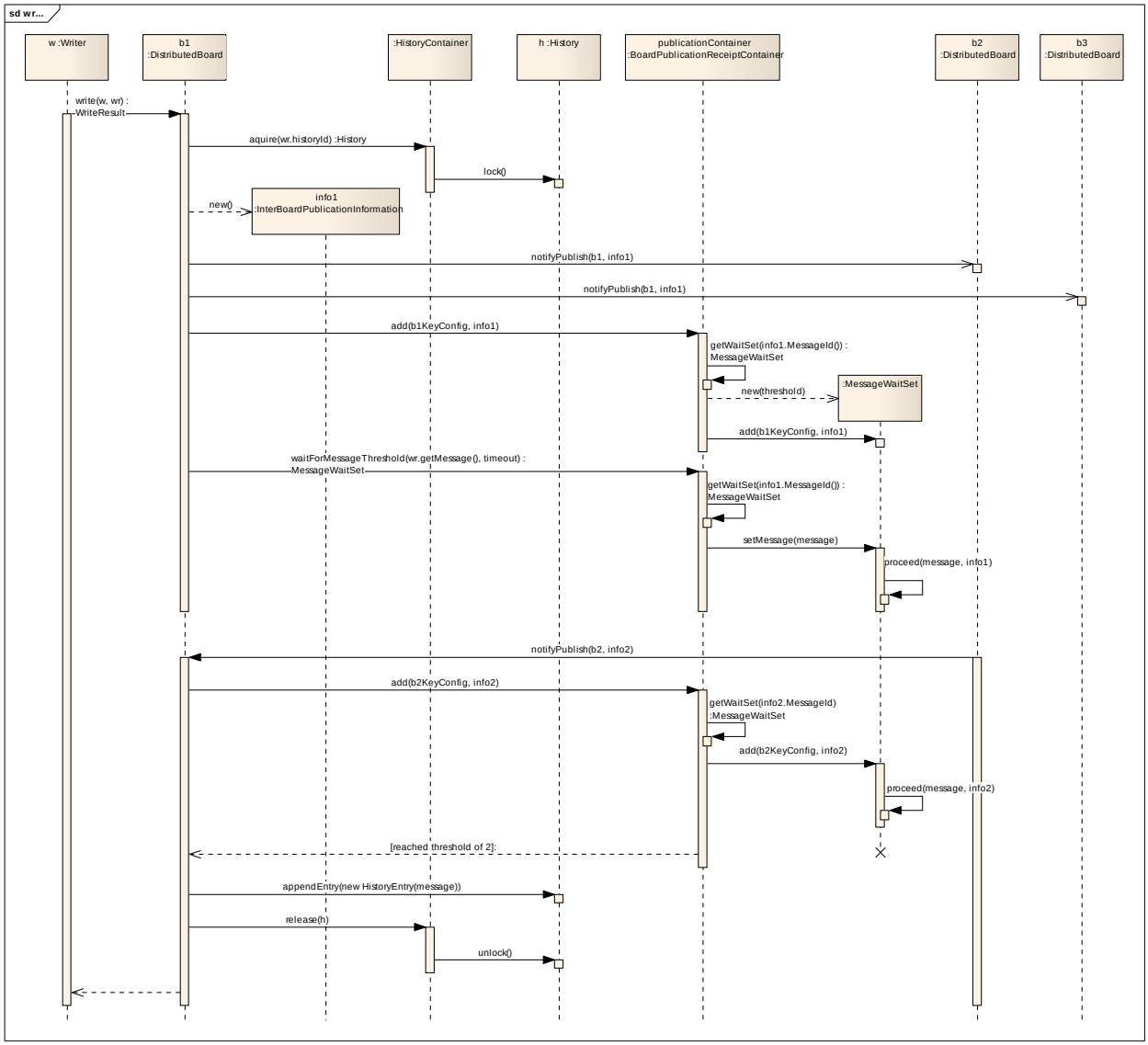
Das Board dient als zentrale Komponente innerhalb der WBB – Umgebung. Sie dient als Datencontainer für die Nachrichten, welche innerhalb der WBB – Umgebung gespeichert werden sollen. Dazu verfügt jedes Board über mehrere Histories auf denen die einzelnen Nachrichten in Historyentries abgespeichert werden. Die einzelnen Histories werden vom HistoryContainer verwaltet, wobei dieser entscheidet auf welche History eine neue Nachricht gespeichert werden soll.

Eine neue Nachricht darf zudem erst auf einer History publiziert werden, wenn diese mindestens von einer Anzahl Boards grösser oder gleich dem Threshold innerhalb des Threshold-Sets ebenfalls publiziert wird. Dazu dienen die Teilkomponenten BoardPublicationReceiptContainer und das MessageWaitSet innerhalb der Board-Komponente. Nach dem Eingang einer neuen Nachricht welche auf dem Board publiziert werden soll, stellt das Board eine 'InterBoardPublicationInformation' aus und sendet diese an alle anderen Boards innerhalb des gleichen Threshold-Set. Der BoardPublicationReceiptContainer wiederum empfängt eben solche InterBoardPublicationInformationen von den anderen Boards und fügt diese dem dazugehörigen MessageWaitSet hinzu. Ein MessageWaitSet gehört einer Message-ID an und beinhaltet somit alle InterBoardPublicationInformation die bereits von anderen Boards empfangen wurden und aussagen, dass diese Boards diese Nachricht speichern würden. Sobald nun die Anzahl empfangener InterBoardPublicationInformationen den Threshold überschreitet, wird die entsprechende Nachricht auf die History gespeichert. Falls vorher ein zuvor definiertes Timeout überschritten wurde, wird die Nachricht verworfen.

Sämtliche ausgetauschten Nachrichten sind natürlich signiert und gehasht, die InterBoardPublicationInformationen sogar mit der partiellen Threshold Signatur versehen. Dabei sei auf die Kapitel verwiesen welche auf diese Thematik genauer eingehen.



Sequenzdiagramm

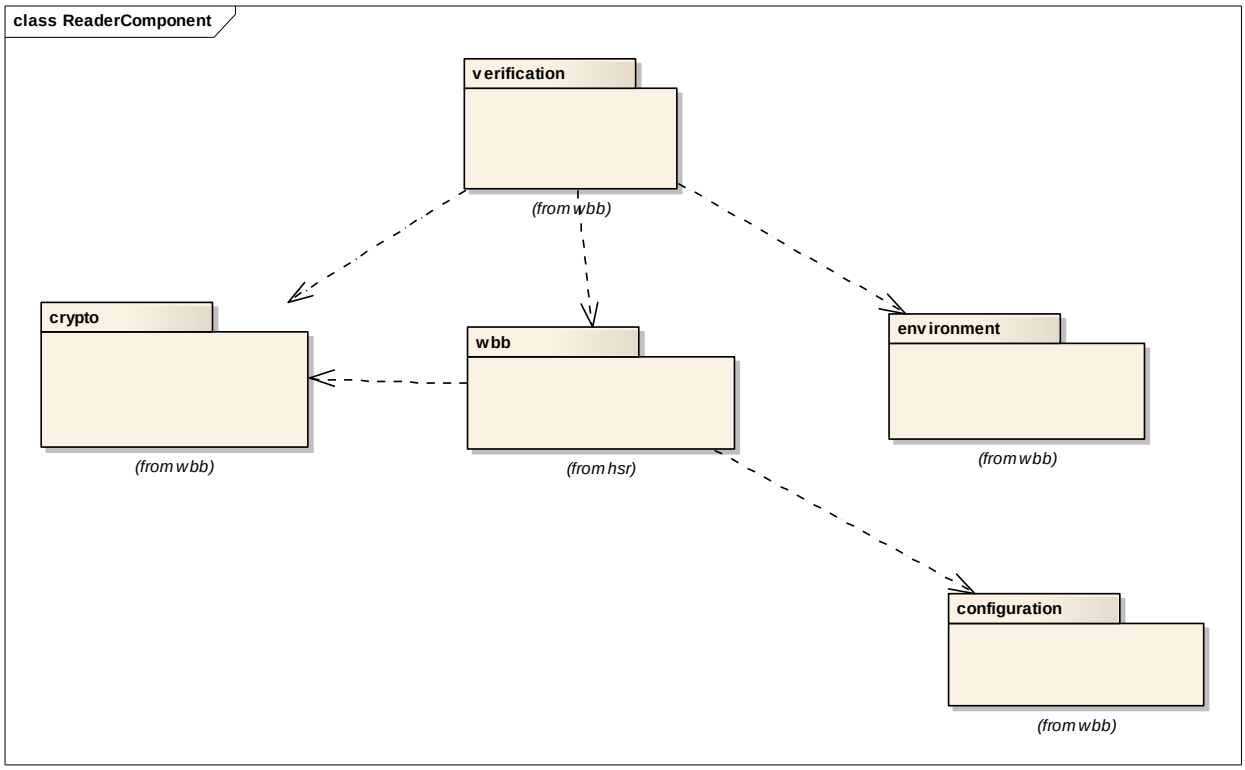


Ablauf einer Publikation

5.3.4 verification

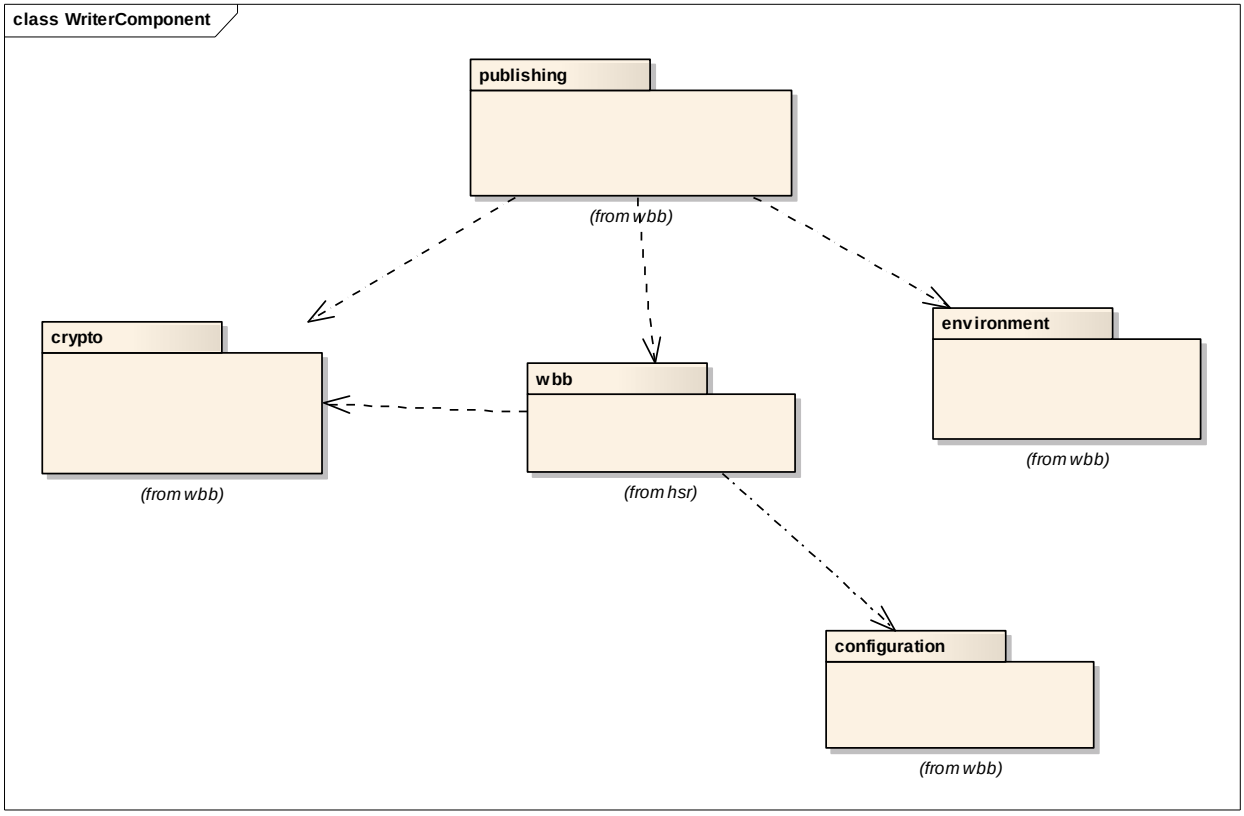
Das Package ‚verification' enthält als zentrale Klasse den Reader. Der Reader erlaubt es die Histories eines oder mehrerer Boards zu prüfen. Dabei prüft er die einzelnen Einträge in der History von hinten nach vorne. Es wird validiert, ob der Hash und die Signatur korrekt sind.

Das Package kann als eigenständige Komponente betrieben werden. Dabei benötigt es, wie im Diagramm ersichtlich, die Schnittstellen aus dem wbb Package und die Kryptofunktionen aus dem crypto package. Das environment package dient dazu die Laufzeitumgebung zu konfigurieren.



5.3.5 publishing

Dieses Package beinhaltet als zentrale Klasse den Write. Der Writer erlaubt es eine Nachricht an die vorhandenen Boards innerhalb des Threshold-Sets zu senden und zu prüfen, ob der Inhalt auch publiziert wurde. Dabei versendet er parallel an alle Boards die gleiche Meldung. Wenn er von mindestens einem Board eine Bestätigung, respektive eine gültige verteilte Threshold-Signatur erhält, dann ist die Meldung sicherlich auf genügend Boards publiziert worden. Erhält er jedoch von keinem Board eine Bestätigung, so versucht er es erneut die Meldung zu versenden.

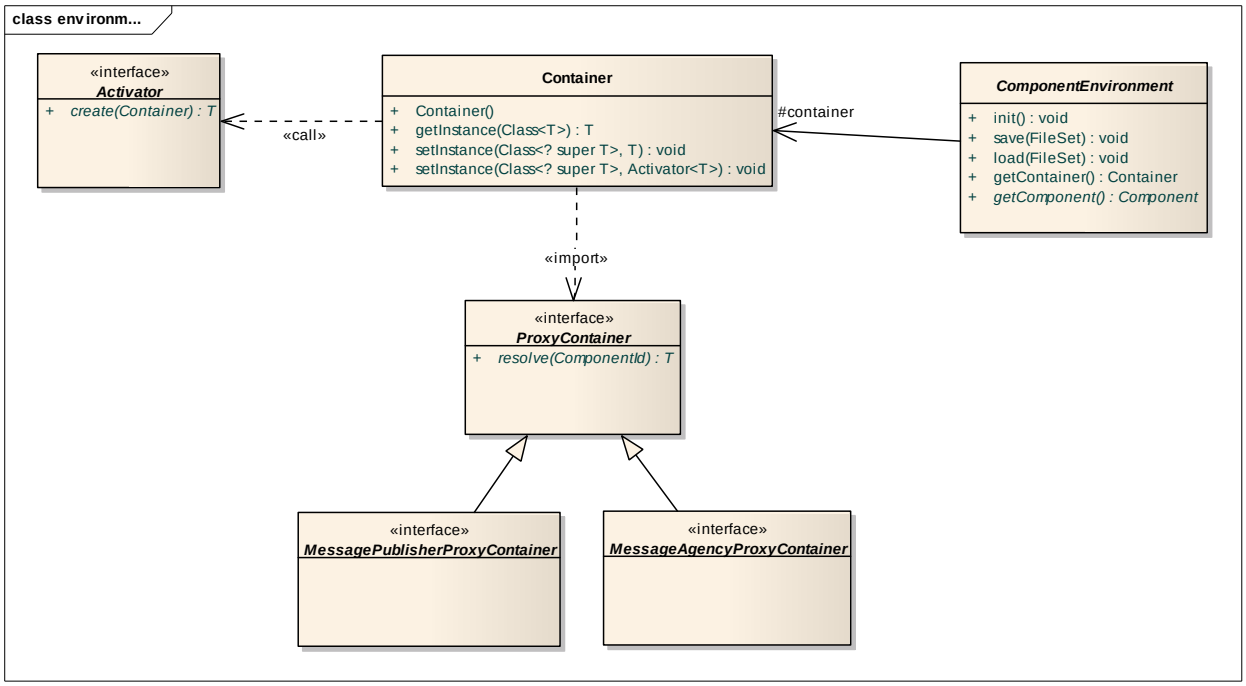


5.3.6 environment

Das environment package definiert und konfiguriert die Laufzeitumgebung der einzelnen Komponenten. Dabei fungiert die Klasse ‚Container' als das zentrale Element. Es enthält die für die Laufzeit erforderlichen Instanzen. Die einzelnen Klassen verwenden den Container um an die spezifischen Instanzen zu gelangen. Die Instanzen können zur Startzeit der Applikation auf dem Container gesetzt werden oder dynamisch über den Activator nachträglich beladen werden. Der Activator wird vom Container aufgerufen sobald eine Instanz eines spezifischen Typen nachgefragt wird. Es ist dann die Aufgabe des Activators die Instanz zu kreieren und sie an den Container zurückzugeben.

Die einzelnen Klassen der Applikation sind so aufgebaut, das sie den Container verwenden um an die konkrete Instanz der geforderten Schnittstelle zu gelangen. Damit hat man die Möglichkeit während dem Start der Applikation den Container zu konfigurieren und somit das Laufzeitverhalten zu steuern.

Für die Initialisierung des Containers ist das ComponentEnvironment zuständig. Konkrete Environments für die drei in der WBB-Simulation vorfindbaren Komponenten sind im Package ‚simulation' untergebracht.

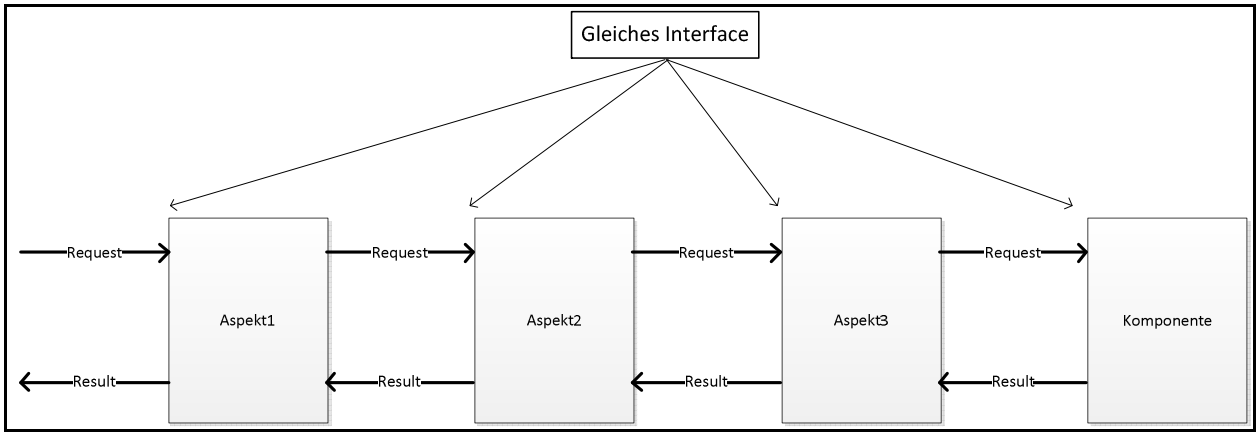


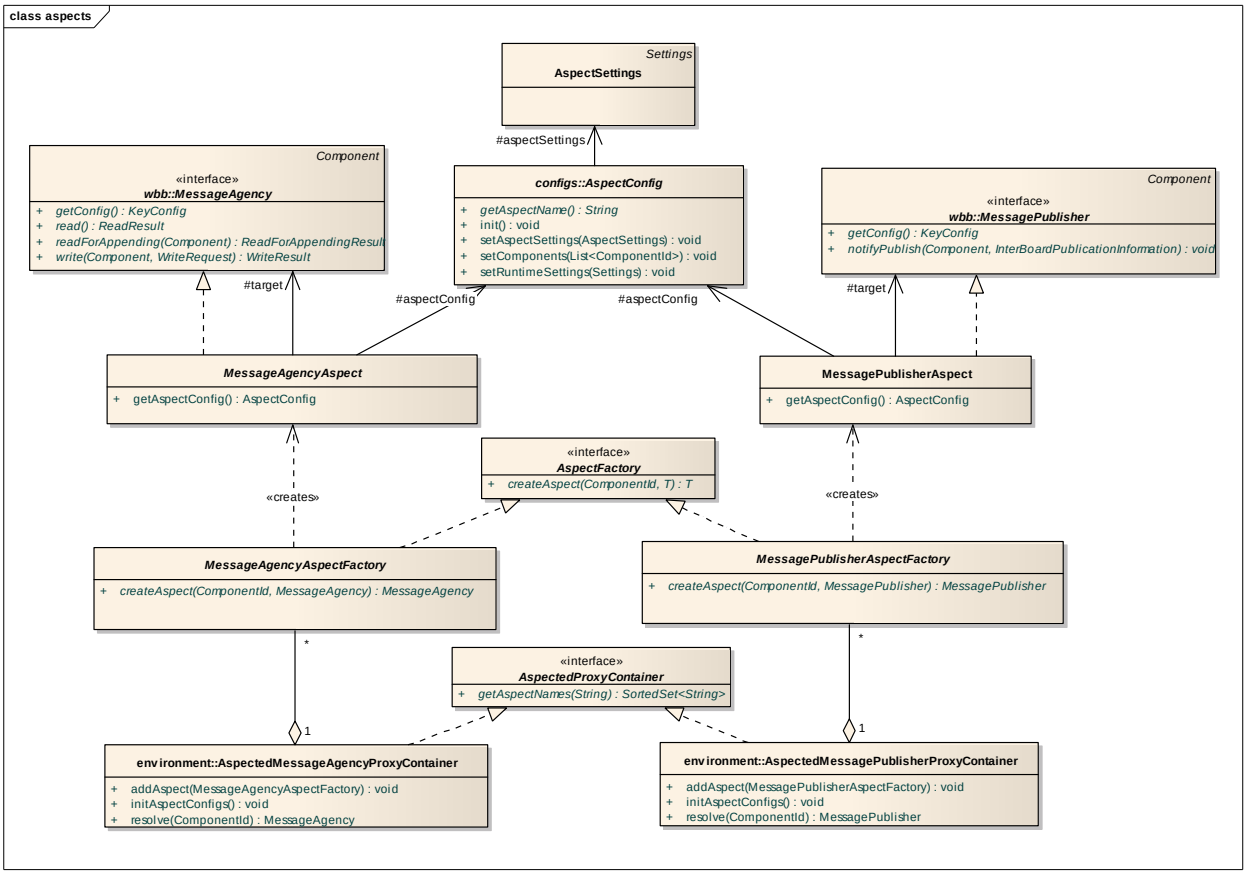
5.3.7 aspects

Das Package ‚aspects' stellt die Funktionalität zur Verfügung, einer gegebenen Komponente Aspekte einzuspielen, sodass sich diese anders verhält, ohne dass sie selbst etwas davon mitbekommt. Dies ist nötig um die verschiedenen Szenarien innerhalb der Simulation zu simulieren und auszutesten. Die folgenden Aspekte wurden bereits implementiert und können je nach Belieben auf einer gewünschten Komponente aktiviert und konfiguriert werden:

Aspekt	Beschreibung
NetworkDelayAspect	Jeder Methodenaufruf wird vor und nach dem Aufruf mit einem konfigurierbaren Delay versehen, was innerhalb der Simulation den Netzwerkdelay simulieren soll, welcher ein Methodenaufruf über einen Webservice ebenfalls erfahren würde. Anwendbar bei • MessageAgency • MessagePublisher
StopwatchAspect	Wenn aktiviert, wird bei allen Methodenaufrufen die Zeit gemessen und im zentralen Log abgespeichert. Anwendbar bei • MessageAgency • MessagePublisher
WrongHashAspect	Wenn aktiviert wird der Hash innerhalb des Methodenaufruf ‚write()' manipuliert, sodass dieser nicht mehr dem richtigen Hashwert entspricht. Anwendbar bei • MessageAgency
FinalThresholdSignatureAspect	Wenn aktiviert, wird bei einem WriteResult welches auf ein WriteRequest von einem Board zum Writer zurückgesendet wird, die verteile Thresholdsignatur manipuliert. Anwendbar bei • MessageAgency
PartialThresholdSignatureAspect	Wenn aktiviert, wird bei einem InterBoardPublicationInformation – Paket welches zwischen zwei Boards ausgetauscht wird, die partielle Thresholdsignatur manipuliert. Anwendbar bei • MessagePublisher
NoWriteAfterReadAspect	Wenn aktiviert, wird der Aufruf der Methode ‚write()' auf der MessageAgency unterbunden. Anwendbar bei • MessageAgency

Die zentrale Komponente bei den aspects ist der AspectedMessage(Agency/Publisher)ProxyContainer. Dieser kapselt einen richtigen Proxycontainer seines Typs und fügt bei jeder aufgelösten Komponente innerhalb der WBB-Umgebung die bei ihm registrierten Aspekte hinzu. Die Aspekte dekorieren dabei die eigentlich aufgelöste Komponente, wobei sie dasselbe Interface implementieren und eine Referenz auf die Komponente haben. Dabei hat jeder Aspekt die Möglichkeit vor und nach jedem Methodenaufruf ins Geschehen einzugreifen. Zusätzlich besitzt jeder Aspekt eine ‚AspectConfig', welche die konfigurierten Werte für den jeweiligen Aspekt besitzt.



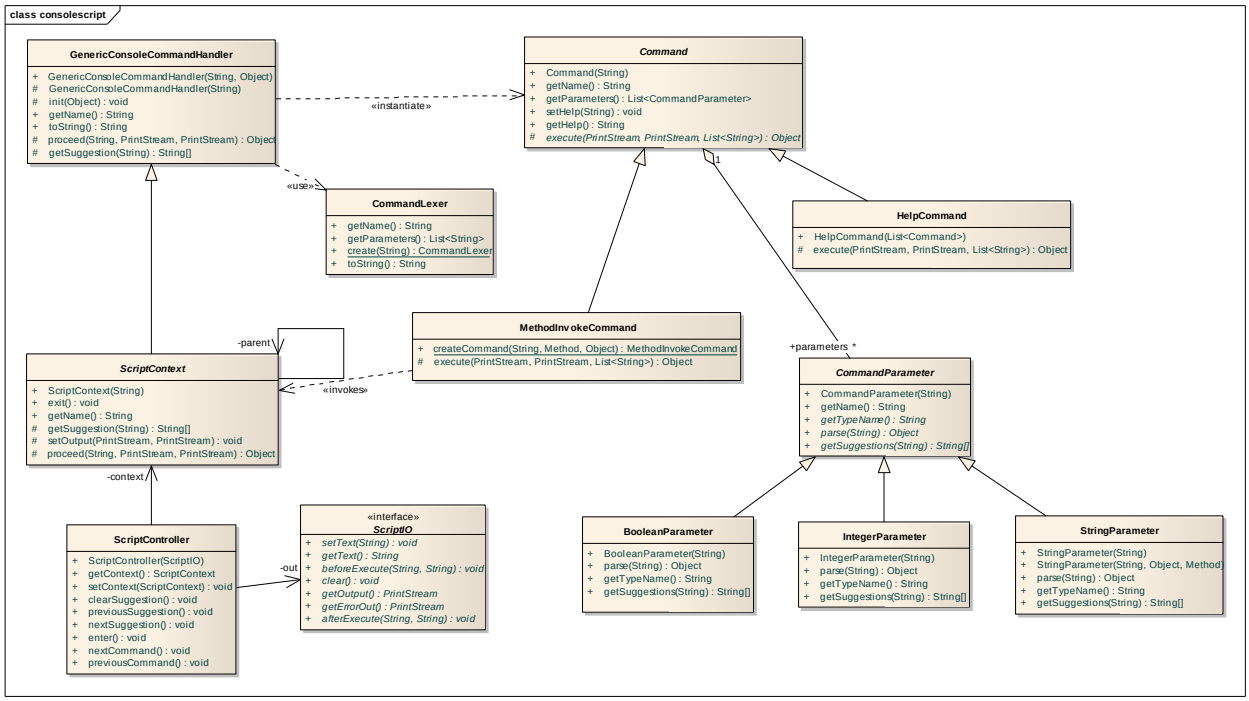


5.3.8 script

Mit Hilfe des Package ‚script' kann zur Laufzeit die Applikation gesteuert werden. Es ist eine Skiptsprache definiert, welche es erlaubt Java-Methoden dynamisch aufzurufen.

ScriptContext: Die ScriptContext Klasse erlaubt es die Skriptsprache durch Vererbung der Klasse ScriptContext zu erweitern und neuen Kommandos zu definieren.

ScriptController: Der ScriptController steuert die Standardeingabeverarbeitung für eine Script-Laufzeitumgebung. Dabei kann ein eigener ScriptIO implementiert werden, welcher dazu da ist die Benutzereingabe entgegenzunehmen und dem Benutzer auch die Ausgabe des Skripts weiterzuleiten.



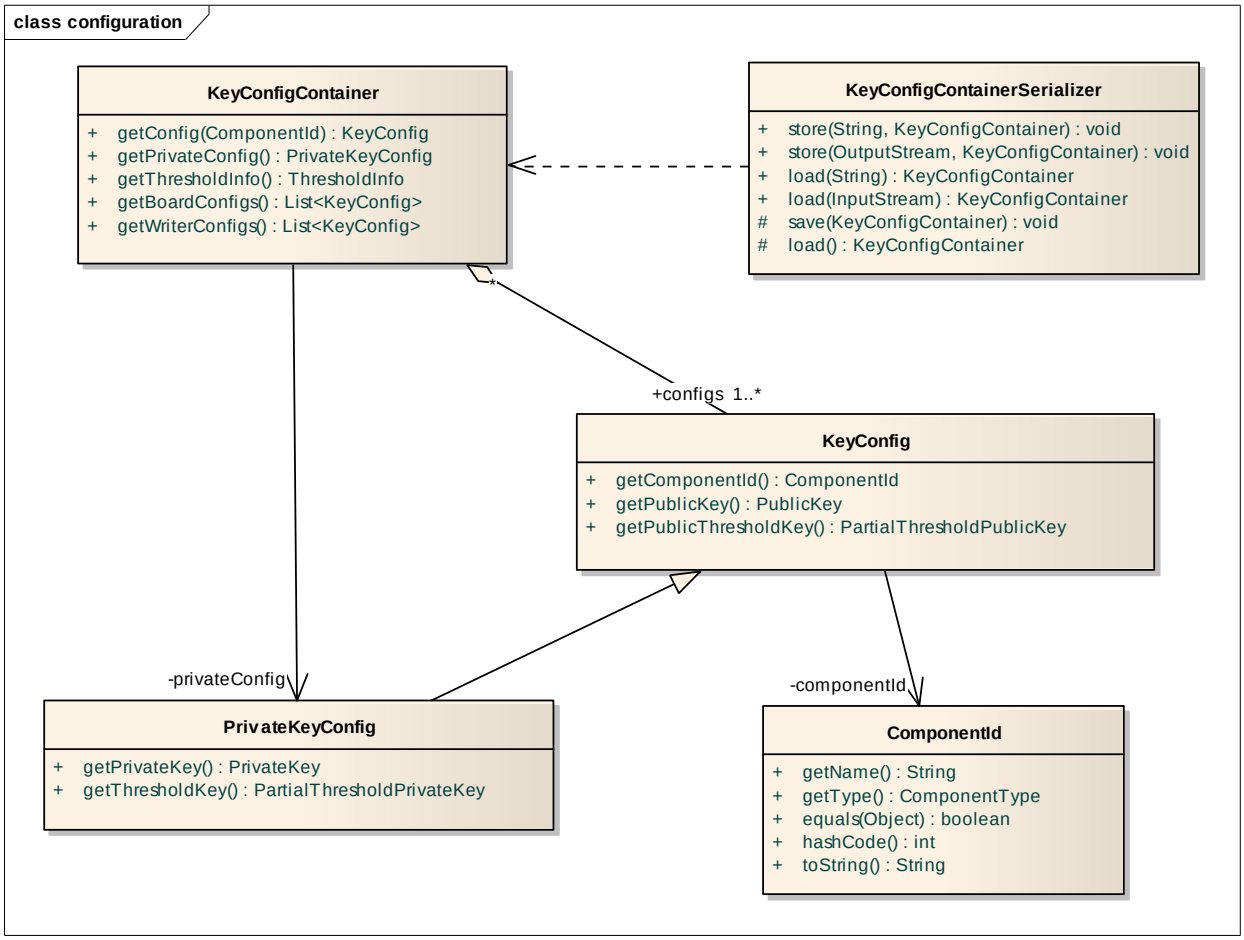
In der Applikation sind zwei Implementationen für den ScriptIO. Zum einen für die Darstellung in einem Swing UI und zum anderen für den Zugriff über ein SSH-Konsole.

5.3.9 messages

Alle Meldungen, die zwischen zwei Komponenten ausgetauscht werden können sind im messages Package definiert. Eine Meldung ist zum grössten Teil nur ein einfaches ValueObject, d.h. die Werte für die Felder können nur während dem Erstellen des Objekts gesetzt werden.

Ein weiteres Konzept, das umgesetzt wurde sind die Contracts. Die Contracts definieren die Beziehung zwischen zwei Komponenten und einer ausgetauschten Nachricht. Dabei unterstützen sie die Validierung und auch die Erfüllung des Vertrages. Für weiterführende Details sei an dieser Stelle auf die JavaDoc verwiesen.

5.3.10 configuration



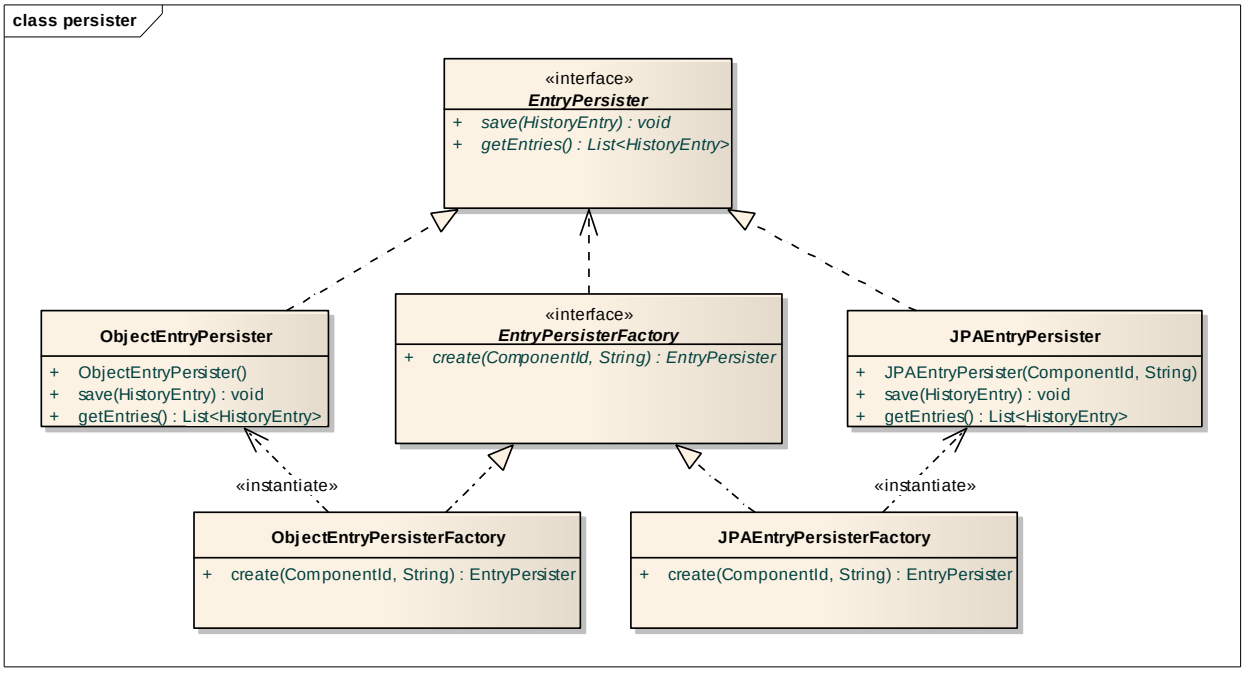
5.3.11 crypto

Im crypto Package sind die Implementationen für die benötigten kryptographischen Verfahren enthalten.

Zentral ist CryptoKontext Klasse, welche die Verfahren für die Applikation definiert. Dabei stellt er drei Arten von Providern zu Verfügung:

- Signature
  - Wrapper über die Java Signature Implementation.
- Hash
  - Wrapper über die Java MessageDigest Implementation.
- Threshold Multisignatur
  - Implementation nach Shoup (Siehe Kapitel Practical Threshold Signatures nach Shoup)

5.3.12 persister

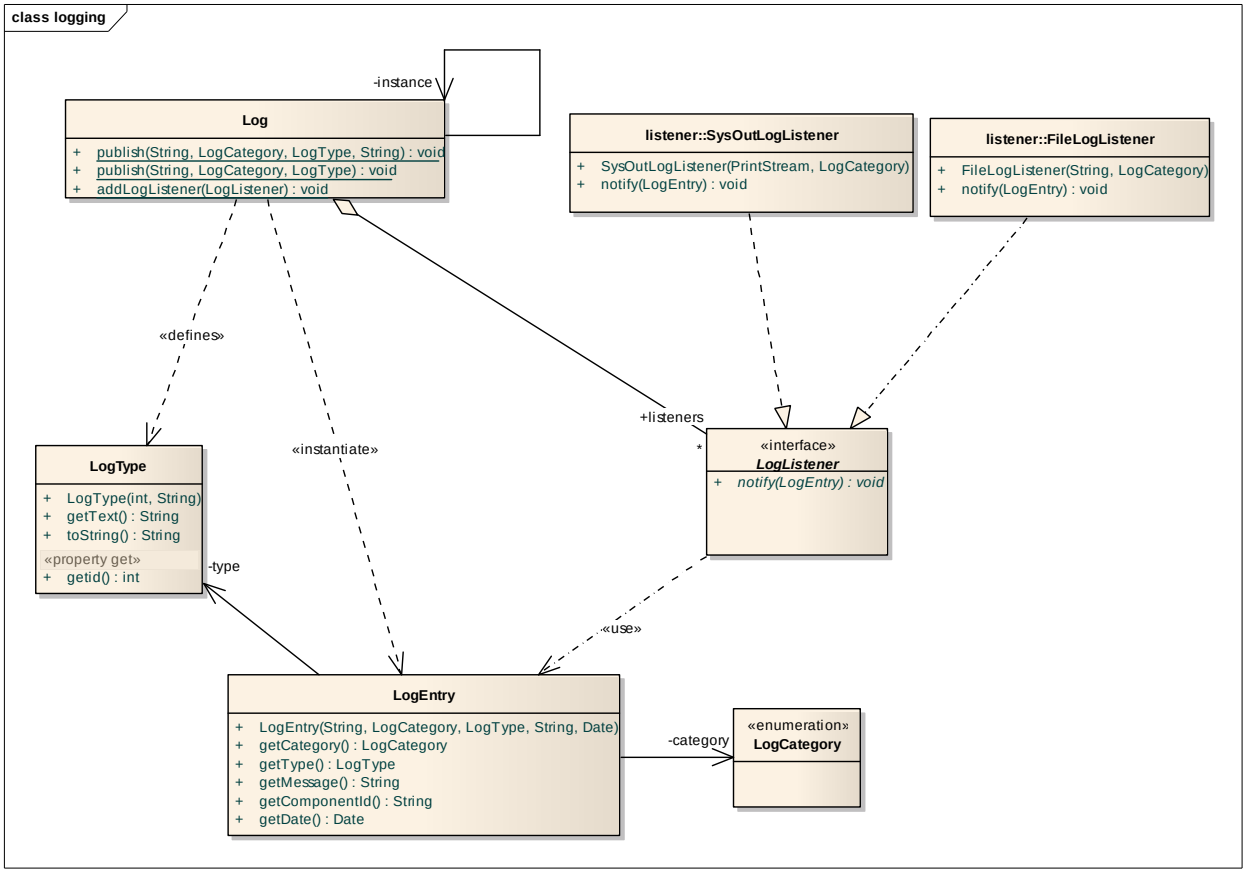


Das package ‚persister' beinhaltet die nötigen Schnittstellen und Implementationen, um einen Eintrag aus der Historie zu persistieren und diese wieder zu laden. Ein EntryPersister hat hiermit die Aufgabe, die Nachrichten zu speichern und die gespeicherten auch wieder zu laden. Über eine konkrete Implementation des EntryPersisterFactory Interfaces können die konkreten EntryPersister erstellt werden. Dabei sind zwei konkrete Implementation im Package vorhanden. Zum einen gibt es über den JPAEntryPersister die Möglichkeit die Einträge in eine Datenbank zu speichern und zum anderen über den ObjectEntryPersister die Einträge im Memory zu verwalten.

Das Package wird ausschliesslich von der Komponente ‚Board' verwendet.

5.3.13 logging

Für eine einfache Logging-Funktionalität stellt das logging Package die nötigen Klassen zur Verfügung. Über die Log-Klasse können Meldungen publiziert werden. Alle Lognachrichten werden in Info-, Warnungs- und Error-Meldungen unterteilt. Zudem wird vermerkt zu welchem Zeitpunkt welche Komponente geloggt hat. An der Log-Instanz können mehrere LogListeners angehängt werden, die über neue Meldungen benachrichtigt werden und diese dann weiterverarbeiten können. Zwei LogListeners sind bereits implementiert. Der FileLogListener erlaubt es, Meldungen einer bestimmten Kategorie in einen Datei zu schreiben. Ebenso kann auch der SysOutLogListener die Meldungen nach einer Kategorie filtern. Die Meldungen werden dann jedoch in einen gegebenen PrintStream geschrieben.



6 Testszenarien

6.1 Feststellen eines falschen Nachrichten-Hashes von einem Writer

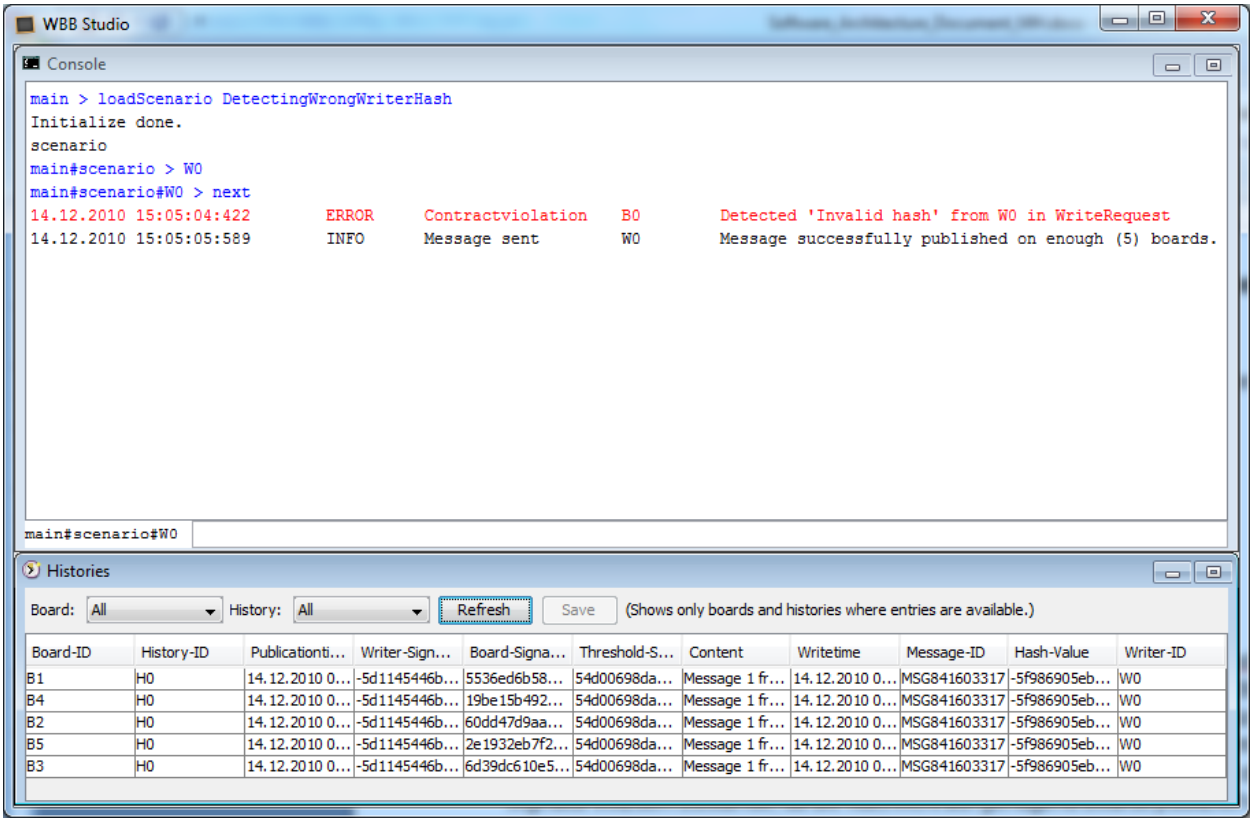
Beschreibung

Im folgenden Szenario wird festgehalten, wie ein Board feststellen kann, wenn ein Writer bei einer Publikationsanfrage einen fehlerhaften Hashwert mitgibt.

- **Szenarioname:** DetectingWrongWriterHash
- **Konfiguration:**
  - Anzahl Boards: 6
  - Anzahl Histories pro Board: 20
  - Anzahl Writer: 5
  - Anzahl Reader: 1
  - Threshold: 4

Der Writer ‘W0’ will bei diesem Szenario eine Nachricht innerhalb eines Threshold-Set publizieren. Dabei verwendet er aber einen falschen Hashwert beim publizieren auf das Board ‘B0’. Dies wird vom Board ‘B0’ bemerkt, wodurch das Board diese Nachricht nicht auf ihrer History publiziert und somit auch nicht an der verteilten Threshold-Signatur des angehörigen Threshold-Set teilnimmt. Da die Nachricht vom Writer ‘W0’ zu den anderen 5 Boards jedoch korrekt gehasht wurden, kommt die verteilte Threshold-Signatur trotzdem zustande, da die Nachricht auf genügend Boards publiziert werden konnte.

Demo im Simulator



6.2 Nichtzustandekommen der verteilten Threshold-Signatur über ein Threshold-Set

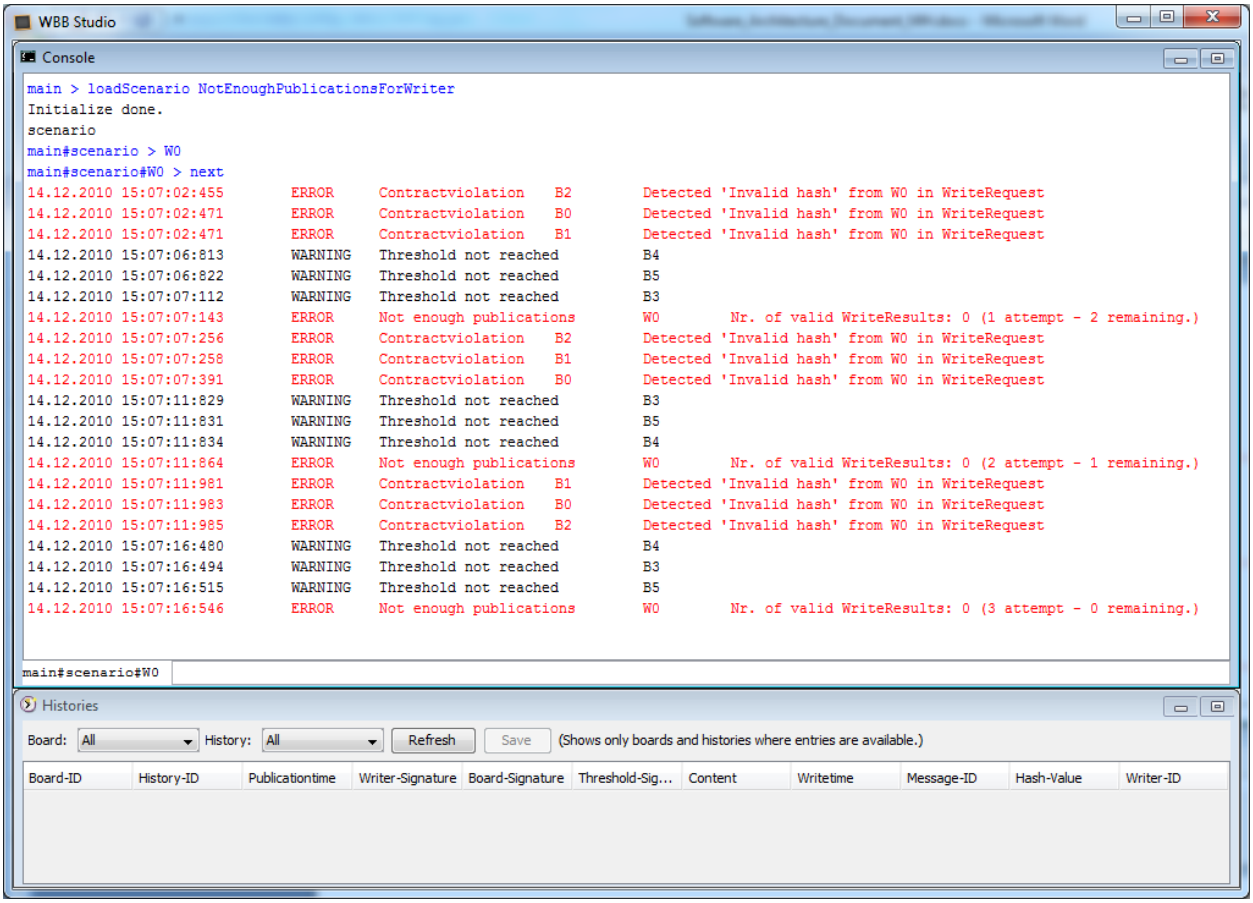
Beschreibung

Im folgenden Szenario wird demonstriert wie sich das System beim Nichtzustandekommen der verteilten Threshold-Signatur innerhalb eines Threshold-Sets verhält.

- **Szenarioname:** NotEnoughPublicationsForWriter
- **Konfiguration:**
  - Anzahl Boards: 6
  - Anzahl Histories pro Board: 20
  - Anzahl Writer: 4
  - Anzahl Reader: 1
  - Threshold: 4

Der Writer 'W0' will bei diesem Szenario eine Nachricht innerhalb eines Threshold-Sets publizieren. Dabei verwendet er aber einen falschen Hashwert bei der Publikation auf die Boards 'B0' – 'B2'. Dies wird von den entsprechenden Boards bemerkt, wodurch diese die Nachricht nicht auf ihrer History publizieren und somit auch nicht an der verteilten Threshold-Signatur des angehörigen Threshold-Sets teilnehmen. Die Nachrichten vom Writer 'W0' zu den restlichen 3 Boards wurden jedoch korrekt gehasht. Da der Threshold für das Threshold-Set jedoch bei 4 Boards liegt kommt die verteilte Threshold-Signatur mit nur 3 teilnehmenden Boards nicht zustande und die Nachricht wird auf keiner Boardhistory publiziert. Dies wird vom Writer bemerkt, wodurch dieser noch zwei weitere Publikationsversuche startet. Diese scheitern aufgrund des fehlerhaften Hashes jedoch ebenfalls.

Demo im Simulator



6.3 Feststellen einer falschen verteilten Threshold-Signatur eines Boards

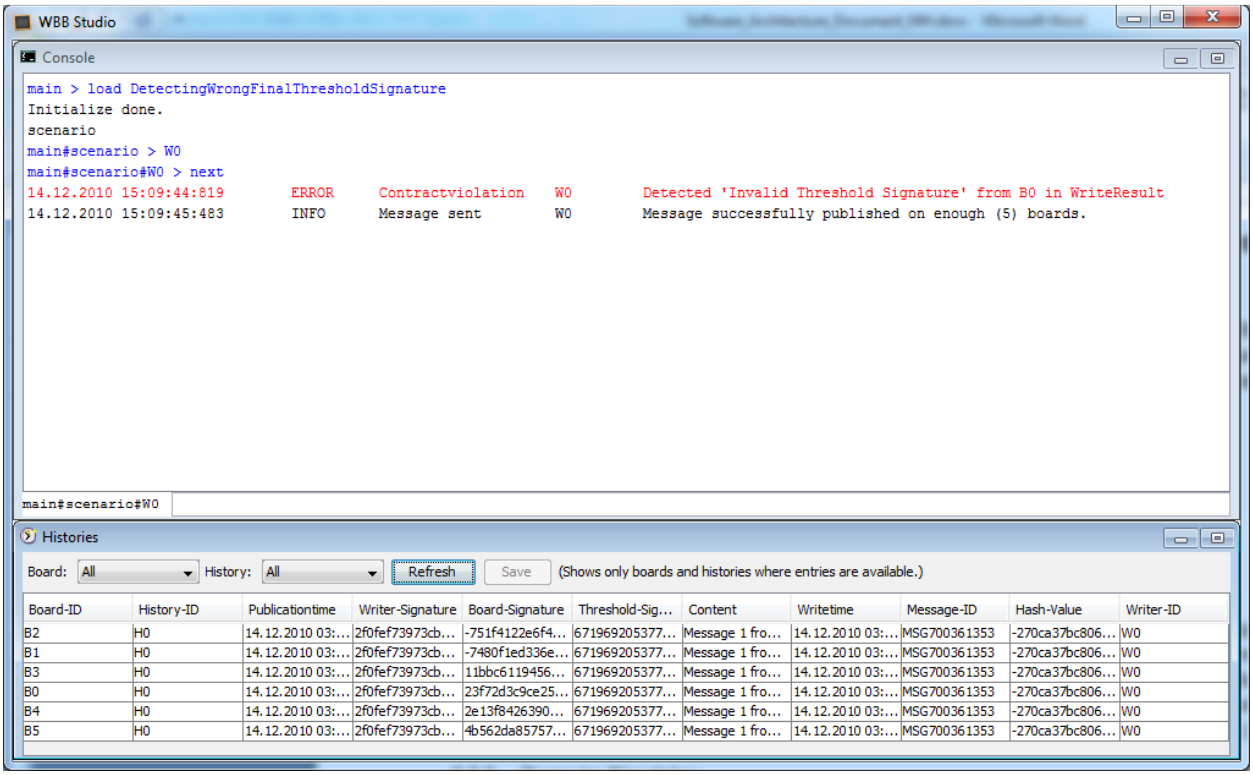
Beschreibung

Im folgenden Szenario wird demonstriert, wie der einzelne Writer feststellen kann, wenn ein Board nicht fähig war eine gültige verteile Threshold Signatur zu erstellen. Dies ist der Fall wenn das Board von den anderen Boards im Threshold-Set nicht genügend ‚InterBoardPublicationInformation' – Messages empfangen hat.

- **Szenarioname:** DetectingWrongFinalThresholdSignature
- **Konfiguration:**
  - Anzahl Boards: 6
  - Anzahl Histories pro Board: 20
  - Anzahl Writer: 4
  - Anzahl Reader: 1
  - Threshold: 4

Der Writer 'W0' will bei diesem Szenario eine Nachricht innerhalb eines Threshold-Sets publizieren. Dadurch, dass das Board ‚B0' eine falsche verteile Threshold-Signatur zurückschickt wird dies vom Writer detektiert und die Annahme getroffen, dass die entsprechende Nachricht auf diesem Board nicht publiziert wurde. Da jedoch die gültige verteilte Threshold-Signatur eines Boards genügt, wurde die Nachricht trotzdem auf genügend Boards publiziert.

Demo im Simulator



6.4 Feststellen einer falschen partiellen Threshold-Signatur eines Boards

Beschreibung

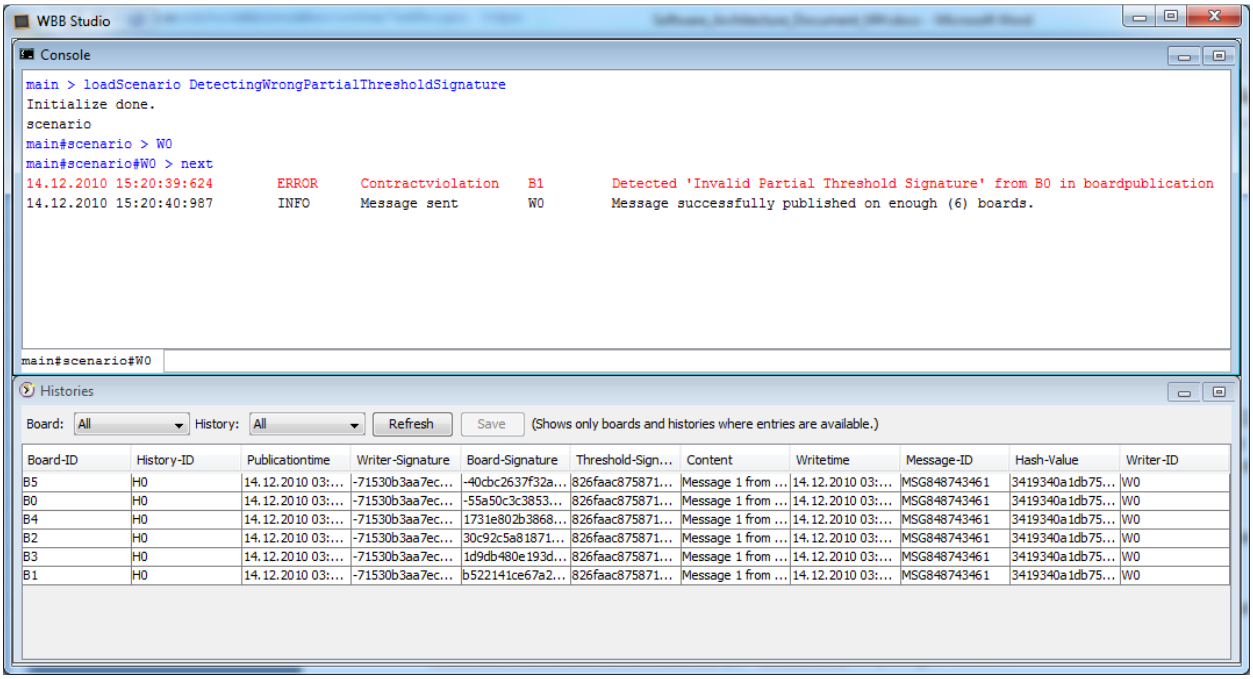
Im folgenden Szenario wird demonstriert, wie sich ein Board beim detektieren einer falschen partiellen Threshold-Signatur von einem anderen Board verhält.

- **Szenarioname:** DetectingWrongPartialThresholdSignature
- **Konfiguration:**
  - Anzahl Boards: 6
  - Anzahl Histories pro Board: 20
  - Anzahl Writer: 4
  - Anzahl Reader: 1
  - Threshold: 4

Der Writer 'W0' will bei diesem Szenario eine Nachricht innerhalb eines Threshold-Sets publizieren. Dazu sendet der Writer die Nachricht an alle sich im Threshold-Set befindenden Boards. Darauf prüfen die Boards die Nachricht und versenden an alle anderen Boards eine InterBoardPublicationInformation, mit der Nachricht, dass sie vorhaben, die soeben empfangene Nachricht zu publizieren. Diese InterBoardPublicationInformation wird ebenfalls mit dem persönlichen RSA-Key und dem partiellen Threshold-Key signiert. Mittels Zero-Knowledge Proof kann ein Board die Echtheit einer empfangenen partiellen Threshold-Signatur überprüfen.

Im folgenden Beispiel war die partielle Threshold-Signatur welche das Board ‚B0' an das Board ‚B1' gesendet hatte fehlerhaft – was vom Board auch korrekterweise festgestellt wurde. Die Message wurde jedoch trotzdem auf allen Boards publiziert, da alle Boards genügend gültige InterBoardPublicationInformationen (>= Threshold) empfangen haben.

Demo im Simulator



6.5 Nichterreichen des Threshold bei BoardPublikationen

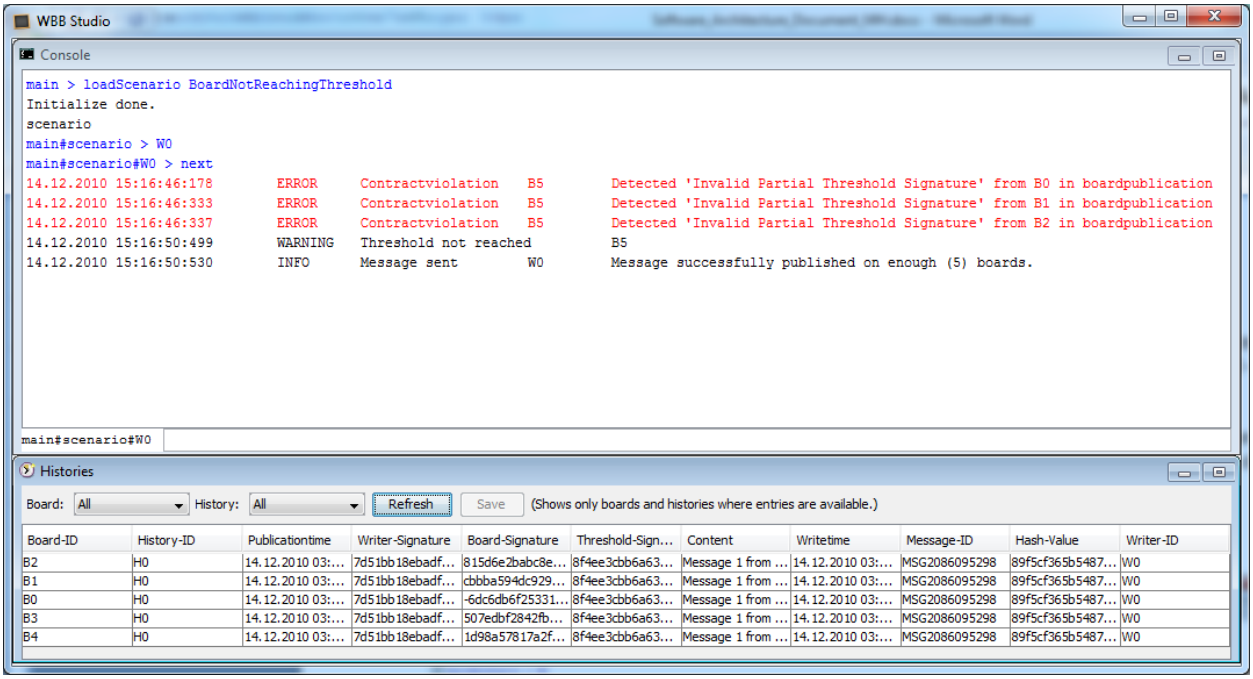
Beschreibung

Im folgenden Szenario wird demonstriert, wie das einzelne Board gültige InterBoardPublicationInformationen von mindestens einer Anzahl Boards empfangen haben muss, die grösser ist als der Threshold, um dem Writer eine gültige verteilte Threshold-Signatur auszustellen.

- **Szenarioname:** BoardNotReachingThreshold
- **Konfiguration:**
  - Anzahl Boards: 6
  - Anzahl Histories pro Board: 20
  - Anzahl Writer: 4
  - Anzahl Reader: 1
  - Threshold: 4

Der Writer 'W0' will bei diesem Szenario eine Nachricht innerhalb eines Threshold-Sets publizieren. Im Beispiel beinhalten die InterBoardPublicationInformationen welche das Board B5 von den Boards B0 – B2 erhält, korrupte partielle Threshold-Signaturen. Da nun die Anzahl gültiger InterBoardPublicationInformationen kleiner als der Threshold ist, kann das Board B5 dem Writer W0 keine verteilte Threshold-Signatur ausstellen und publiziert die empfangene Nachricht auch auf keiner History. Auf den restlichen 5 Boards kam die verteilte Threshold-Signatur jedoch zustande, wodurch die Nachricht trotzdem auf den 5 Boards publiziert wurde. Damit ist der Writer ‚W0' zufrieden, da diese Anzahl grösser ist als der Threshold von 4.

Demo im Simulator



6.6 Versuch ein Board mit ReadForAppending-Requests zu überlasten

Beschreibung

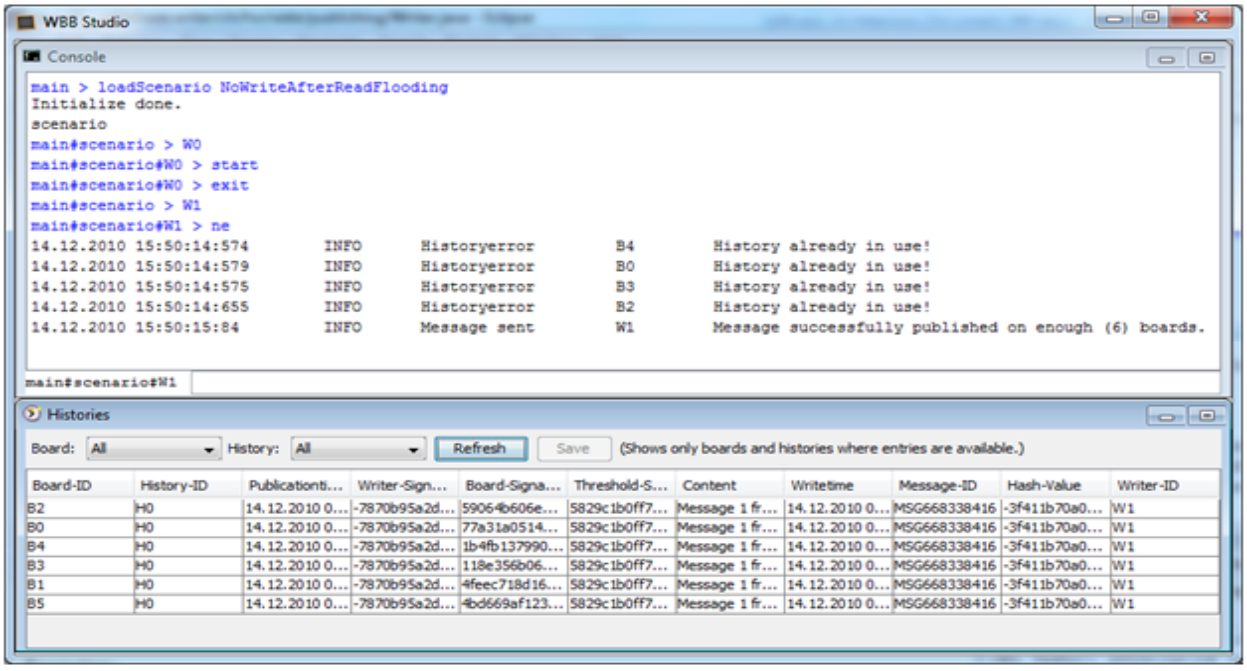
In folgendem Szenario wird demonstriert wie sich das System bei kurz aufeinanderfolgenden ReadForAppending-Requests von einem Writer ohne darauffolgende Write-Requests verhält.

- **Szenarioname:** NoWriteAfterReadFlooding
- **Konfiguration:**
  - Anzahl Boards: 6
  - Anzahl Histories pro Board: 8
  - Anzahl Writer: 4
  - Anzahl Reader: 1
  - Threshold: 4
  - W0: taskRun.updateInterval = 2 (ms)

Der Writer ‚W0' versucht in diesem Szenario die Boards mittels nicht kompletten Publish-Versuchen in die Knie zu zwingen, respektive die anderen Writer in der WBB-Umgebung verhungern zu lassen. Dazu setzt der Writer alle 2ms einen ReadForAppending-Request auf alle Boards ab und erhält dadurch jeweils immer die nächste freie History. Auf diesen ReadForAppending-Request würde im Normalfall ein Write-Request folgen, diesen lässt der korrupte Writer ‚W0' jedoch weg und startet sogleich den nächsten ReadForAppending-Request.

Will nun ein Writer ‚W1' ebenfalls eine Nachricht publizieren, so kriegt er vom Board auf jeden Fall eine freie History, da die Histories erst bei einem write-Request gelockt werden. Wurde eine History erst bei einem ReadForAppending-Request vergeben, so kann es vorkommen, dass diese zweimal vergeben wird falls alle Histories vergeben wurde. Der Writer ‚W1' setzt nun nach der Zuweisung der History einen Write-Request ab, dadurch wird die History vom HistoryManager gelockt. Der korrupte Writer ‚W0' ‚überrundet' nun je nach Anzahl Histories den Writer ‚W1' wieder, die gelockte Historie wird beim ReadForAppending-Request zwar herausgegeben, der Writer ‚W0' könnte darauf aber keine Nachricht publizieren, da diese gelockt ist.

Demo im Simulator



Bemerkung: Im obigen Screenshot wurden die Errormessages vom korrupten Writer ‚W0' ausgeblendet. Zu beachten ist zudem wie keine Nachricht vom Writer ‚W0' auf einer History gelandet wurde, da der Write-Request gar nie ausgeführt wurde.

Veranschaulichung

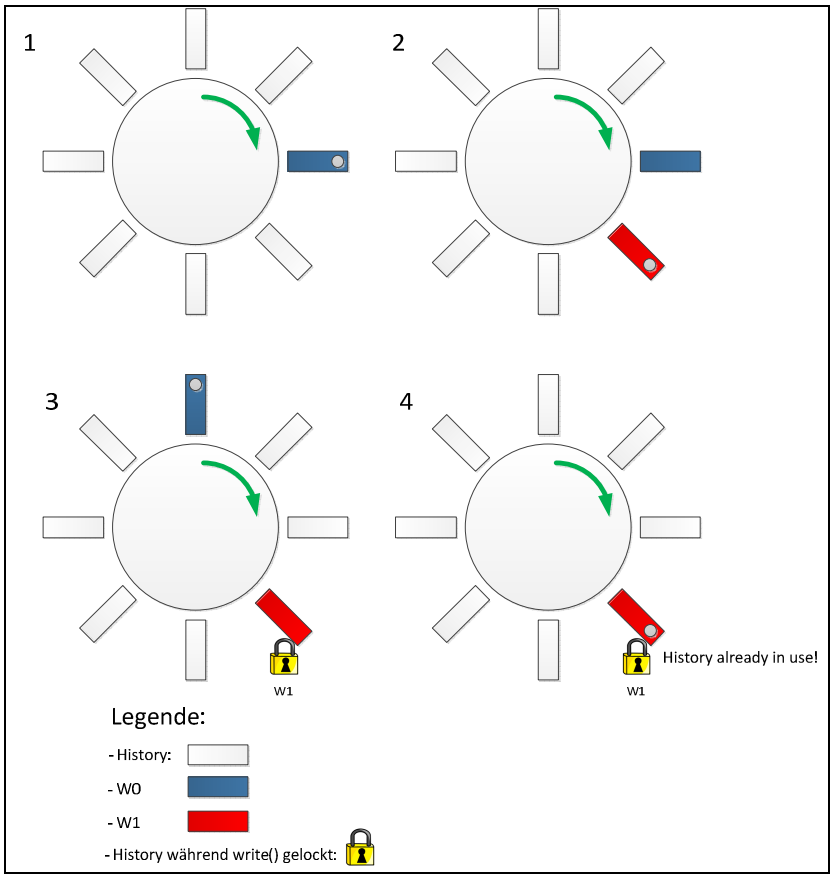


Abbildung: Sicht auf ein Board mit 8 Histories

- In Phase 1 setzt der Writer ‚W0' im Intervall von 2ms ReadRequests auf die Boards ab und kriegt jeweils eine freie History zugewiesen. Der Writer ‚W0' verzichtet aber danach auf das Absetzen des WriteRequest, wodurch die herausgegebene History nicht gelockt wird und für spätere ReadRequests herausgegeben werden kann, wenn alle anderen Histories einmal herausgegeben wurden. Der Historycontainer gibt hier jeweils die nächste HistoryID nach dem RoundRobin – Prinzip zurück.
- In Phase 2 setzt der Writer ‚W0' einen regulären ReadRequest ab wonach er ebenfalls die nächste HistoryID zugewiesen bekommt.
- In Phase 3 setzt der Writer ‚W1' den WriteRequest ab wodurch die History gelockt wird. Dies mittels eines gesetzten Timeouts innerhalb welchem das Board nun auf die InterBoardPublicationInformationen der anderen Boards wartet um die Nachricht definitiv auf der History publizieren dürfen zu können.
- In Phase 4 ist der korrupte Writer ‚W0' einmal durch alle Histories durchgegangen und gelangt nun auf die History welche durch den Writer ‚W1' gelockt ist. Je nach Anzahl Histories ist dies bei einem Intervall von 2ms sehr schnell möglich. Dies stellt für das System jedoch kein Problem dar. Die History wird zwar herausgegeben und eine LOG-Nachricht ‚History already in use!' abgesetzt – jedoch kann der Writer darauf keine Nachricht absetzen, da die History gelockt ist. Dieses Fehlerhandling muss der Writer implementieren.

Falls diese spezielle Situation einmal bei einem ‚normalen' Writer auftreten sollte, sieht die Standardimplementation vor, dass eine neue History per ReadRequest angefordert wird auf die dann versucht wird die Nachricht zu publizieren.

6.7 Mehrere korrupte Writer blockieren das Threshold-Set

Beschreibung

Im folgenden Szenario wird demonstriert wie sich das verhält, wenn eine Anzahl Boards kleiner als der Threshold ein ganzes Threshold-Set blockieren wollen, indem sie eine Nachricht jeweils nicht auf allen verfügbaren Boards zu publizieren versuchen.

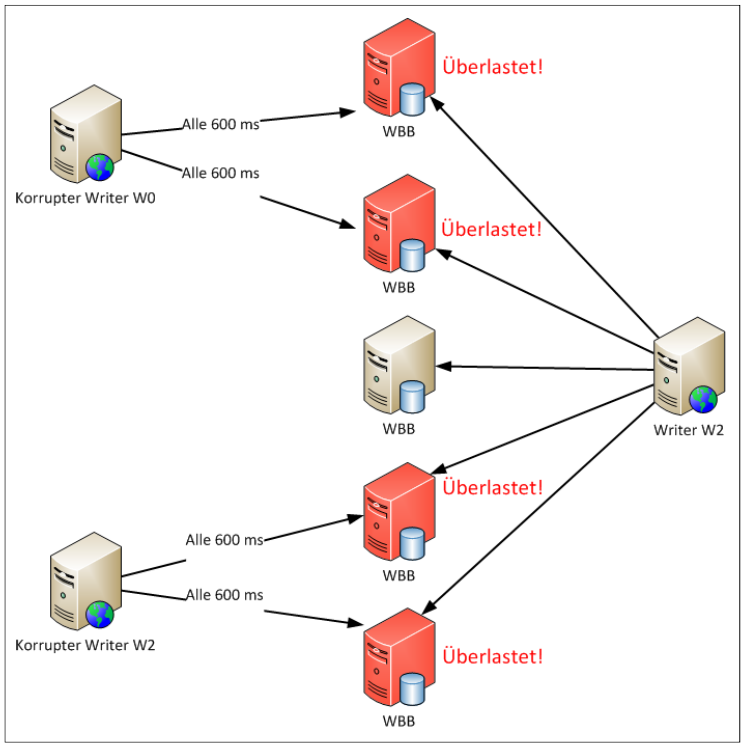
- **Szenarioname:** blockThreshold
- **Konfiguration:**
  - Anzahl Boards: 5
  - Anzahl Writer: 3
  - Anzahl Reader: 1
  - Anzahl Histories pro Board: 5
  - Threshold: 3
  - W0
    - taskRun.updateInterval = 600 (ms)
    - taskRun.useRandom = false
    - taskRun.useFixRate = true
  - W2
    - taskRun.updateInterval = 600 (ms)
    - taskRun.useRandom = false
    - taskRun.useFixRate = true

Writer ‚W0' und ‚W2' seien in diesem Beispiel korrupt und versuchen die WBB-Umgebung für andere Writer zu blocken, sodass deren Nachrichten nicht mehr auf genügend Boards publiziert werden. Dazu versucht ‚W0' in häufigen Intervallen seine Nachrichten auf dem Board ‚B0' und ‚B1' zu publizieren. Bei den restlichen Boards im Threshold-Set setzt er keine WriteRequest ab. Dadurch dass die Zahl der Boards auf die W0 zu publizieren versucht kleiner als der Threshold (3) des Threshold-Sets ist, kommt eine Nachrichtenpublikation nicht zustande. Die 2 Boards warten jeweils den vollen Timeout ab während dem die Histories gesperrt bleiben, was bei dem Intervall bei einer kleinen Anzahl schnell alle Histories sind.

Analog dazu führt der Writer ‚W2' seine „Attacke" auf zwei andere Boards ‚B3' und ‚B4' aus, wodurch dieser die Histories dieser zwei Boards belegt.

Für die anderen Writer im Threshold-Set wird es nun erschwert, deren Nachrichten auf genügend Boards zu publizieren, da der Threshold aufgrund der belegten Histories meist nicht mehr zustande kommt.

Wirkliche Abhilfe gegen dieses Problem schafft eigentlich nur die Massnahme, bei der jeder Writer eine beschränkte Anzahl Write-Slots zur Verfügung gestellt bekommt, wodurch er nicht mehr alle Histories belegen und locken kann. Die Anzahl Histories pro Board zu erhöhen ist nur eine Scheinlösung, da bei genügend kleinem Updateintervall fast jede Anzahl Histories geblockt werden kann.



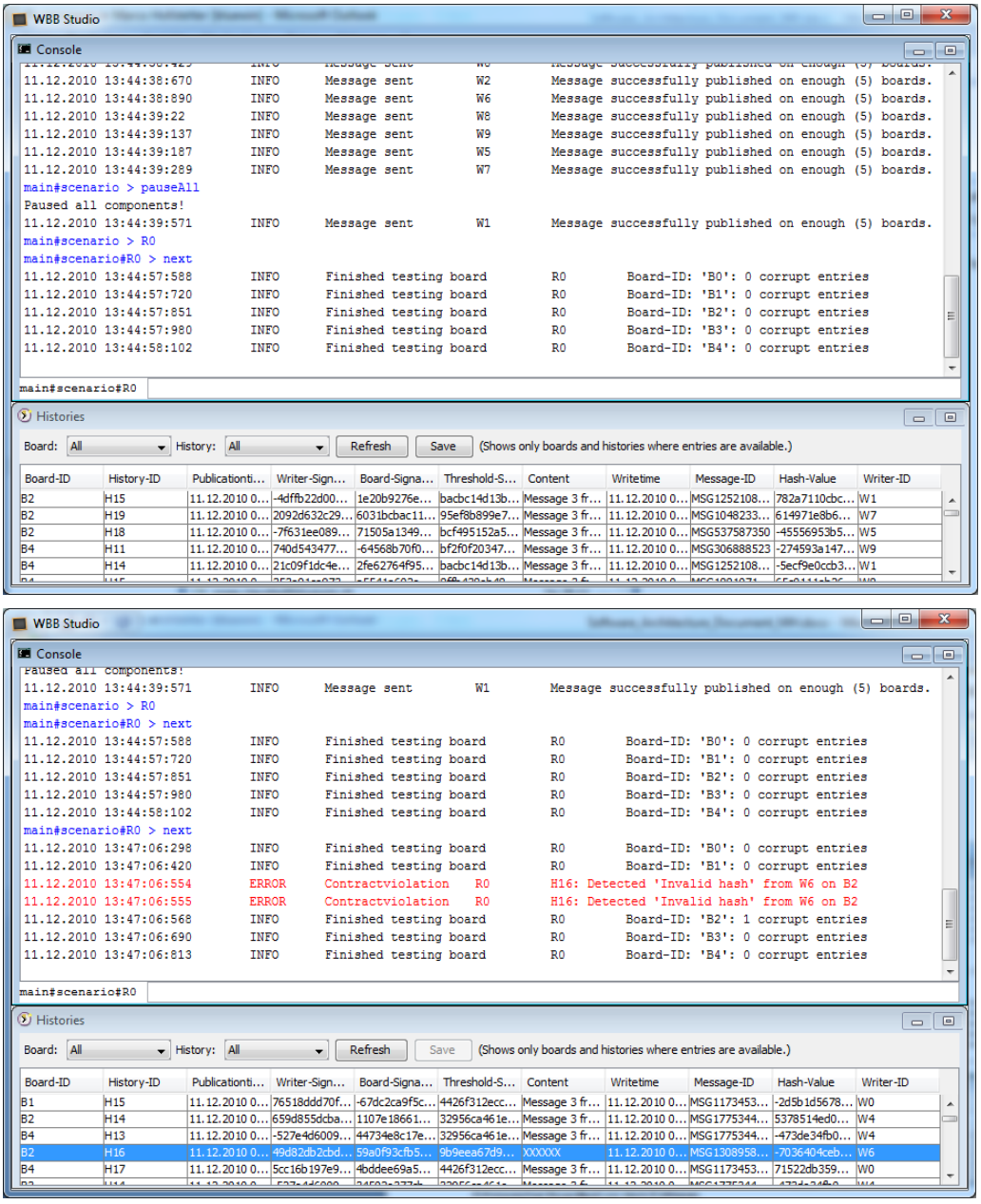
6.8 Reader stellt nachträglich geänderte Historyentries auf Boards fest

Beschreibung

Im folgenden Szenario wird demonstriert, wie ein nachträgliches Verändern von Daten innerhalb der Datenbank vom Reader festgestellt wird. Dabei fordert der prüfende Reader von den zu überprüfenden Boards innerhalb des Threshold-Sets alle HistoryEntries an und überprüft diese auf deren Korrektheit bezüglich Hash, Signatur des Boardes, Signatur des Writers und der Threshold-Signatur. Ebenfalls wird mittels des Hashes geprüft, dass keiner der Einträge im Nachhinein heraus gelöscht wurde. Dies wird dadurch überprüft, dass der Hash des vorhergehenden Historyeintrages auch in den Hash des nächsten Historyeintrages einfliesst.

Entdeckt der Reader einen korrupten Eintrag, so gibt es dies mit der entsprechenden Board-ID und der History-ID aus.

Demo im Simulator



Boardtest mit Detektierung einer Veränderung vor und nach dem Editieren

6.9 Die Boards besitzen eine zu kleine Anzahl an Histories für die Anzahl Writer im WBB-Environment

Beschreibung

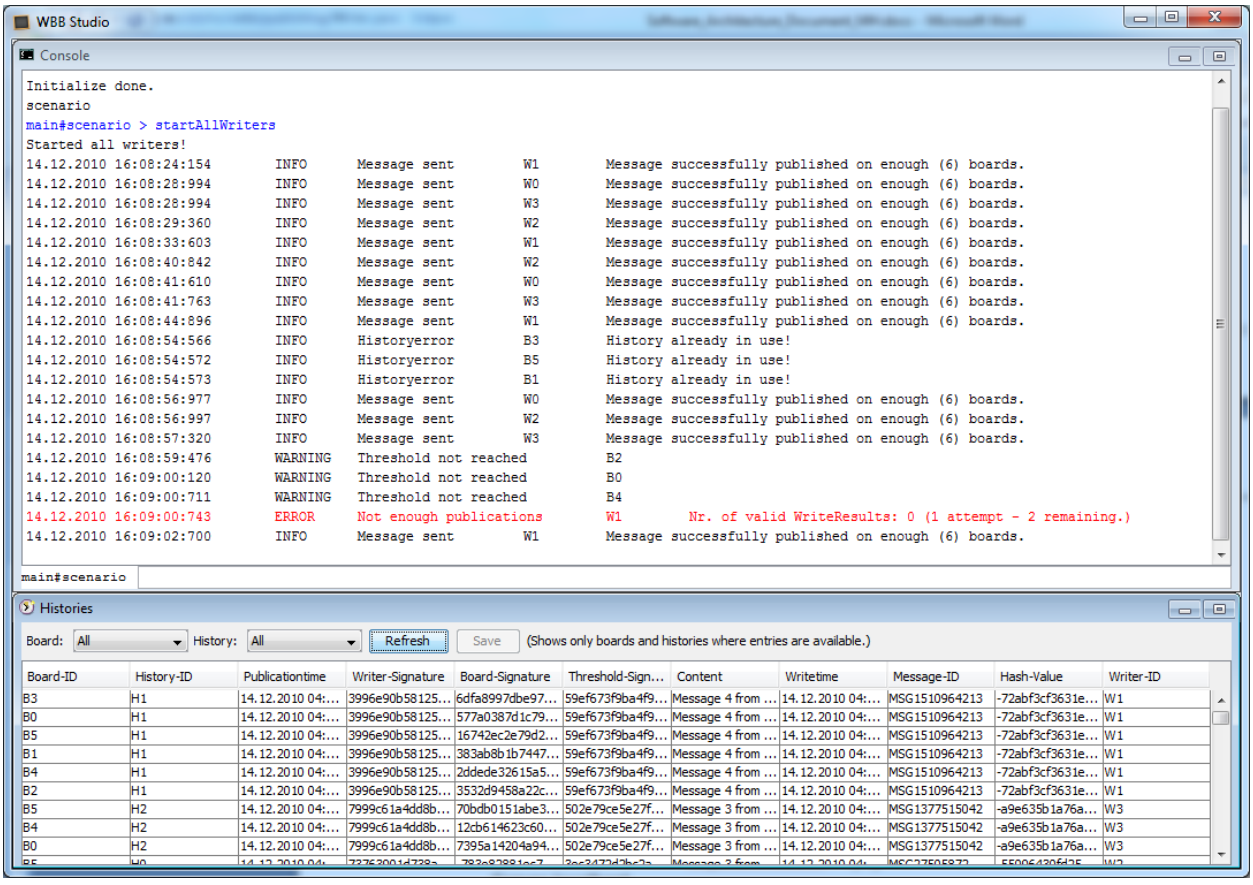
In folgendem Szenario wird demonstriert, wie sich die WBB-Umgebung verhält, wenn ein Board weniger Histories besitzt als Writer in der Umgebung sind.

- **Szenarioname:** TooFewHistories
- **Konfiguration:**
  - Anzahl Boards: 6
  - Anzahl Writer: 4
  - Anzahl Reader: 1
  - Anzahl Histories pro Board: 3
  - Threshold: 3

Falls die vier Writer ungefähr zeitgleich auf die WBB-Umgebung publizieren wollen, kann es vorkommen, dass für den letzten Writer, in diesem Beispiel der Writer ‚W1', die Nachricht nicht mehr publiziert werden kann. Die Boards ‚B0', ‚B2' und ‚B4' hatten hier jeweils keine freie History mehr. Dadurch kam der Threshold von 4 nicht zustande. Der Writer detektiert dass die Nachricht auf ungenügend vielen Boards publiziert wurde und versucht ein weiteres Mal die Nachricht zu publizieren. Im zweiten Anlauf konnte der Writer seine Nachricht ebenfalls erfolgreich publizieren.

Hierbei sei zu Bemerken, dass die Anzahl Histories über die jedes Board verfügt, jeweils der Anzahl Writer anzupassen ist, damit das System genügend skaliert.

Demo im Simulator



7 Installation

7.1 WBB Studio (& Postgres)

Die ZIP-Datei wbb.zip muss irgendwohin entpackt werden. Nach dem korrekten aufsetzten des Datenbankserver und des Anlegen der Datenbank kann das wbb_simulation.jar ausgeführt werden.

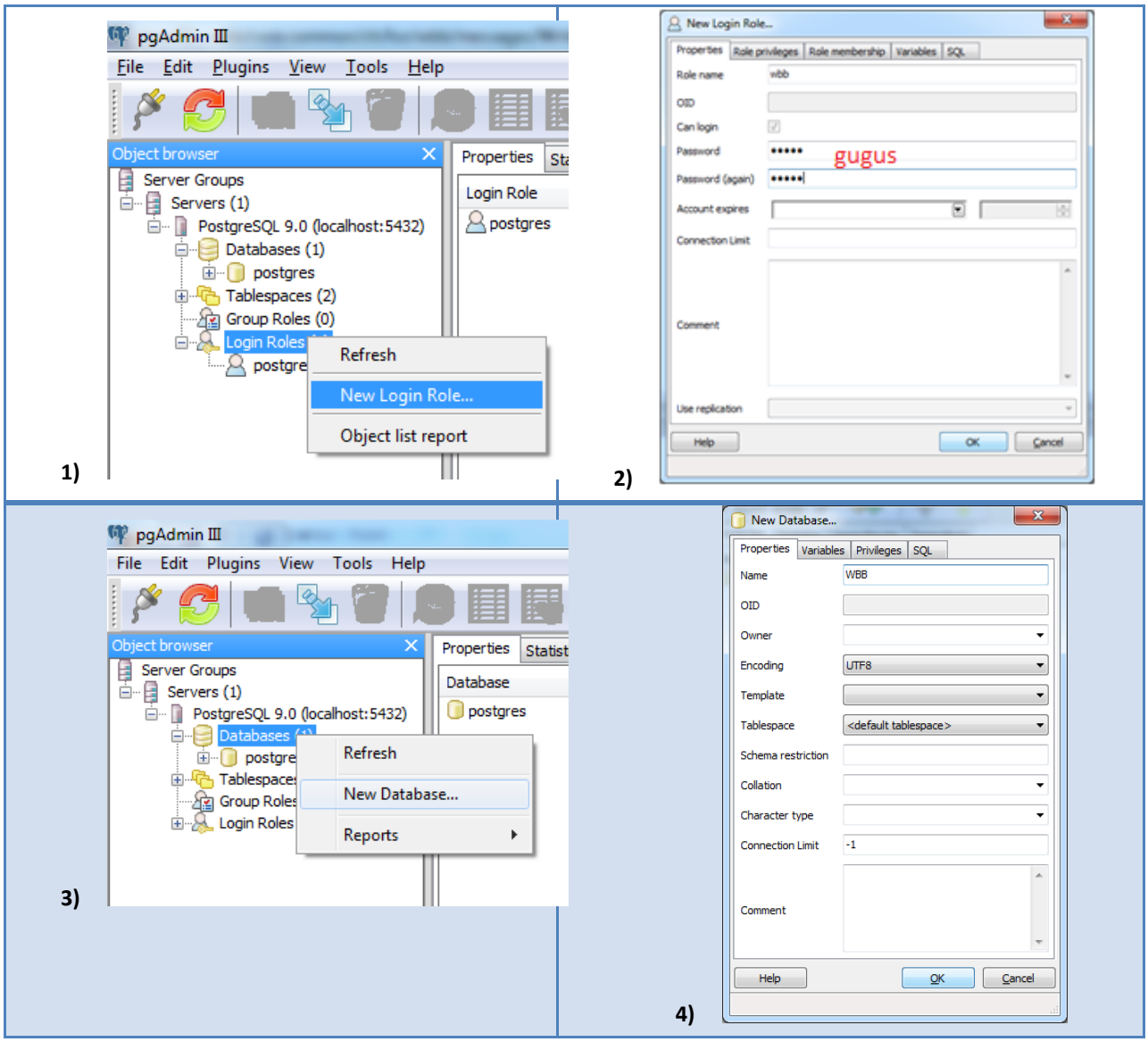
Um das WBB Studio laufen zu lassen, werden folgende Komponenten vorausgesetzt:

- Java Runtime Version 6
- Postgres Datenbankserver.

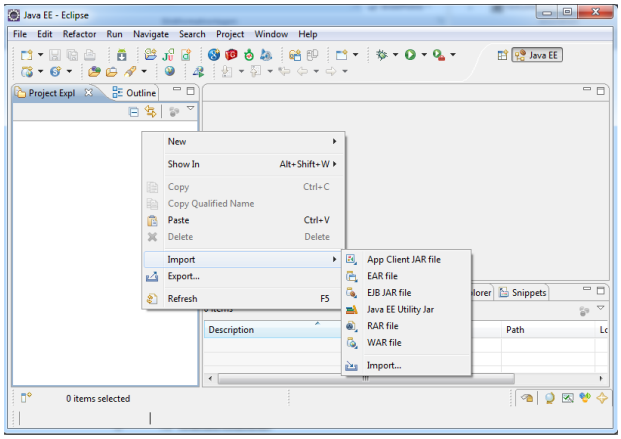
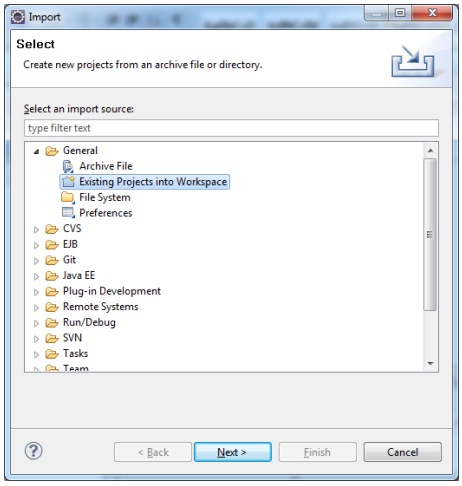
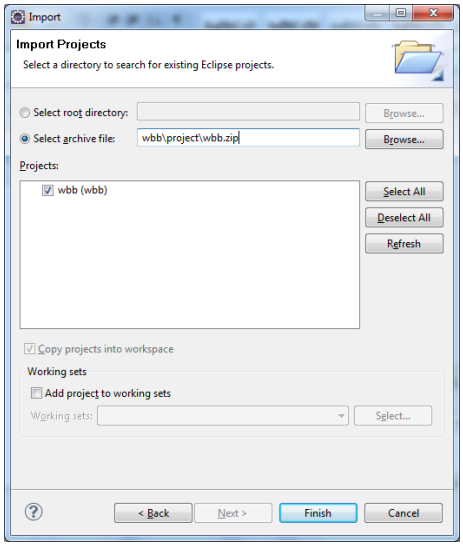
Auf den Datenbankserver muss folgendermassen zugegriffen werden können:

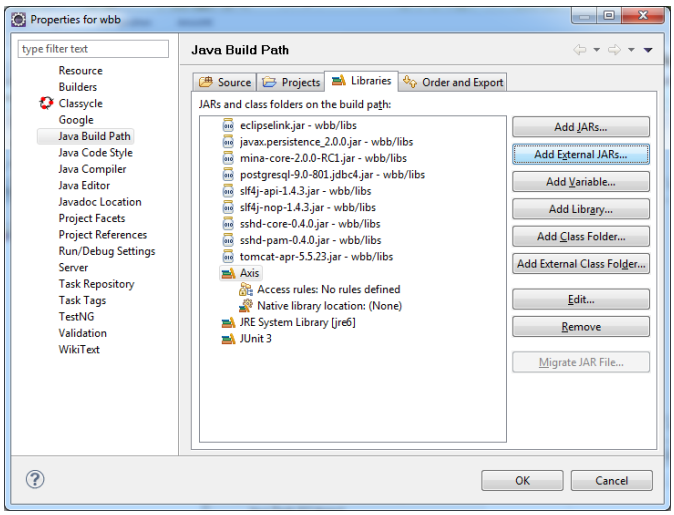
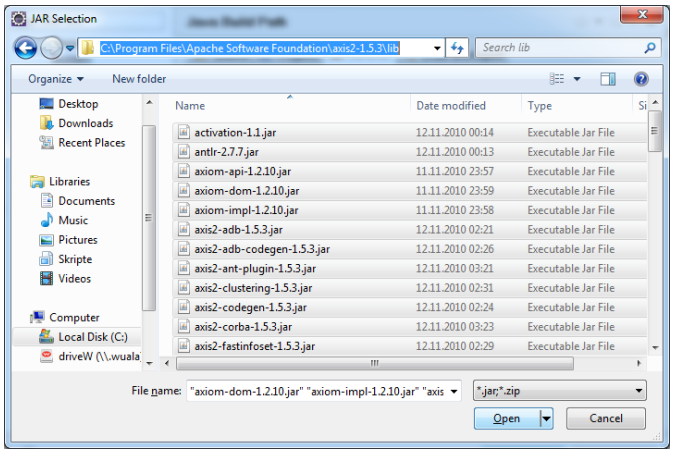
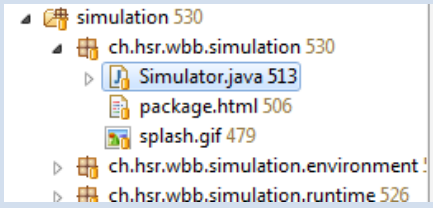
- Server: localhost:5432
- Loginrole: wbb
- Passwort: gugus
- Datenbank: WBB

Es muss zusätzlich auch eine leere Datenbank mit dem Name ‚WBB' angelegt werden. Die Tabellen werden automatisch beim ersten Start des WBB Studio angelegt.



7.2 Einrichten der Entwicklungsumgebung

Beschreibung	Bild
Eclipse IDE for Java EE Developers herunterladen (http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr1) • Neuen Workspace erstellen	-
Projekt 'WBB' importieren • Quelle: Source/wbb_source.zip	  

Apache Axis2 für Webservices installieren Apache Axis2 Version 1.5.3 Binary Files an beliebigen Pfad entpacken • Source/axis2-1.5.3-bin.zip oder alternativ • http://axis.apache.org/axis2/java/core/download.cgi	-
Apache Axis2 UserLibrary in Eclipse JEE einbinden Kontextmenu auf Projekt 'wbb' • ➔Menu 'Properties' ➔ Java Build Path ➔ Register 'Libraries' • UserLibrary 'Axis' ➔ 'Add External JAR's..' • Importiere alle Libraries aus dem zuvor entpackten Axis2-ZIP-File • Mit ‚OK' abschliessen	 
Einstiegspunkt in den Simulator ist die Klasse ‚Simulator' • Package: ch.hsr.wbb.simulation im Source Folder 'simulation'	

7.3 ANT Auto-Deployment

Im Ordner ant befindet sich das Ant-Build Skript mit den Tasks für verschiedene Deploymentvarianten.

7.3.1 Standard Build-Tasks

Um die Standard-Build-Tasks in Eclipse ausführen zu können, müssen die folgenden Punkte erfüllt sein:

- Für die korrekte Funktionsweise von JavaDoc muss die genutzte Installed JRE auf den Ordner mit der jdk zeigen.
- Der Pfad auf die Axis2 lib Ordner muss im Property axis.path angepasst werden.

Name	Beschreibung
export-source	Exportiert alle Quelldateien in eine Zip-Datei. Die Datei wird im Ordner ant/bin/project abgelegt. Die Zip-Datei kann dann wiederum im eclipse als ein Projekt importiert werden.
build	Kompiliert die vorhanden Java-Klassen und packt die resultierenden .class-Files in eine jar-Datei. Für jeden Source-Folder im eclipse wird auch ein eigenes jar generiert. Am Ende des Dateinamen wird die Versionsnummer des aktuellen Builds angehängt. Die kompilierten Dateien liegen unter ant/bin/temp/classes und die generierten jar-Dateien unter ant/bin/temp/libs. In das libs-Verzeichnis werden ebenfalls die benötigen externen jars kopiert.
clean	Löscht alle von einem Ant-Task generierten Dateien.
javadoc	Erstellt die javadoc-Htmlseiten für das gesamte Projekt. Die resultierenden Dateien werden nach ant/bin/temp/doc kopiert.
jar	Erstellt das Runnable-jar, welches das wbb-Studio startet. Die resultierende jar-Datei mit dem Namen wbb_studio.jar wird in das ant/bin/temp/jars Verzeichnis kopiert.
zip	Dieser Task erlaubt es, alle Dateien welche benötigt werden, um das wbb-Studio laufen zu lassen, in eine zip-Datei zu verpacken. Die resultierende zip-Datei wird unter ant/bin/zip abgelegt. Die zip-Datei kann auf dem Zielrechner einfach ausgepackt werden und das wwb_studio.jar kann ausgeführt werden.

7.3.2 Remote Build-Tasks

Die Remote-Tasks erlauben es eine Instanz auf einen externen Rechner zu kopieren und die Simulation zu starten. Damit dies möglich ist muss auf dem Zielrechner ein SSH-Server installiert sein. Es muss weiter ein Benutzer mit dem Namen wbb per SSH haben. Für die Authentifizierung wird über eine RSA Signatur gelöst. Der öffentliche Schlüssel liegt ist in der id_rsa.pub Datei hinterlegt. Der Inhalt dieser Datei muss in die authorized_keys Datei im SSH-Verzeichnis des wbb Benutzers kopiert werden. Eine weitere Voraussetzung ist, dass im lib Verzeichnis unter des Ant-Home Verzeichnis eine Version von Java Secure Channel (JSch) liegt (Download unter http://www.jcraft.com/jsch/)

Es gibt die Möglichkeit die Zielsysteme zu konfigurieren. Dazu müssen alle Hosts in den Tasks remote-clean, remote-deploy, remote-start und remote-stop eingetragen werden.

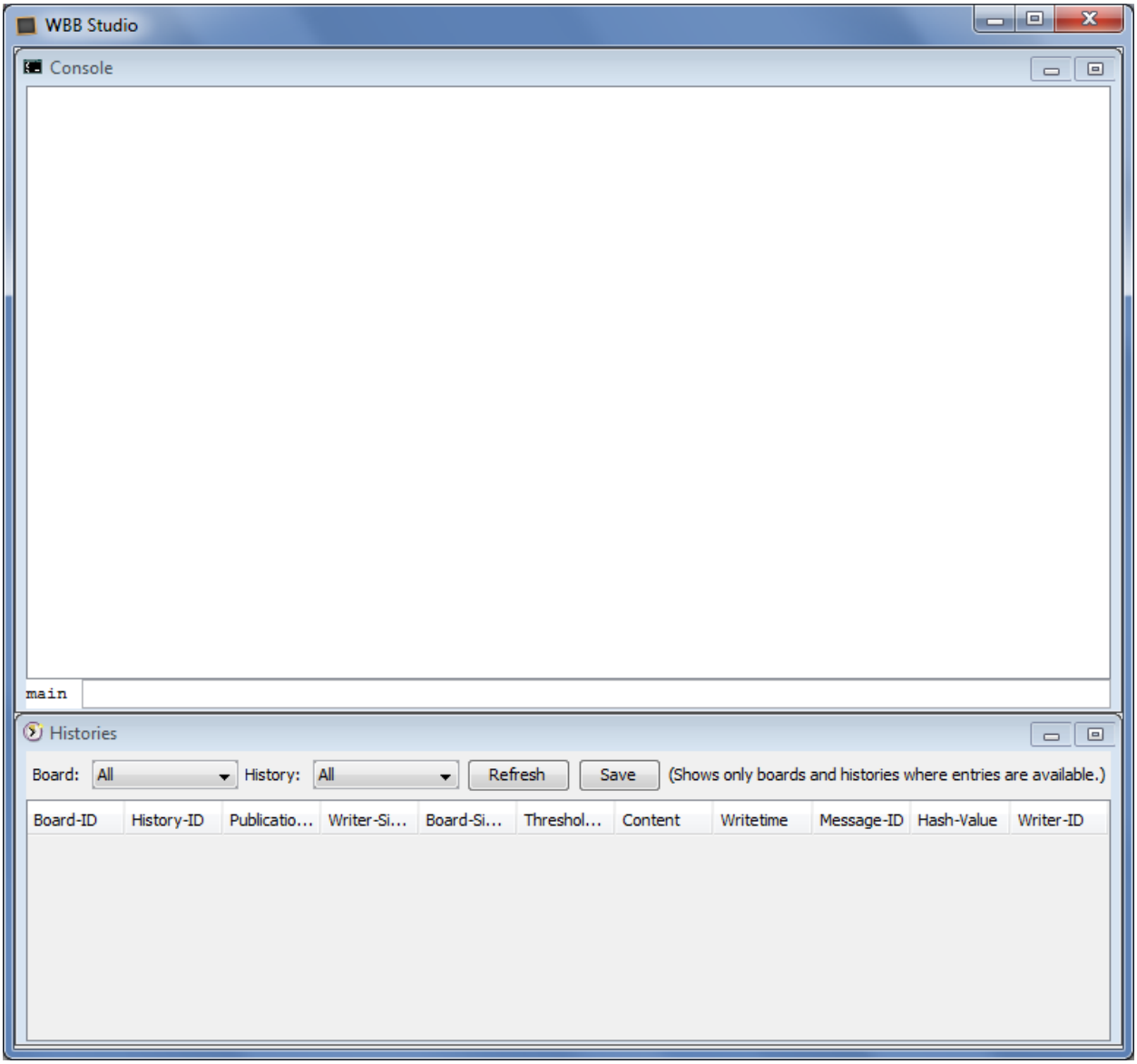
Name	Beschreibung
remote-clean	Es werden alle kopierten Dateien von allen Zielsystemen entfernt.
remote-deploy	Stoppt etwaig laufende Simulationen auf den Zielsystemen und kopiert alle benötigten Dateien auf die Zielrechner.
remote-start	Führt als erstes den remote-deploy Task aus. Danach werden die Simulationen auf allen Zielrechnern gestartet.
remote-stop	Stoppt alle Simulationen auf den Zielrechnern.

7.4 Verwendete Komponenten

- Java Runtime Environment 6.0 (http://www.oracle.com/us/sun/)
- Postgres 9.0 Datenbank (http://www.postgresql.org/)
- Eclipselink-2.1.1.v20100817-r8050 (http://www.eclipse.org/eclipselink/)
- Apache Axis2 (http://axis.apache.org/axis2/java/core/download.cgi)
- SSH (Apache Mina SSHD) (http://mina.apache.org/sshd/sshd-050.html)

8 Benutzerhilfe

Nach dem Start des WBB Studio erscheint folgender Bildschirm:



Das obere Fenster beinhaltet die Konsole und im unteren Fenster können die Einträge aus der Datenbank gelesen werden. Beide werden im Folgenden noch detaillierter beschrieben.

8.1 Konsole

Das Konsolenfenster bietet die Möglichkeit eine Simulation zu konfigurieren oder während dem Ablauf der Simulation noch Einfluss darauf zu nehmen. Dabei können im unteren Eingabefeld Kommandos abgesetzt werden. Das Resultat des Kommandos und mögliche Log-Ausgaben erscheinen dann im oberen Teil des Fensters.

8.1.1 Absetzten eines Kommandos

Um ein Kommando auszuführen wird der Name des Kommandos im unteren Feld eingegeben. Mit Enter bestätigt man die Eingabe und das Kommando wird ausgeführt. Das Kommando kann auch über Parameter verfügen. Diese werden mit einem Leerschlag voneinander getrennt eingegeben.

Die Eingabe sollte folgende Struktur haben: [Name des Kommandos] [Parameter1] [Parameter2] […]

Wenn ein Parameterwert über einen Leerschlag verfügen muss, dann muss dieser Wert von zwei Anführungszeichen (“) umschlossen sein.

Verwendung der Autovervollständigung

Das Skriptmodul der Simulation verfügt über eine Autovervollständigung für die Kommandonamen und die möglichen Parameterwerte. Um einen Vorschlag für ein mögliches Kommando oder Parameterwert zu erhalten kann die Tabulatortaste gedrückt werden. Beim mehrmaligen Drücken der Taste werden die nächsten Vorschläge dargestellt. Um wieder den vorhergehenden Wert zu erhalten kann Shift+Tab gedrückt werden.

Kommandoverlauf

Über die UP-Taste können die vorhergehenden Eingaben abgerufen. Mit der DOWN-Taste kann innerhalb des Verlaufs navigiert werden.

8.1.2 Ausgabe

In der Ausgabe erscheint jeweils nochmals das eingegebene Kommando. Danach wird die Ausgabe des einzelnen Kommandoaufrufs ausgegeben. In folgenden Format [Kontext] > [Command]. Weitere Informationen dazu sind im Kapitel Skriptmodul zu finden.

8.2 Skriptmodul

Das Skriptmodul kann über die Konsole angesprochen werden.

8.2.1 Kontext

Die aktuelle Ausgabe des Skriptmoduls befindet sich immer in einem bestimmten Kontext. Der Kontext bestimmt, welche Kommandos verfügbar sind. Ein Kontext kann verschachtelt sein. Das heisst, dass mit einem Aufruf eines bestimmten Kommandos der Kontext gewechselt wird. In der Ausgabe ist die Verschachtelung der Kontexte ersichtlich. Und zwar werden die einzelnen Kontext durch eine #-Symbol getrennt. Um einen Kontext zu verlassen und zu dem ursprünglichen Kontext zu gelangen, kann das exit Kommando verwendet werden.

8.2.2 Hilfe

Um Hilfe über den aktuellen Kontext zu erhalten kann das help Kommando verwendet werden. Dabei werden alle verfügbaren Kommandos des aktuellen Kontexts aufgelistet. Um detailliertere Informationen über ein einzelnes Kommando zu erhalten kann help [Name des Kommandos] aufgerufen. Danach erhält man auch einen Auflistung der benötigen Parameterwerte.

8.2.3 Globale Kommandos

Globale Kommandos können aus jedem Kontext aufgerufen. Dabei sind sie am Anfang des Namen mit einem Punkt gekennzeichnet. Mit .help können alle verfügbaren globalen Kommandos abgerufen werden. Mit .clear kann so zum Beispiel aus irgendeinem Kontext das Ausgabefenster gelöscht werden. Provoziert ein globaler einen Kontextwechsel, so kann mit .leave wieder in den ursprünglichen Kontext gewechselt werden.

8.2.4 Alias

Einige Kommandos verfügen über einen Alias. Über diesen Alias kann das Kommando ebenfalls aufgerufen werden. So kann das exit Kommando beispielsweise über ein einfaches Fragezeichen (?) und der exit über quit aufgerufen werden.

8.2.5 Simulation

8.2.5.1 Szenario

Ein Szenario setzt sich aus einer Anzahl Komponenten zusammen. Ebenfalls bezieht sie die Konfiguration der einzelnen Komponenten mit ein. Diese Informationen werden in eine .scenario Datei gespeichert.

In der Scenario-Datei befinden sich alle Informationen für ein Simulationsszenario. Die Dateien befinden sich im scenarios/saveScenarios Ordner und haben die .scenario Endung. Der Aufbau der Datei ist im nachfolgenden Kapitel beschrieben.

In der Simulationsumgebung kann nur ein Szenario geladen sein. In einer Simulation ohne Szenario kann auch ein neues Szenario kreiert werden.

Aufbau der Scenario-Datei

Die scenario-Datei ist eine einfache Zip-Datei welche die benötigt Ordnerstruktur beinhaltet. Als oberster Ordner wir ein Ordner benötigt, welcher denselben Name wie das Szenario hat. Darunter liegen dann die verschieden Ordner mit den Konfigurationsdateien für die einzelnen Komponenten. Die Ordner für die Board-Komponenten beginnen mit einem grossen B. Jene für die Writer mit W und die Reader mit R. Innerhalb dieser Ordner befinden sich jeweils vier Dateien für die Konfiguration. Diese lauten aspectSettings.ini, keyFile.ini, runtimeSettings.ini, simulationSettings.ini.

8.2.5.2 Aspekt

Ein Aspekt kann zwischen zwei Komponenten konfigurierte werden. Er kann somit die Kommunikation zwischen zwei Komponenten abhören und diesen, wenn gewünscht, auch manipulieren. Die Aspekte können für jede Komponente individuell konfiguriert werden.

8.2.5.3 Referenz

Diese Referenz umschliesst alle möglichen Kommandos, welche in der Simulationsumgebung verwendet werden können.

main : SimulationScript (Nach dem Start der Konsole befindet man sich im main Kontext.)	
createScenario nrOfBoards:*Integer* nrOfWriters:*Integer* nrOfReaders:*Integer*	Erstellt ein neues Szenario mit der angegebenen Anzahl von Boards, Writern und Readern. Diese Anzahl kann im erstellten Szenario nicht mehr verändert werden. Ebenfalls muss eine Internetverbindung zu http://security.hsr.ch/msevote/threshold bestehtn, damit die Schlüssel für das Threshold-Multisignaturverfahren heruntergeladen werden können. Die bereits heruntergeladenen Schlüssel werden im keys Ordner gespeichert. Nach dem Aufruf des Kommandos wird der Kontext auf das neu erstellte Szenario gewechselt (main#scenario)
deleteScenario scenarioName:*String*	Löscht das Szenario mit dem angegeben Namen aus dem scenarios/saved_scenarios Ordner.
listScenarios	Listet alle verfügbaren Szenarios aus dem scenarios/saved_scenarios auf.

main#scenario: ScenarioScript Nach dem Kreieren oder Laden eines Szenarios wechselt man in den scenario-Kontext.	
pauseAll	Pausiert alle laufenden Reader und Writers.
saveScenario scenarioName:String	Speichert das aktuelle Szenario im „scenarios/saved_scenarios“ Ordner mit dem angegebenen Namen.
startAllWriters	Startet alle verfügbaren Writer. Diese beginnen in ihrem angegeben Zyklus Meldungen auf die Boards zu publizieren.
Rn (n = Readernummer)	Für jeden Reader im Szenario wird ein eigener Skriptkontext angelegt um ihn zu managen. Die Reader sind von 0 bis n durchnummeriert. Nach dem Aufruf des Kommandos wird der Kontext auf den Reader gewechselt (main#scenario#Rn)
Wn (n = Writernummer)	Für jeden Writer im Szenario wird ein eigener Skriptkontext angelegt um ihn zu managen. Die Writer sind von 0 bis n durchnummeriert. Nach dem Aufruf des Kommandos wird der Kontext auf den Writer gewechselt (main#scenario#Wn)
Bn (n = Boardnummer)	Für jedes Board im Szenario wird ein eigener Skriptkontext angelegt um es zu managen. Die Boards sind von 0 bis n durchnummeriert. Nach dem Aufruf des Kommandos wird der Kontext auf das Board gewechselt (main#scenario#Bn)

main#scenario#Rn: TaskRunScript In diesem Kontext ist es möglich den Reader anzusteuern.	
aspect_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration der Aspekte zu verwalten. (main#scenario#Rn#aspect_config)
component_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration der Komponente zu verwalten. (main#scenario#Rn#component_config)
next	Führt einen Schritt für den Reader aus. Dabei werden die Einträge von allen Boards gelesen und diese geprüft. Es wird ausgegeben, wie viele korrupte Einträge für die einzelnen Boards gefunden wurden.
pause	Pausiert den Reader. D.h. die Boards werden nicht mehr geprüft.

simulation_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration für die Simulation zu verwalten. (main#scenario#Rn#simulation_config)
start	Prüft die Boards zyklisch mit der Zeit welche in der Simulationskonfiguration angegeben wurde.

main#scenario#Wn: WriterScript In diesem Kontext ist es möglich den Writer anzusteuern.	
aspect_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration der Aspekte zu verwalten. (main#scenario#Wn#aspect_config)
component_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration der Komponente zu verwalten. (main#scenario#Wn#component_config)
next	Führt einen Schritt für den Writer aus. Dabei wird versucht ein Eintrag auf die verfügbaren Boards zu schreiben.
pause	Pausiert den Writer. D.h. es werden keine Einträge mehr geschrieben.
simulation_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration für die Simulation zu verwalten. (main#scenario#Wn#simulation_config)
start	Schreibt zyklisch Einträge auf die Boards. Der Zeitabstand ist in der Simulationskonfiguration gespeichert.
publish message:*String*	Schreibt ein einzelner Eintrag mit dem gegeben Inhalt auf die Boards.

main#scenario#Bn: BoardScript In diesem Kontext ist es möglich ein Board anzusteuern.	
aspect_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration der Aspekte zu verwalten. (main#scenario#Bn#aspect_config)
component_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration der Komponente zu verwalten.

	(main#scenario#Bn#component_config)
simulation_config	Wechselt in den Kontext, in welchem es möglich ist die Konfiguration für die Simulation zu verwalten. (main#scenario#Bn#simulation_config)
list	Schreibt alle Einträge von allen Histories des Boards in die Ausgabe.

main#scenario#Bn#component_config:ComponentConfigScript

main#scenario#Rn#component_config:ComponentConfigScript

main#scenario#Wn#component_config:ComponentConfigScript

Mit diesem Kontext ist es möglich die Konfiguration für die einzelnen Komponenten zu konfigurieren.

Mehr dazu unter Komponentenkonfiguration. Nachdem die Konfiguration verändert wurde, muss das Szenario gespeichert werden und nochmals geladen werden, damit die Änderungen übernommen werden.

list	Listet alle Konfigurationswerte für die Komponente auf
set key:*String* value:*String*	Setzt den Wert für den gegebenen Schlüssel neu.

main#scenario#Rn#simulation_config:SimulationConfigScript

main#scenario#Wn#simulation_config:SimulationConfigScript

Mit diesem Kontext ist es möglich die Konfiguration für die einzelnen simulierten Komponenten zu konfigurieren.

Mehr dazu unter Simulationskonfiguration.

list	Listet alle Konfigurationswerte für die Simulation auf
set key:*String* value:*String*	Setzt den Wert für den gegebenen Schlüssel neu.

main#scenario#Bn#aspect_config: AspectConfigScript

main#scenario#Rn#aspect_config: AspectConfigScript

main#scenario#Wn#aspect_config: AspectConfigScript

Mit diesem Kontext ist es möglich die Konfiguration für die einzelnen Aspekte der simulierten Komponenten zu konfigurieren. Mehr dazu unter Aspektkonfiguration.

list	Listet alle Konfigurationswerte für die Aspekte auf
set key:*String* targetBoard:*String* value:*String*	Setzt den Wert für den gegebenen Schlüssel neu. Der Wert für den Aspekt zwischen der aktuellen Komponente und dem Board mit dem gegebenen Name.

log:LogScript Mit dem log-Kontext können Einstellungen über die Logging-Ausgabe getätigt werden. Dieser Kontext kann von irgendwoher über .mod_log aufgerufen werden. Um den Kontext wieder zu verlassen kann .leave verwendet werden.	
disableAll	Schaltet alle Logging-Meldungen auf der aktuellen Ausgabe ab.
enableAll	Schaltet alle Logging-Meldungen auf der aktuellen Ausgabe ein.
error	Wechselt in die spezifische Konfiguration für alle Fehlermeldungen. (log#error)
info	Wechselt in die spezifische Konfiguration für alle Informationsmeldungen. (log#info)
status	Gibt den Status jeder Logging-Kategorie aus.
warning	Wechselt in die spezifische Konfiguration für alle Warnungen. (log#warning)

log#error:LogCategoryScript **log#warning**:LogCategoryScript **log#info**:LogCategoryScript Dieser Kontext ermöglicht es für eine spezifische Logging-Kategorie die Konfiguration vorzunehmen.	
disable	Schaltet alle Logging-Meldungen dieser Kategorie ab.
enable	Schaltet alle Logging-Meldungen dieser Kategorie ein.
isEnabled	Zeigt an, ob die Logging-Meldungen dieser Kategorie ein- oder ausgeschaltet sind.

8.2.6 Komponentenkonfiguration

Folgende Werte können in der Konfiguration der einzelnen Komponenten verändert werden:

Für alle:

- crypto.hashAlgo Gibt den welche Hash-Funktion verwendet wird.
- crypto.signatureAlgo Gibt an, welches asymmetrische Kryptosystem verwendet wird.

Für beide Konfigurationen können die gültigen Werte können unter
http://download.oracle.com/javase/1.4.2/docs/guide/security/CryptoSpec.html#AppA nachgeschlagen werden.

Für Board:

- board.nrOfHistories Gibt an, über wie viele Histories das einzelne Board verfügt.

Für Writer

- writer.nrOfPublishAttemts Gibt an, wie viele Mal der Writer nach einen missglückten Publikationsversuch nochmals versucht wird die Meldung nochmals zu publizieren.

8.2.7 Aspektkonfiguration

Ein Schlüssel für die Aspektkonfiguration ist folgendermassen aufgebaut: [Name des Aspekt]_[Ziel]

Als Ziel kann auch ALL verwendet werden. Damit müssen nicht die Aspekte für zu allen anderen Komponenten definierte werden. Wenn für einen Aspekt kein spezifischer Konfigurationseintrag zum Ziel existiert, dann wird der Konfigurationseintrag mit ALL verwendet.

Für alle:

- networkdelay Mit dem Networkdelay-Aspekt kann die Kommunikationszeit zwischen den Komponenten verzögert werden. Ist der Wert 0, so wird der Aufruf direkt weitergegeben. Die Verzögerung geht von der aktuellen Komponente. Es wird also vor und nach dem konkreten Aufruf die angegebene Zeit gewartet.
- Stopwatch: Misst die Zeit für den konkreten Methodenaufruf und gibt die gemessen Zeit in der Ausgabe aus.

Für Board:

- partialThresholdSignature: Manipuliert die partielle Threshold Signatur des aktuellen Boards.

Für Writer:

- finalThresholdSignature: Das gegebene Board liefert eine fehlerhalft Signatur zurück.
- noWriteAfterRead: Ist dieser Aspekt auf dem Writer gesetzt, dann setzt der Writer nur einen readForAppending Anfrage aus. Anschliessend führt er jedoch die write Anfrage nicht mehr aus.
- Wronghash: Der Writer signiert die Meldung an das gegebene Board falsch.

8.2.8 Simulationskonfiguration

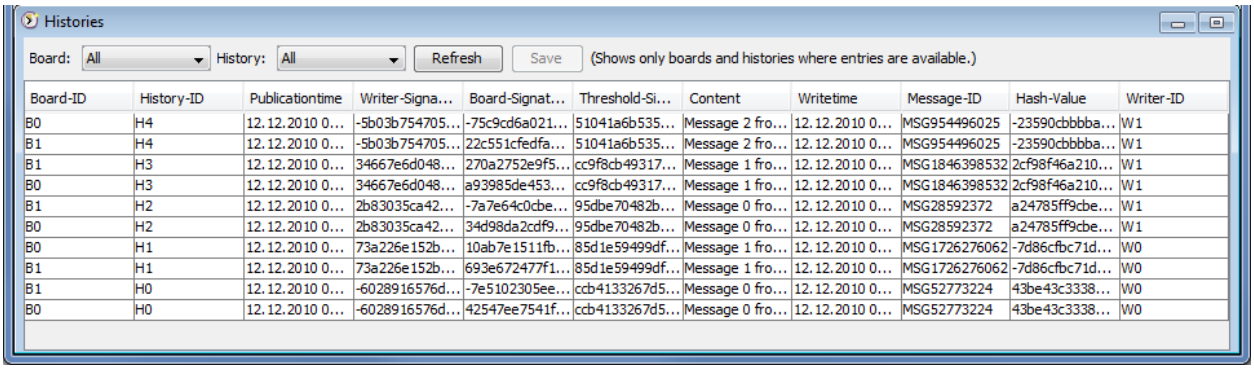
In der Simulation können für die verschieden Komponenten Operation in unterschiedlichen Zeitabständen aufgerufen. Konkret ist beim Writer das Schreiben eines Eintrags auf die Boards die Operation. Die Operation des Readers prüft zyklisch die Einträge auf den Boards.

Für Writer und Reader:

- taskRun.updateInterval: Gibt die Zeit in Millisekunden an, bis die Operation wieder ausgeführt wird.
- taskRun.useRandom: Gibt an ob, der Abstand zwischen den Operationen zufällig gewählt werden soll. Die Zeit wird dabei folgendermassen gerechnet: Zufallszahl(0 – updateInterval) + updateInterval / 2
- taskRun.useFixRate: Gibt an, ob der Abstand zwischen den Operation einer fixen Zeitspanne entspricht, oder ob dieser dynamisch gewählt wird. Wenn useFixRate auf false gesetzt, dann wird nach dem ausführen der Operation gewaret. Ansonsten wird die folgende Operation schon geplant, bevor die Operation beendet ist.

8.3 Histories

Im Histories Fenster sind alle Einträge ersichtlich, die sich in der aktuellen Datenbank befindet.



Board-ID	History-ID	Publicationtime	Writer-Signa...	Board-Signat...	Threshold-Si...	Content	Writetime	Message-ID	Hash-Value	Writer-ID
B0	H4	12.12.2010 0...	-5b03b754705...	-75c9cd6a021...	51041a6b535...	Message 2 fro...	12.12.2010 0...	MSG954496025	-23590cbbbba...	W1
B1	H4	12.12.2010 0...	-5b03b754705...	22c551cfedfa...	51041a6b535...	Message 2 fro...	12.12.2010 0...	MSG954496025	-23590cbbbba...	W1
B1	H3	12.12.2010 0...	34667e6d048...	270a2752e9f5...	cc9f8cb49317...	Message 1 fro...	12.12.2010 0...	MSG1846398532	2cf98f46a210...	W1
B0	H3	12.12.2010 0...	34667e6d048...	a93985de453...	cc9f8cb49317...	Message 1 fro...	12.12.2010 0...	MSG1846398532	2cf98f46a210...	W1
B1	H2	12.12.2010 0...	2b83035ca42...	-7a7e64c0cbe...	95dbe70482b...	Message 0 fro...	12.12.2010 0...	MSG28592372	a24785ff9cbe...	W1
B0	H2	12.12.2010 0...	2b83035ca42...	34d98da2cdf9...	95dbe70482b...	Message 0 fro...	12.12.2010 0...	MSG28592372	a24785ff9cbe...	W1
B0	H1	12.12.2010 0...	73a226e152b...	10ab7e1511fb...	85d1e59499df...	Message 1 fro...	12.12.2010 0...	MSG1726276062	-7d86cfbc71d...	W0
B1	H1	12.12.2010 0...	73a226e152b...	693e672477f1...	85d1e59499df...	Message 1 fro...	12.12.2010 0...	MSG1726276062	-7d86cfbc71d...	W0
B1	H0	12.12.2010 0...	-6028916576d...	-7e5102305ee...	ccb4133267d5...	Message 0 fro...	12.12.2010 0...	MSG52773224	43be43c3338...	W0
B0	H0	12.12.2010 0...	-6028916576d...	42547ee7541f...	ccb4133267d5...	Message 0 fro...	12.12.2010 0...	MSG52773224	43be43c3338...	W0

8.3.1 Einträge aktualisieren

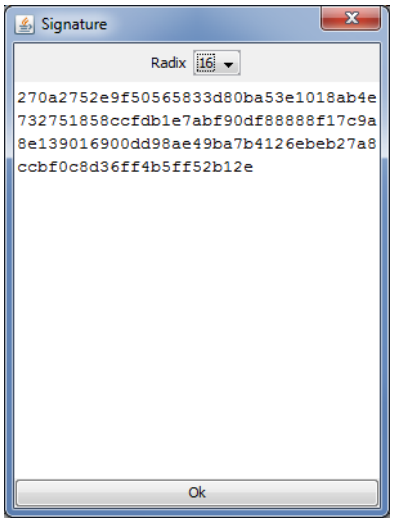
Über den “Refresh” Knopf können die Einträge von der Datenbank neu geladen werden. Dabei werden jeweils die neusten Einträge zuoberst angezeigt.

8.3.2 Filtern von Einträgen

Es gibt die Möglichkeit die dargestellten Einträge zu filtern. Dabei kann nach dem Board oder der History gefiltert werden. Um nach alle Einträge eines Boards zu betrachten, kann in der Auswahlliste neben „Board:“ der Name des gewünschten Boards ausgewählt werden. Nach dem Selektieren, wird die Liste automatisch aktualisiert. Analog dazu kann man auch die Histories filtern. Dafür kann die Auswahlliste neben „History:“ verwendet werden.

In der Auswahlliste erscheinen nur die Boards und Histories, welche auch über Einträge in der Datenbank verfügen. Die Auswahllisten werden ebenfalls beim drücken des „Refresh“-Knopfes aktualisiert.

8.3.3 Verändern von Einträgen



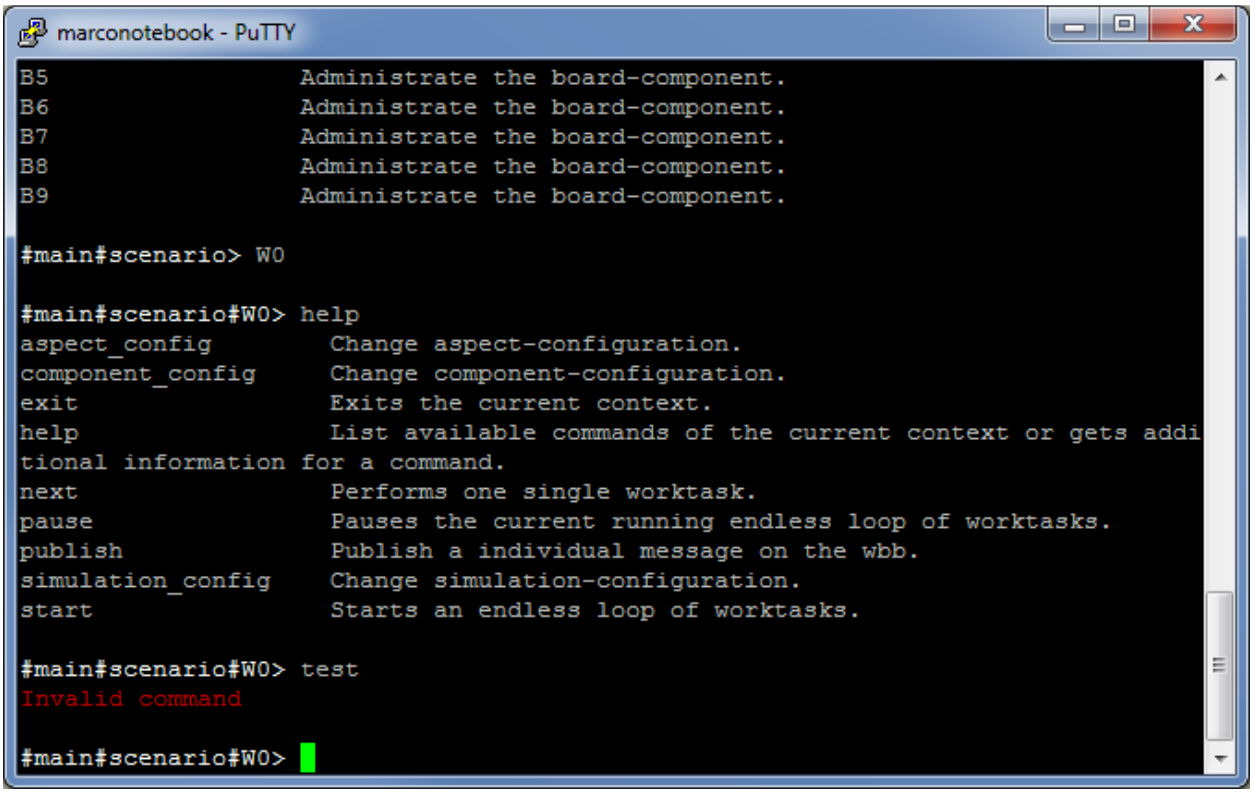
Die gespeicherten Einträge können auch verändert werden. Damit kann zum Beispiel geprüft werden, ob der Reader ein korruptes Board detektieren kann. Um Änderungen vorzunehmen können Werte in einzelnen Zellen bearbeitet werden. Dies passiert mittels einem Doppelklick in der gewünschten Zelle. Falls man einen binären Wert in verändert erscheint ein zusätzlicher Dialog, der es ermöglicht vereinfacht die Werte zu bearbeiten.

Der Dialog erlaubt es die Werte in unterschiedlichen Stellenwertsystemen zu bearbeiten. Um das Stellwertsystem zu verändern, kann in der Auswahlliste neben „Radix“ das gewünschte System ausgewählt werden. Nach dem Bearbeiten können die Änderung mit dem Drücken des „Ok“-Knopfes bestätigt werden.

Andere, nicht binäre, Werte können direkt in der Zelle editiert werden. Nach dem Bearbeiten wird der „Save“-Knopf aktiv. Mit drücken des „Save“-Knopfes werden die geänderten Daten in die Datenbank zurückgespeichert.

8.4 Zugriff per SSH

Auf die Skiptkonsole kann auch mittels eines SSH-Client Zugegriffen werden. Dafür kann unter Windows beispielsweise putty verwendet werden. Der SSH-Server läuft auf Port 4444 und darüber angesprochen werden. Für Benutzername und Passwort kann irgendein Wert verwendet werden. Nach dem Login verfügt man über die gleiche Funktionalität wie über das WBB Studio. Die gestartete Konsole verfügt ebenfalls über Autovervollständigung und eine Hilfe für die einzelnen Kommandos. Nach dem Schliessen des WBB Studio wird jedoch auch die SSH-Verbindung getrennt.



8.5 Zugriff per Webservice

Für jedes Board im laufenden Scenario werden zwei Webservices erstellt. Der eine Webservice deckt die MessagePublisher-Schnittstelle ab, welche für die Kommunikation zwischen den Boards benötigt wird. Der andere implementiert die MessageAgency-Schnittstelle welche für das Schreiben von neuen Einträgen benötigt wird.

Alle Webservices werden auf einem eigenständigen Webserver deployet. Dieser ist über den Port 4445 erreichbar. Über die Seite http://localhost:4445/axis2/services/ sind alle verfügbaren Services ersichtlich. Jeder Webservice erlaubt es Operationen im REST-Stil aufzurufen. So kann beispielsweise über http://localhost:4445/axis2/services/B0_MessageAgencyService/read alle Einträge vom Board B0 abgerufen werden.

9 Offene Punkte

Skalierbarkeit

Die wirkliche Skalierbarkeit des Systems konnte leider nicht hinreichend mit der Simulation getestet werden. Das Problem liegt daran, dass alle Komponenten eines Systems während der Simulation in einem Prozess laufen und sie sich somit die verfügbaren Ressourcen des Systems Teilen müssen. Um die Skalierbarkeit auch richtig zu testen müsste die Simulation ebenfalls verteilt ausgeführt werden können.

Zweifache Erstellung von Hash

Die Signatur und Erstellung des Hash weicht marginal von der Spezifikation von Roland Krummenacher ab. Dabei werden Signaturen, welche, gemäss Spezifikation, nur einen Hash beinhalten zweimal durch die Hashfunktionen durchlaufen. Als erstes wird ein Hash für den Ursprungstext gebildet. Danach wird dieser Hash am, in Java integrierten, Signaturprovider übergeben. Dieser Signaturprovider bildet vom binären Wert nochmals ein Hash, und signiert dann diesen. Dieses Problem konnte nicht einfach behoben werden, da man nicht vom Standard-Signaturprovider in Java abwenden wollte.

MessageWaitSet‘s werden nicht gelöscht

Um die Implementation über einen längeren Zeitraum ohne Memoryleaks betreiben zu können, müsste man die MessageWaitSet‘s gelegentlich mit einem eigenständigen Thread löschen. Dies würde ein allfälliges überlaufen des Memories verhindern. Ohne diese Modifikation könnte mittels gezieltem senden von korrupten, halbfertigen Write Aufträgen ein Speicherüberlauf erzwungen werden.

Writer blockiert

In der aktuellen Implemenation kann der Writer nicht mehrere Nachrichten parallel absetzten. Der einzelne Aufruf blockiert und wartet bis er von allen Boards eine Rückmeldung erhält. Der Writer könnte jedoch bereits nach dem Erhalt der ersten Rückmeldung die Publikation als gelungen anschauen. In der aktuellen Implementation wurde darauf verzichtet, da das Board im Kontext des Writers läuft. Würde der Writer also den Aufruf abbrechen, so wird auch der Ablauf des Boards unterbrochen.

Identifikation der Nachricht als Ursache für Probleme

Es können mehrere Probleme daraus entstehen, dass der Writer die Identifikation der Nachricht selber wählen kann. Das erste Problem kann entstehen, wenn zwei Writer zum ungefähr gleichen Zeitpunkt dieselbe Identifikation für eine Nachricht verwenden. Die zwischen den Boards ausgetauschten Meldungen beinhalten nämlich keine Angaben über den Writer, sondern nur die Identifikation der Nachricht. Wenn also ein Board bereits von einem anderen Writer eine Nachricht mit derselben Identifikation erhalten hat, so verwirft es die Meldung von einem anderen Board mit derselben Identifikation der Nachricht, welche dieses jedoch von einem anderen Writer erhalten hat. Somit bietet sich also ein möglicher Angriffspunkt an, denn kann ein Writer von einem anderen Writer die Identifikation der Nachricht während der Publikation lesen, so kann er eine Nachricht mit derselben Identifikation auf die Boards schreiben. Und somit werden beide Nachrichten nicht publiziert.

Eine andere Schwachstelle ist, dass ein Writer ein Board als korrupt erscheinen lassen kann. Dann nämlich wenn er an verschiedene Boards unterschiedliche Nachrichten jedoch mit derselben Identifikation versendet. Die Boards erstellen ihre partiellen Signaturen und verteilen diese an die übrigen Boards. Die Boards welche diese Teilsignatur erhalten stellen jedoch bei der Prüfung fest, dass die Signatur ungültig ist. Das Board kann jedoch nicht feststellen für welchen Nachrichteninhalt die Signatur erstellt wurde. Darum bekommt es das Gefühl das sein Partnerboard korrupt ist.

Component prüfen bei Methodeneintritt

Sicherheitstechnisch müsste geprüft werden, dass nur autorisierte Komponenten spezifische Methoden auf anderen Komponenten ausführen können.

Readerimplementation verbessern

Die derzeitige Implementation des Readers ist so ausgelegt, dass bei jeder Überprüfung der Boards innerhalb des Threshold-Sets alle Histories aller Boards heruntergeladen werden und diese dann geprüft werden. Dabei wäre es sinnvoller, den jeweils letztgeprüften Historyeintrag zu merken und bei einer Folgeprüfung nur jeweils inkrementell die neuen Einträge auf ihre Korrektheit zu überprüfen. Nachteil dieser Logik ist allenfalls, dass nach einer ersten fehlerfreien Prüfung die Einträge auf den Boards geändert werden.

Partielle Signaturen abspeichern

Abzuklären ist noch, ob die Speicherung der partiellen Signaturen, respektive die InterBoardPublication-Information Messages ebenfalls auf den WebBulletin-Boards abgespeichert werden müssen. Allenfalls genügen die bisher abgespeicherten Informationen (Message und an den Writer gesendete Quittung).

10 Glossar

Begriff	Bedeutung
WBB	Web-Bulletin-Board
Threshold-Set	Eine Gruppe von WBB's auf welche eine Nachricht m redundant publiziert wird.
Partielle Threshold-Signatur / Teilsignatur	Mehrere partielle Threshold-Signaturen ermöglichen die Erstellung einer verteilten Threshold-Signatur.
Verteilte Signatur	Eine RSA-Signatur welche von einem Threshold-Set ausgestellt werden kann.
Verteilte Threshold-Signatur	Eine verteilte Signatur, die nur dann zustande kommt, wenn die Anzahl WBB's die an der Signatur teilnahm mindestens dem Threshold des entsprechenden Threshold-Sets entspricht.
Threshold	Minimale Anzahl WBB's die an einer verteilten Threshold-Signatur beteiligt sein müssen, damit diese gültig ist. Die Grösse des Threshold ist zuvor für jedes Threshold-Set zu bestimmen.

11 Projektmanagement

11.1 Erfahrungsberichte

11.1.1 Marco Hofstetter

Projektverlauf

Als ich mich zu Beginn der Studienarbeit erstmals genauer mit dem ausgehändigten Paper auseinandersetzte, dachte ich anfangs, dass es sich dabei um eine reine Fleissarbeit entpuppen würde, die Ideen und das Konzept von Herrn Roland Krummenacher praktisch umzusetzen. Doch schnell bemerkte ich, dass das Thema viel mehr hergab als ich mir dies zuerst dachte. Ich fand es angenehm wie uns Herr Steffen unseren Freiraum liess und wir unsere eigenen Ideen und Vorstellungen einbringen konnten. So floss dann doch die ein andere andere Mehrstunde in eine Idee, die so eigentlich zu Anfang gar nicht vorgesehen war.

Spannend war vor allem zu bemerken wie viele verschiedene Themenbereiche durch das Thema abgedeckt wurden, wovon ich im bisherigen Studienverlauf noch nicht so viel praktisch umgesetzt habe. Sei dies der für einmal etwas andere, spannendere Kontakt mit der Kryptographie und der dahintersteckenden Mathematik oder auf der Programmierseite die Abdeckung von Multithreading, der dazugehörigen Synchronisation, und einer etwas komplexeren Softwarestruktur als dies bei einer normalen Datenhaltungs-Software der Fall gewesen wäre.

Arbeit im Team

Das Arbeiten im Team mit Marco Schälle hat aus meiner Sicht wieder einmal sehr gut funktioniert. Ich denke wir konnten die Arbeit gut und fair aufteilen. Zudem konnten wir beide die ein oder andere gute Idee einbringen oder uns gegenseitig für die Implementierung des ein oder anderen Extra-Feature animieren. Wir hatten eigentlich keine Probleme auf ein gemeinsam gestecktes Ziel hinzuarbeiten. So war für uns beide schon recht früh klar, dass wir den Hauptfokus dieser auf die Implementierung der WBB-Kernkomponenten als Library und eine dazugehörige Simulation legen würden. Dies stellte sich aus meiner Sicht auch als richtige Entscheidung heraus, war dies doch die einzig sinnvolle Art und Weise die Szenarien von Herrn Krummenacher's Arbeit nachzubilden, ohne den grössten Teil der Zeit mit dem Deployment auf die verschiedenen Server verbringen zu müssen.

Fazit

Allgemein kann man sagen, dass auch wenn wir das ausgeschriebene Ziel mit der richtig verteilten WebBulletin-Board Umgebung nicht ganz erreicht haben, wir doch eine sehr gute Ausgangslage für die auf uns wartende Bachelorarbeit haben. Der einzige Wehrmutstropfen der diese Arbeit in meinen Augen mit sich brachte war die ein bisschen fehlende Wegleitung auf ein Ziel. Dies wäre allenfalls mehr gegeben gewesen, hätte man einen Industrie- oder Pilotpartner welche klare Anforderungen stecken würde, wie das Projekt schlussendlich aussehen müsste.

11.1.2 Marco Schälle

Projektverlauf

Der Start in das Projekt war nicht allzu einfach, da wir uns mit viel neuer Thematik konfrontiert sahen. So war uns auch die mathematische Grundlage zum Verständnis der benötigen Verschlüsselungsverfahren nicht gegeben. Trotzdem konnten wir das wesentliche für die Umsetzung verstehen. Bei der Realisation kamen wir gut voran und hatten auch genug Zeit, um uns Gedanken über die Testbarkeit der Applikation Gedanken zu machen. Wir wendeten dann auch recht viel Zeit dafür auf. Das hatte auch zur Folge, dass wir nicht alles aus der ursprünglichen Ausschreibung realisieren konnten. Konkret ist davon die Kommunikation der einzelnen Komponenten über das Netzwerk betroffen. Dafür wurde eine solide Basis für die Weiterführung des Projektes gebildet.

Die Realisation hat auch zu jeder Zeit Spass gemacht, da es sich nicht um eine Applikation handelt, welche nur Daten visualisiert, sondern auch komplexere Logik beinhaltet. Man war mit nicht-alltäglichen Problemen konfrontiert, welche man ansonsten nur theoretisch aus dem Unterricht an der HSR kannte.

Arbeit im Team

Die Arbeit in unserem Team, ist nach meiner Sicht, wieder einmal sehr gut verlaufen. Wir hatten schon frühere Projekte gemeinsam realisiert und sind uns an einander gewohnt. Es fanden konstruktive Gespräche statt. Auch konnte eine homogene Codequalität durch häufige Code-Reviews über die ganze Arbeit hinweg garantiert werden. Bei Meinungsverschiedenheiten konnten wir meistens einen Konsens finden.

Fazit

Die Realisation der Arbeit machte Spass und wir kamen auch zügig vorwärts. Das Endresultat hat zwar einen weniger grossen Umfang, als dass ich zu Beginn erwartet hätte, doch bin ich sehr zufrieden und glaube an die Zukunft des Projektes. Wir haben eine Basis geschaffen, auf welcher man aufbauen kann.

11.2 Projektplan

Wochenplanung	22.09-28.09	29.09-05.10	06.10-12.10	13.10-19.10	20.10-26.10	27.10-02.11	03.11-09.11	10.11-16.11	17.11-23.11	24.11-30.11	01.12-07.12	08.12-14.12	15.12-21.12	22.12-24.12
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Einarbeitung	90													
Technische Analyse	10	10												
Implementation einfaches WBB		40	70	60										
Implementation WBB mit Persistenz														
Implementation verteiltes WBB					60	80	40			40				
Ausarbeitung Testszenarien		40					40							
Implementierung Testzenarien				20				70	70					
Unittest		10	10	10	10	10	10			40				
Implementation Netzwerk											80	60		
Aufsetzen Testumgebung												30		
Dokumentation			20	10	30	10	10	30	30	20	20	10	100	100
Milestones				M1					M2			M3		M4

Milestones (Stichtag Di = Ende Woche)

M1: Einarbeitung und technische Analyse abgeschlossen, Prototyp einfaches WBB mit mehreren Histories (Board, Writer)

M2: Implementation verteiltes WBB, inkl. Reader

M3: Implementation abgeschlossen

M4: Abgabe

12 Referenzen

- Roland Krummenacher: Umsetzung eines Web-Bulletin-Boards für E-Voting-Applikationen
- Victor Shoup: Practical Threshold Signatures
- James Heather & David Lundin: The Append-Only Web Bulletin Board

13 Anhänge

- Sourcecode / Eclipse-Project (Ordner ‚Source' auf CD)
- JavaDoc (Ordner ‚JavaDoc' auf CD)
- Axis-Library (Ordner ‚Source' auf CD)
- Simulation als ausführbare Java JAR-Datei (In ZIP-Datei ‚wbb.zip' auf CD)
- Poster und Abstract
- Unter Punkt 12 erwähnte Referenzen (Ordner ‚Referenzen' auf CD)