HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

ita Institute for Internet Technologies and Applications

Verteiltes Web Bulletin Board

Umsetzung eines robusten und vertrauenswürdigen Datenspeichers

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühlingssemester 2011

Autoren:	Marco Hofstetter Marco Schälle
Betreuer:	Prof. Dr. Andreas Steffen
Projektpartner:	ITA, Rapperswil
Experte:	Dr. Ralf Hauser
Gegenleser:	Prof. Dr. Peter Heinzmann

Datum: 16. Juni 2011

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, 16. Juni 2011

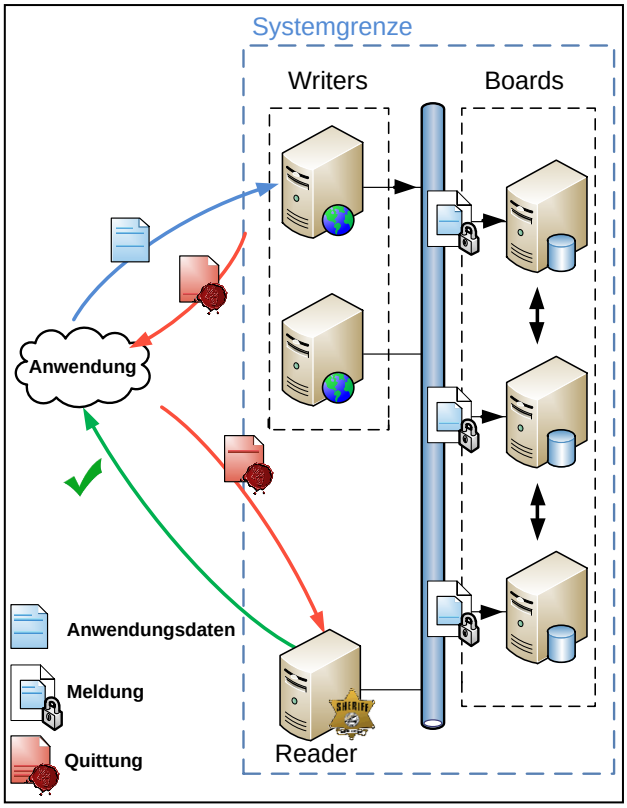
Marco Hofstetter Marco Schälle

Inhaltsverzeichnis

1 Abstract

Ausgangslage

Ein verteiltes Web Bulletin Board System wird als Ablage für Meldungen verwendet. Das System garantiert, dass keine Meldungen gelöscht oder verändert werden können, und dass die Ablage selbst keine eigenen Meldungen erstellt. Es stellt zudem sicher, dass Manipulationsversuche entdeckt werden und diese keinen negativen Einfluss auf die Funktionsweise des Systems haben. Klassische Wahl- oder Abstimmungsverfahren garantieren, dass alle Stimmen der Wähler unverändert zum Endresultat beitragen. Dabei wird ebenfalls sichergestellt, dass das Wahlsystem selbst keine Stimmen erstellt. Ein verteiltes Web Bulletin Board ermöglicht es nun, diese Anforderungen auch bei einem elektronischen Wahlverfahren abzudecken. Weitere mögliche Anwendungen für ein Web Bulletin Board wären z.B. die Ablage von Geboten bei Auktionen oder die zuverlässige Speicherung von Log-Meldungen.



Vorgehen/Technologien

Während der vorgängig verfassten Studienarbeit wurden die einzelnen Komponenten (Board, Writer und Reader) eines Web Bulletin Board Systems in Java implementiert und das Zusammenspiel in einem eigens dazu umgesetzten Simulator getestet. Im Zuge dieser Bachelorarbeit wurden die Grundlagen erarbeitet, um ein verteiltes Web Bulletin Board produktiv einsetzen zu können. Mit Hilfe einer verteilten Threshold Signatur ist es möglich sicherzustellen, dass neue Meldungen mindestens auf einer zuvor definierten Anzahl von Boards gespeichert werden. Durch einen Mehrheitsentscheid unter den einzelnen Boards wird garantiert, dass alle die identische Meldung speichern. Dem Writer wird für jede publizierte Meldung eine Quittung ausgestellt. Nachträglich kann damit geprüft werden, ob die Meldung immer noch unverändert auf dem Web Bulletin Board vorhanden ist.

Ergebnis

Die einzelnen Komponenten des Systems wurden als J2EE Applikationen umgesetzt. Eine API ermöglicht es die Funktionen des verteilten Web Bulletin Boards zu verwenden. Es können neue Meldungen publiziert, bereits publizierte Meldungen abgefragt und validiert werden. Weiter gibt es die Möglichkeit ein Web Bulletin Board System automatisiert auf einzelne J2EE Applikationsserver zu verteilen.

2 Einführung

2.1 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 webbasiertes Bulletin-Board (WBB) erreicht. Es registriert alle individuellen Wählerstimmen in verschlüsselter Form und protokolliert die Aufsummierung zum Gesamtresultat. Dabei ist es wichtig, dass sämtliche Einträge auf dem Bulletin-Board weder modifiziert noch gelöscht werden können. Das Web Bulletin Board soll verteilt ausgebildet sein, einerseits um eine hohe Verfügbarkeit zu gewährleisten, andererseits um zu vermeiden, dass einer einzelnen Instanz vertraut werden muss. Ein „Threshold Signing“ Verfahren garantiert, dass mindestens T aus N Parteien den gleichen Eintrag auf ihren Boards veröffentlichen, so dass Manipulationen der Einträge leicht erkannt werden können.

Im Rahmen dieser Bachelorarbeit soll auf der Basis der in der Studienarbeit erarbeiteten Grundlagen eine Implementation eines verteilten Bulletin Boards erstellt werden.

Aufgabenstellung

- Definition der Web Services Schnittstelle des Web Bulletin Boards.
- Implementation des verteilten Web Bulletin Boards auf der Basis einer relationalen Datenbank. Alle Meldungen sind mit einer verteilten RSA Threshold Signatur versehen.
- Erstellen eines Writers, der für den Eintrag eines Datensatzes im verteilten WBB sorgt und eine entsprechende Quittung zurückgibt.
- Erstellen einer Reader Komponente, welche die Konsistenz aller Einträge über alle Boards überprüft.

2.2 Aufbau der Arbeit

Während der vorgängig verfassten Studienarbeit wurden die einzelnen Komponenten eines verteilten Web Bulletin Boards in Java implementiert, wodurch das Zusammenspiel dieser in einem Simulator nachgestellt werden konnte. In vorliegender Bachelorarbeit werden die Grundlagen erarbeitet um ein verteiltes Web Bulletin Board produktiv einsetzen zu können. Die Komponenten des verteilten Web Bulletin Board wurden als J2EE Applikationen umgesetzt.

Zu Beginn der vorliegenden Bachelorarbeit wird nochmals kurz auf das Konzept, welches dem verteilten Web Bulletin Board zugrunde liegt, eingegangen. Dabei werden verschiedene Anwendungsfälle genannt, bei welchen der Einsatz eines verteilten Web Bulletin Board von Nutzen sein kann.

Danach folgt die Spezifikation, welche aus dem Konzept abgeleitet wurde. Sie beschreibt Funktionsweise und das Zusammenspiel der Komponenten in einem Web Bulletin Board System. Im Zuge der Bachelorarbeit wurde eine Referenzimplementation anhand dieser Spezifikation umgesetzt.

Im Kapitel „Implementation“ wird auf diese Referenzimplementation detaillierter eingegangen. Auch wird auf die einzelnen Applikationen und Module eingegangen. Dieses Kapitel soll helfen die realisierte Software zu verstehen.

Abschluss bilden die nach Abschluss der Bachelorarbeit noch offenen Punkte. Auch sind Anleitungen zu finden, welche unter anderem erklären wie man ein verteiltes Web Bulletin Board System von Grund auf aufzusetzen.

3 Einleitung / Konzept

Das Konzept des verteilten Web Bulletin Board basiert zum einen auf der Publikation „The append only WBB“ (Heather & Lundin, 2009) und zum anderen auf der Publikation „Umsetzung eines Web-Bulletin-Boards“ (Krummenacher, 2010) in welcher gewisse Themen im Detail beschrieben und Ansätze für eine praktische Implementation ausgearbeitet wurden. Ebenfalls fliessen Erkenntnisse mit ein, welche in der vorgängig verfassten Semesterarbeit „Vertrauenswürdiges und robustes Bulletin Board für E-Voting“ (Hofstetter & Schälle, 2010) gewonnen wurden.

Ein verteiltes Web Bulletin Board System besteht aus drei unterschiedlichen Komponenten – mehreren Boards, mehreren Writern und mehreren Readern. Die Idee hinter dem System des Web Bulletin Board ist, dass mehrere Writer Meldungen auf die Boards publizieren können, welche später durch die Reader wieder abgefragt werden können. Die Meldungen werden innerhalb der Boards in einer geordneten Sequenz, einer sogenannten Historie, gespeichert. Dabei werden die einzelnen Meldungen miteinander verkettet. Die Reader im System prüfen zyklisch die Verkettung der Meldungen. Die Meldungen enthalten dazu auch noch Informationen, welche für die Validierung der Meldung benötigt werden. Neue Meldungen werden immer zuoberst auf eine solche Historie abgespeichert. Es ist nicht erlaubt Meldungen dazwischen einzufügen oder bereits publizierte Meldungen zu löschen oder abzuändern.

Die Writer sind für die Erzeugung der Meldung verantwortlich. Dabei versehen sie die Meldung mit einer persönlichen Signatur, um zu beweisen, dass sie die Meldung erstellt haben. Das Board wiederum ist verantwortlich die Signaturen der Meldungen vor dem Abspeichern zu überprüfen und sicherzustellen, dass die Meldungen im Nachhinein nicht mehr verändert werden. Dabei ist zu beachten, dass der Inhalt der Meldung nicht verschlüsselt ist und somit von jeder Komponente gelesen werden kann.

Solange die Meldungen nur auf einem Board abgespeichert werden würden, würde die Möglichkeit bestehen, Meldungen bei einem Systemausfall zu verlieren. Das ganze System wäre abhängig von diesem einen Board.

Um dies zu verhindern, besteht das verteilte Web Bulletin Board aus mehreren Boards, einem sogenannten Threshold-Set, auf welchen die Meldungen vom Writer parallel publiziert werden. Diese Boards können von unterschiedlichen unabhängigen Parteien betrieben werden. Nebst dieser reinen Replizierung der Meldungen auf mehrere Boards basiert das verteilte Web Bulletin Board zusätzlich noch auf einer verteilten Signatur, einer sogenannten Threshold Signatur, welche nur durch Zusammenarbeit mehrerer Boards im Threshold-Set erstellt werden kann. Eine gültige Threshold Signatur wird für jedes erfolgreiche Publizieren einer Meldung benötigt.

Alle auf dem verteilten Web Bulletin Board gespeicherten Meldungen sind öffentlich zugänglich, wodurch jedermann einen Reader stellen kann und die Histories mit allen Meldungen auslesen kann. Somit kann jeder die Signaturen und Hashwerte der Meldungen prüfen. Dies macht es für das einzelne Board unmöglich, neue Meldungen zu erstellen oder eine gespeicherte Meldung zu löschen/ändern – ohne dass dies durch einen Reader festgestellt werden würde.

Im Folgenden sind einzelne Anwendungsfälle aufgeführt, für welche der Einsatz eines verteilten Web Bulletin Boards sinnvoll ist und von Nutzen sein kann.

E-Voting Klassische Wahl- oder Abstimmungsverfahren garantieren, dass alle Stimmen der Wähler unverändert zum Endresultat beitragen. Dabei wird ebenfalls sichergestellt, dass das Wahlsystem selbst keine Stimmen erstellt. Ein verteiltes Web Bulletin Board ermöglicht es, diese Anforderungen auch bei einem elektronischen Wahlverfahren abzudecken.

Auktionen Bei einer Auktion wünscht sich der Bietende ein Gebot auf ein Angebot abgeben zu können und sein Gebot später auch auf seine Gültigkeit überprüfen zu können. Im

Gegenzug will sich das Auktionshaus vor Behauptungen schützen, es würde die Auktion manipulieren. Dies ist gewährleistet, da die Korrektheit der Daten von jedem überprüft werden können.

System Logs Um zu verhindern, dass Logeinträge in einem sicherheitsrelevanten Bereich gelöscht oder manipuliert werden, kann ebenfalls ein verteiltes Web Bulletin Board eingesetzt werden.

3.1 Anforderungen

Zusammengefasst muss ein robustes Web Bulletin Board nach Heather & Lundin folgende Anforderungen erfüllen.

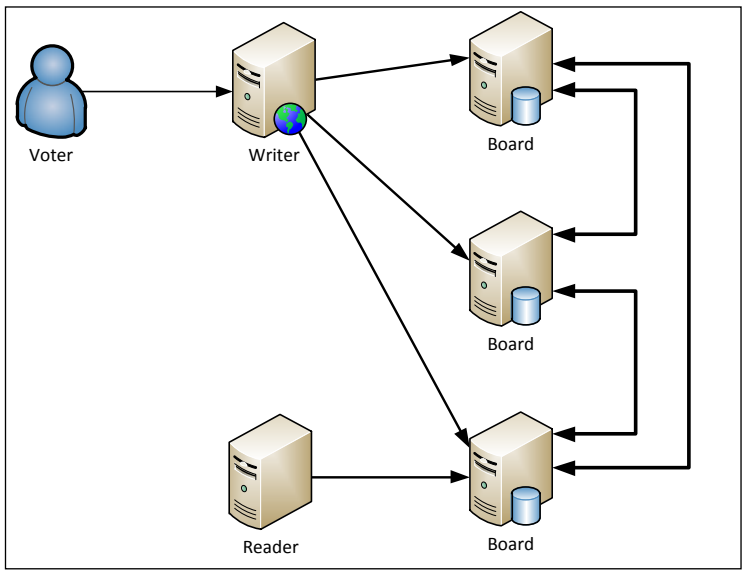
- Eine publizierte Meldung darf im Nachhinein nicht mehr editiert werden.
- Eine publizierte Meldung darf im Nachhinein nicht mehr aus der Historie entfernt werden.
- Neue Meldungen können nur an die bestehende Historie des Boards angehängt und nicht dazwischen eingefügt werden.
- Das Board ist öffentlich,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

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

- Eine Meldung muss parallel auf mehrere Boards publiziert werden.
- Korrupte Boards dürfen keinen negativen Einfluss auf den Betrieb des verteilten Web Bulletin Boards haben.

3.2 Lösungsvariante

Basierend auf den Überlegungen von R. Krummenacher entschied man sich die Variante „Asynchron verteiltes Web Bulletin Board“ mit mehreren Histories pro Board umzusetzen.



Bei dieser Variante wird vorgesehen, dass ein Writer seine Meldung unabhängig auf alle Boards innerhalb seines dazugehörigen Threshold-Sets publiziert. Zur Sicherstellung, dass die Meldung auch auf genügend vielen Boards publiziert wurde, benötigt der Writer die gültige Threshold-Signatur von mindestens einem Board. Damit ein Board wiederum eine gültige Threshold-Signatur ausstellen kann, benötigt es die Bestätigung von mindestens k-1 (k entspricht dem Threshold, weiteres dazu im folgenden Kapitel) anderen Boards, dass diese die gleiche Meldung ebenfalls publizieren werden. Somit kann der Writer mit nur einer korrekt erhaltenen Threshold-Signatur sicher sein, dass seine Meldung auf genügend Boards publiziert wurde.

Hat das einzelne Board genügend Bestätigungen von anderen Boards wird die Meldung abgespeichert und eine dazugehörige Quittung mit der Threshold-Signatur an den Writer zurückgesendet.

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

3.3 Practical Threshold Signatures nach Shoup

Die Arbeit von R. Krummenacher über das verteilte Web Bulletin Board setzt ein Threshold-Signatur-Verfahren voraus. Die aktuelle Arbeit enthält eine Implementation des Threshold-Signatur-Verfahren wie es von Victor Shoup in der Publikation „Practical Threshold Signatures“ beschrieben wurde.

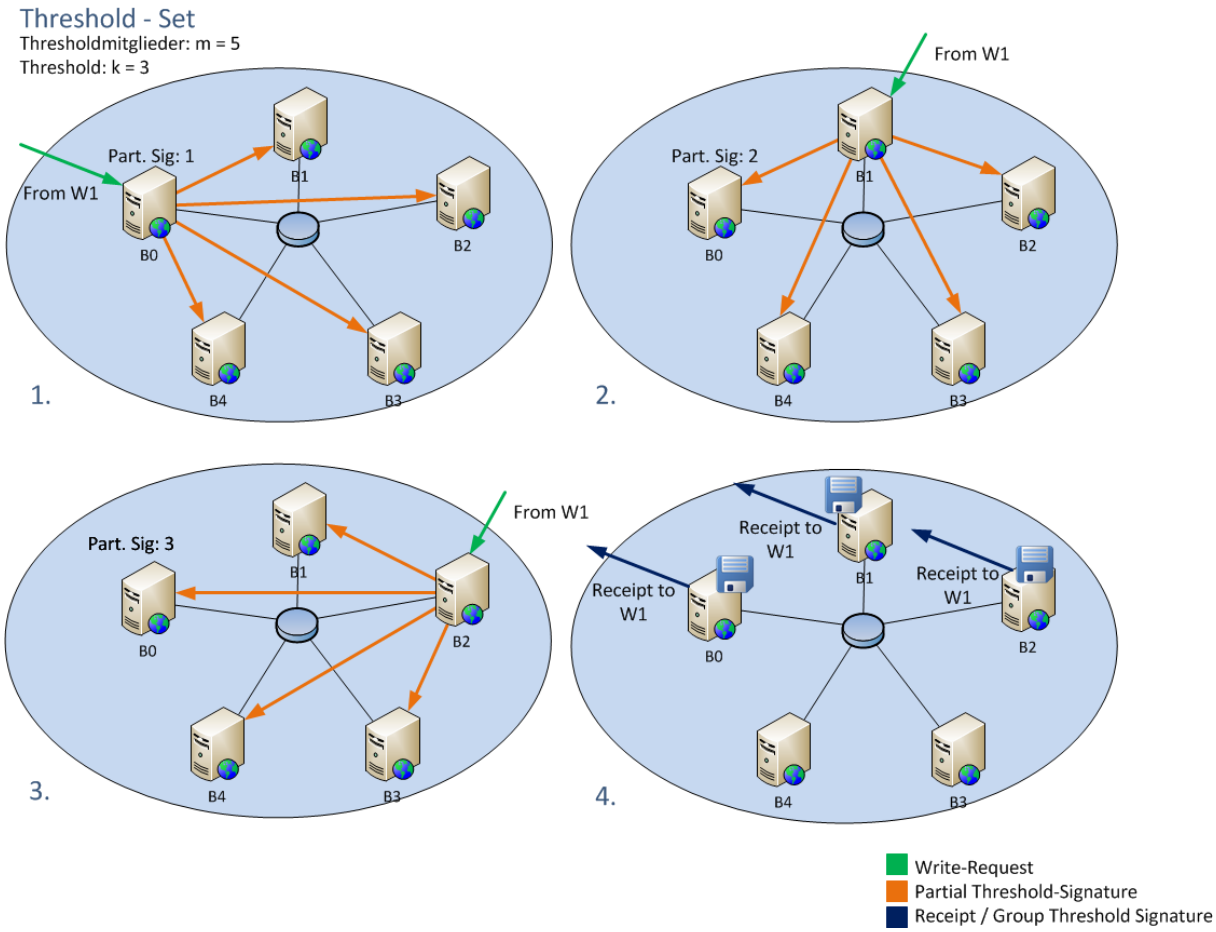
Dieses erlaubt es eine gültige verteilte RSA Signatur zu erstellen, wenn genügend Parteien eine Teilsignatur ausgestellt haben. Diese Teilsignatur kann von jeder Partei mittels Zero-Knowledge-Proof überprüft werden. Dies ist erforderlich, um die einzelnen Teilsignaturen zu prüfen, bevor die verteilte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

Die erstellte Threshold-Signatur beweist, dass mindestens eine Anzahl Parteien grösser oder gleich dem Threshold (Schwellenwert) an der Erstellung der Threshold-Signatur teilgenommen haben. Die Validierung der verteilten Threshold-Signatur geschieht danach analog der Validierung einer normalen RSA-Signatur, wobei der entsprechende öffentliche RSA Schlüssel 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.

Anwendung der Threshold Signatur auf das verteilte Web Bulletin Board



In der obigen Illustration will ein Writer W1 eine Meldung innerhalb des Threshold-Sets publizieren. Das Threshold-Set besteht aus 5 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 der Meldung verteilt das Board jeweils seine dazugehörige partielle Threshold-Signatur innerhalb des Threshold-Sets an die anderen Boards. (Abb. 1-3)

Nach Erreichen des Threshold in Abbildung 3, ist das 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 Meldung als Quittung auszustellen (Abb. 4). Die Meldung kann bei Erreichen des Threshold zudem auf der Historie abgespeichert werden.

In dieser Illustration könnten 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 kann vom jeweiligen Empfänger mittels Zero-Knowledge-Proof detektiert werden.

4 Spezifikation

Im diesem Abschnitt werden die Komponenten des verteilten Web Bulletin Board detailliert beschrieben. Die Spezifikation kann verwendet werden um eine eigene Implementation der Komponenten umzusetzen. Bei der Beschreibung der Spezifikation wurde deshalb darauf geachtet, dass keine Abhängigkeiten von einer Plattform entstehen. Im späteren werden die zwei Grundfunktionen welche ein WBB-System unterstützt beschrieben. Zum einen die Publikation und zum anderen die Endauswertung.

Bemerkungen zur Referenzimplementation sind mit einem gelben Rand gekennzeichnet. Dabei ist zu bemerken, dass genauere Informationen zur Detail-Implementation der einzelnen WBB Komponenten der Studienarbeit entnommen werden können. Besonderheiten, welche bei einer eigenen Implementation berücksichtigt werden sollten, sind ebenfalls speziell gekennzeichnet.

> Beispiel für eine Bemerkung, welche die Referenzimplementation betrifft.

> Beispiel für eine Bemerkung, welche eine Implementation im Allgemeinen betrifft.

Service-Operationen und in der Referenzimplementation und verwendete Klassen sind ebenfalls speziell hervorgehoben.

Beispiel für eine Operation oder eine Klasse: `getId` oder `CryptoBuilder`

4.1 Rollen

In Zusammenhang mit dem verteilten Web Bulletin Board treten verschiedene Rollen auf, welche durch eine natürliche Person oder Organisation (Firma, Partei, …) eingenommen werden können. Um die Sicherheit eines Systems zu gewährleisten sollte von jedem Organ nur eine Rolle eingenommen werden.

> In der Referenzimplementation wurde nicht explizit auf eine Trennung dieser Rollen geachtet. So übernimmt das automatisiert Deployment (Seite 50) die Rollen des Dealer, Assembler und Deployer.

Deployer Der Deployer konfiguriert eine einzelne Komponente. Er erstellt die Schlüssel, welche für die Signaturen benötigt werden und teilt diese dem Assembler mit. Vom Assembler erhält er die nötigen Artefakte, welche er in die Konfiguration der Komponente einbindet. Die fertig konfigurierte Komponente wird an den Host übergeben. Vom Dealer erhält er den privaten Schlüssel um seine partielle Threshold-Signatur zu erstellen.

Vendor Der Vendor implementiert eine Komponente gemäss dieser Spezifikation. Die Lösung übergibt er danach dem Deployer, welcher die Komponente für seine Umgebung konfiguriert. Der Vendor ist dafür verantwortlich, dass die Spezifikation eingehalten wird.

> Die Ersteller der Referenzimplementation treten als Vendor einer Reader-, Writer- und Board-Komponente auf.

Dealer Die Organisation, welche den Dealer übernimmt, muss unbedingt nur diese Rolle einnehmen und keine weitere mehr. Der Dealer erhält vom Assembler die Informationen über die Komponenten. Danach erstellt er die benötigten Schlüssel für die Threshold-Signaturen. Die privaten Schlüssel teilt er den Deployer der Boards über einen sicheren Kanal mit. Der öffentliche Schlüssel des Threshold-Sets wird dem Assembler mitgeteilt.

> Die Schlüssel für die Threshold-Signaturen werden im Deployment über http://security.hsr.ch/msevote/threshold abgefragt.

Host Für den laufenden Betrieb einer Komponente ist der Host verantwortlich. In seinen Bereich fällt, dass die Komponente von allen anderen Komponenten und Clients erreichbar ist. Ebenso, dass die Antwortzeiten in einem vernünftigen Bereich liegen.

Assembler Der Assembler bestimmt, welche Hosts für Writer- und Board-Komponenten in das System einbezogen werden. Er definiert auch das verwendeten Signaturverfahren und den Threshold welcher für die Publikation verwendet wird.

Der Assembler übernimmt die Aufgabe des Vermittlers zwischen den Boards. Er definiert auch die Anforderungen an ein Board um in das System aufgenommen zu werden.

Ombudsstelle Falls es zwischen den verschiedenen Komponenten zu Konflikten kommt, ist es die Aufgabe der Ombudsstelle, Lösungen für den Konflikt zu finden. Jede Komponente hat das Recht sich bei dieser Stelle zu melden, falls es eine andere Komponente verdächtigt, sich nicht wie vorgeschrieben zu verhalten. Die Beschreibung der Ombudsstelle ist nicht Teil dieser Spezifikation. Man kann sich jedoch vorstellen, dass die Ombudsstelle ebenfalls verteilt realisiert wird. Und durch Mehrheitsentscheide den eventuellen Ausschluss einer Komponente beschlossen wird.

4.2 Artefakte

Folgende Artefakte werden für den Betrieb eines verteilten Web Bulletin Boards Systems benötigt. Die Artefakte müssen zwischen den verschieden Komponenten ausgetauscht werden. Sie werden vom Assembler erstellt und an die verantwortlichen Deployer der Komponenten übergeben.

Binding-Konfiguration Die Binding-Konfiguration beschreibt, wie die Services eines Boards erreichbar sind. Diese Konfiguration wird vom Assembler mit den erhaltenen Informationen der einzelnen Board-Deployer zusammengestellt. Nach dem Zusammentragen der Binding-Konfiguration wird diese an alle Deployer weitergereicht.

> Die Binding-Konfiguration ist für den Board-API-Client in der *connections.properties* Datei enthalten. Dabei handelt es sich um eine einfache Property-Datei welche als Schlüssel die Identifikation des Boards enthält und als Wert den Basis-Url für die Services.
>
> In den EJB-Applikationen für den Writer und das Board ist der Inhalt dieser Datei in der *application.xml* Konfiguration eingebettet. Zu finden ist der Inhalt als Umgebungseintrag (env-entry) mit dem Schlüssel *connections*.

Key-Konfiguration In der Key-Konfiguration sind alle öffentlichen Schlüssen aller Komponenten enthalten, welche für die Validierung der Signaturen verwendet werden. Die Key-Konfiguration wird ebenfalls vom Assembler zusammengestellt. Die Board-Deployer und Writer-Deployer liefern dafür ihre öffentlichen Schlüssel. Der Dealer liefert noch zusätzlich den öffentlich Schlüssel des Threshold-Sets. Die gesamte Key-Konfiguration wird dann wieder an alle Deployer

ausgeliefert.

> Die Key-Konfiguration ist für den Board-API-Client in der *keyFile.ini* Datei enthalten. Dabei handelt es sich um eine einfache Property-Datei, welche hierarchisch alle Schlüssel des Systems, inklusive der eigenen privaten Schlüssel, enthält.
>
> In den EJB-Applikationen für den Writer und das Board ist der Inhalt dieser Datei in der *application.xml* Konfiguration eingebettet. Zu finden ist der Inhalt als Umgebungseintrag (env-entry) mit dem Schlüssel *keys*.

Content-Typen-Definition

Die Definition für die Content-Typen wird ebenfalls vom Assembler zusammengestellt und an die Writer- und Board-Deployer übergeben.

Die Content-Typen-Definition wird mithilfe mehrerer XML-Schemas (xsd) ausgetauscht. Dabei werden nur komplexe Elemente mit Attributen unterstützt. Mehr zum Aufbau eines Schemas ist im Kapitel 4.5.1 „Content-Typen“ (Seite 23) vermerkt.

> Die Schemas für die Content-Typen können im Deployment-Skript angegeben werden. Danach werden alle Schemas in eine Zip-Datei (schemas.zip) gepackt. Dieses Zip wird in den META-INF Ordner des DynamicEntities-jar gepackt. Somit ist es der Board-Applikation möglich während der Laufzeit alle Schemas aus der Zip-Datei zu lesen.
>
> Mehr dazu unter 5.6 Content-Typen (Seite 42)

4.3 Komponenten

Komponenten in einem WBB-System sind Anwendungen, welche unabhängig voneinander sind. Sie werden im Normalfall auf unterschiedlichen Hosts betrieben und können auch von verschiedenen Vendor‘s entwickelt worden sein.

4.3.1 Writer

Der Writer muss im WBB-System allen anderen Komponenten bekannt sein. Er kann Meldungen in einem WBB-System erstellen. Alle anderen Komponenten dürfen keine Meldungen erstellen. Der Writer kommuniziert primär mit allen Boards im System. Die Endanwendung kann einen Writer wählen, um Meldungen zu publizieren. Der Writer ist verpflichtet der Endanwendung nach erfolgreicher Publikation eine Quittung auszustellen. Die Quittung kann die Endanwendung einem beliebigen Reader zur Prüfung übergeben. Damit kann sie prüfen, dass der Writer seinen Auftrag korrekt erfüllt hat.

4.3.2 Reader

Die Aufgabe einer Reader Komponente ist die Prüfung der Boards. Die Reader Komponente kann eine externe Vertrauensstellung für die Endanwendung sein. Der Reader kann prüfen, ob die Verknüpfung der Meldungen auf einer Historie immer noch gültig ist. Damit kann verhindert werden, dass a) eine Meldung aus der Verkettung gelöscht wurde, b) die Meldung nicht verändert wurde und c) das Board selber keine Meldungen auf die Historie publiziert hat.

4.3.3 Board

Die Boards fungieren als Datenspeicher in einem WBB System. Das Board nimmt die Meldung eines Writer entgegen. Danach teilt es den anderen Boards den Erhalt der Meldung mit. Erhält es genügend Rückmeldungen (mit der partiellen Threshold Signatur versehen) von anderen Boards,

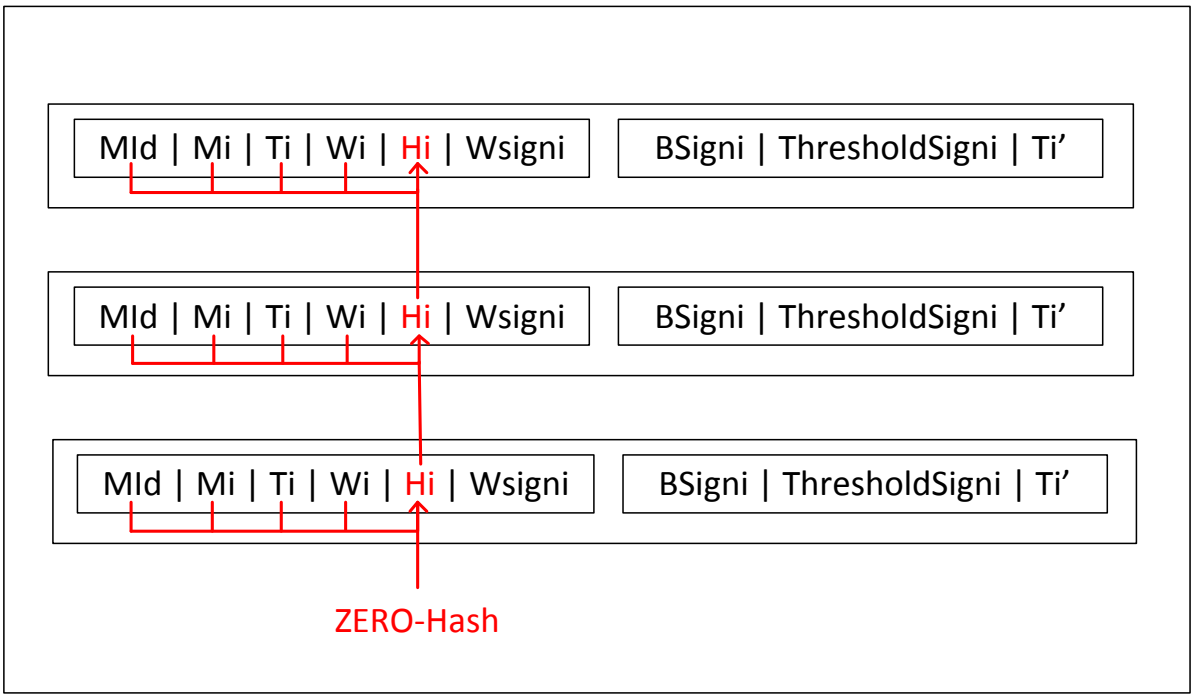
dass sie die Meldung ebenfalls Publizieren werden, kann es eine Threshold Signatur erstellen und die Meldung ablegen.

Ein Board muss über zwei Web Services verfügen dessen Endpunkte es dem Assembler für die Binding-Konfiguration mitteilen muss. Für einen Writer und Reader erscheint das Board als *MessageAgency* - kann neue Meldungen entgegennehmen und die bereits publizierten zurückliefern. Zwischen den Boards tritt das einzelne Board als *MessagePublisher* auf. Dieser Service kann vom Board-Host auch vor dem Writer versteckt werden. Über diesen Service wird zwischen den Boards ausgehandelt, ob eine Meldung publiziert wird oder nicht.

Minimal müssen vom Board-Vendor die beiden Web Services implementiert werden, welche in den WSDL-Dateien beschrieben sind. Dabei muss es sich um einen Web Service handeln, welcher über SOAP kommuniziert. Die Services sind detaillierter im Kapitel „Board-Services“ (Seite 15) beschrieben.

Historie

Um mehrere Schreibanfragen von Writern entgegenzunehmen kann das Board mehrere Histories zum Abspeichern von Meldungen führen. Dabei werden nebst der eigentlichen Meldung noch weitere Informationen abgespeichert, welche zur Validierung der dazugehörigen Meldung gebraucht werden. Die einzelnen Meldungen werden zudem miteinander verknüpft, was dazu führt, dass keine Meldungen aus einer Historie gelöscht und keine dazwischen eingefügt werden können. Dazu fliesst der Hash jeder Meldung in die Bildung der nachfolgenden Meldung ein. Diese Verkettung muss vom Writer übernommen werden. Dazu wird dem Writer vom Board vor der Publikation eine freie Historie und der Hash der neusten Meldung übergeben.



Bezeichnung	Beschreibung
Mid	Identifikation der Meldung, Von Writer definiert (Systemweit eindeutig)
Mi	Meldung i, von Writer verfasst.
Ti	Zeitpunkt der Verfassung der Meldung Mi durch den Writer.

Wi	Identifikation des Writers der Meldung Mi.
Hi	Hashwert von H(Mi, Ti, Wi, Hi-1), wobei H0 = 0.
WSigni	Signatur des Writers Wi mit dessen privaten Schlüssel über den Hash Hi.
BSigni	Signatur des Boards mit dessen privaten Schlüssel über (WSigni, Ti´).
ThresholdSigni	Threshold-Signatur über die Meldung Mi. Wird später noch im Detail behandelt.
Ti´	Zeitpunkt der Signatur durch das Board.

Verwendete Histories

Das Board ist verpflichtet die Identifikationen all seiner Histories zurückzuliefern. Es liefert eine einfache Liste als Resultat der `getHistories` Operation. Das Board kann belangt werden, wenn es nicht alle verwendeten Histories zurückliefert. Dies kann mit den ausgestellten Quittungen oder Antworten auf eine `readForAppending` Anfrage eines Boards festgestellt werden. Die Quittungen werden als Antwort auf die `write` Operation geliefert. In dieser ist die Identifikation der Historie enthalten, auf welcher die Meldung publiziert wurde. Ebenso bei der Antwort der `readForAppending` Operation. Beide Antworten werden vom Board signiert und können somit als Beweis verwendet werden. Der Reader kann also aus allen erhaltenen Quittungen alle vorhanden Histories zusammenziehen. Dieses Resultat kann der danach mit dem Resultat aus der `getHistories` Operation gegenprüfen. Falls diese nicht übereinstimmen, kann er es der Ombudstelle melden.

4.3.4 Identifikation der Komponente

Die Board und Writer Komponenten müssen im WBB-System eindeutig identifiziert werden können. Die Identifikation passiert über eine ID vom Typ *String*. Der Assembler des WBB-Systems ist dafür verantwortlich, dass jede Komponente eine eindeutige Identifikation erhält. Die Identifikation wird verwendet, um die richtigen Schlüssel für die Überprüfung der Signaturen zu finden. Der Reader benötigt deshalb auch keine Identifikation, da er keine Signaturen bildet.

Die Identifikation der Komponenten wird ebenfalls in den beiden Artefakten Binding-Konfiguration und Key-Konfiguration verwendet.

Über die `getId` Operation liefert das Board seine Identifikation innerhalb des WBB-Systems. Die Implementation dieser Operation ist optional und sollte weder vom Writer noch vom Reader verwendet werden. Beide sollten die Identifikation der einzelnen Boards bereits kennen.

> Alle Komponenten verwenden die Identifikation der anderen Komponenten aus der Key-Konfiguration (*keys.config*). Auch die einzelnen Komponenten selbst lesen ihre eigene Identifikation aus der Key-Konfiguration. Und zwar aus der privaten Abschnitt.

4.4 Board-Services

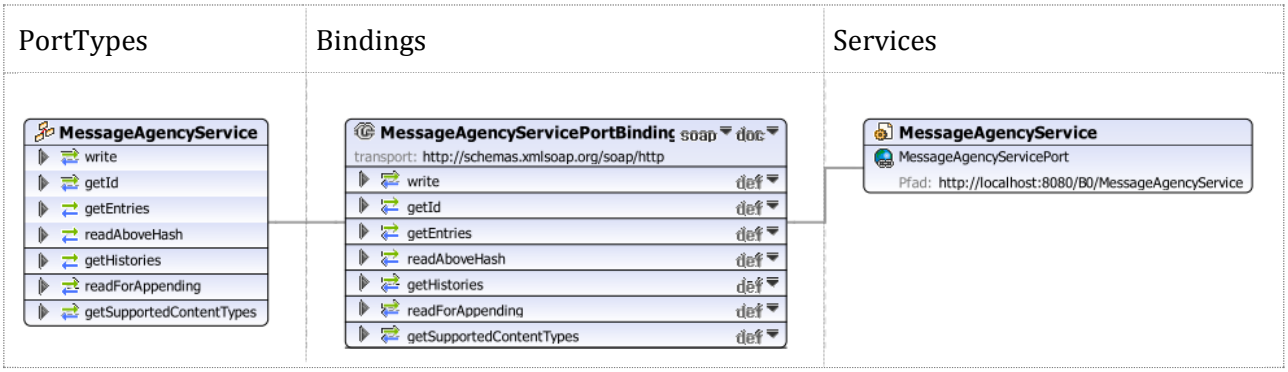
Vom Board müssen zwei Web Services implementiert werden. Sie sind im Detail in den WSDL-Dokumenten beschrieben.

Zum einen das MessageAgencyService.wsdl, welches den MessageAgencyService PortType und das entsprechende SOAP-Binding beinhaltet. Der definierte Service wird für die Kommunikation zwischen dem Writer und einem Board verwendet.

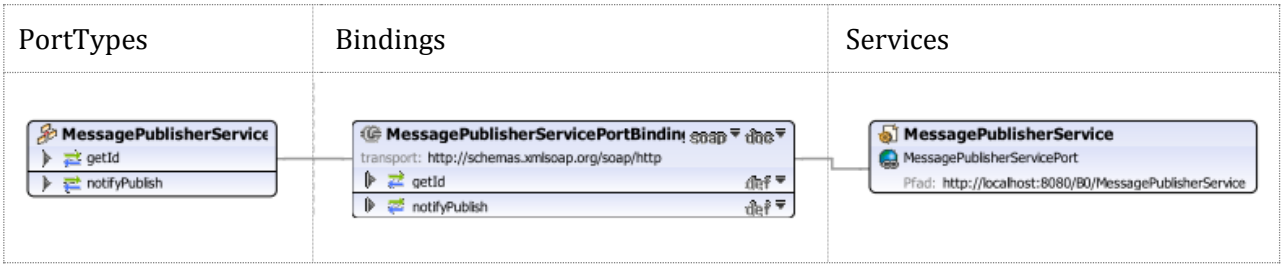
Der zweite Service ist jener, welcher durch das MessagePublisherService.wsdl Dokument beschrieben ist. Dieser Service wir für die Kommunikation zwischen den Boards benötigt.

> Die wsdl-Dateien sind im Verzeichnis WebContent/WEB-INF/wsdl des BoardWeb Projektes zu finden.

4.4.1 MessageAgency



4.4.2 MessagePublisher



4.4.3 Namespaces in wsdl-Dokument

Für die Kommunikation werden die XML-Schemas aus folgenden Namespaces benötigt:

Namespace	Prefix	Beschreibung
http://wbb.hsr.ch	wbb	Beinhaltet die Elemente welche von den wsdl-Dateien benötigt werden. Die Elemente verweisen auf die Typen in den anderen Namespaces.
http://messages.hsr.ch	messages	Typen, welche für die grundlegende Kommunikation benötigt werden.
http://queries.hsr.ch	queries	Alle Typen, welche einen Query Operator oder Bedingung repräsentieren.

http://crypto.wbb.hsr.ch	crypto	Die Typen für den Austausch der Signatur- und Hash-Werte.
http://www.w3.org/2001/XMLSchema	xs	Grundlegende Typen und Elemente einer XML-Datei, wie z.B. string.

In den späteren Beschreibungen der Typen und Namespaces wird jeweils nur der Prefix angegeben. Detaillierte Beschreibungen der Typen und Elemente der Namespaces sind in den jeweiligen XML-Schemas vorhanden.

4.4.4 Signaturen

Die Signaturen werden mittels dem RSA Signaturverfahren erstellt. Der dafür benötigte Hash wird wie im Folgenden beschrieben zusammengesetzt.

In der Spezifikation der Nachrichten sind die Parameter in der richtigen Reihenfolge (z.B. {writerId, messageId}) angegeben. Es wird ein SHA-256 Hash über das byte Array erstellt. Die zuerst folgenden Parameter sind am Anfang des byte Array.

Parameter1	Parameter2	Parameter3	Parameter4

Für die Codierung gelten folgende Regeln:

Typ	Codierung
xs:string	UTF-8 kodierter String
crypto:Signatur	Byte-Array in Big-Endian Byte Reihenfolge. Das höchste signifikante Byte ist das nullte Element. Die Anzahl Bytes entspricht der Anzahl resultierender Bytes der Signaturfunktion.
xs:dateTime	Millisekunden seit dem 1. Januar 1970, 00:00:00 GMT. Der resultierende Long wird als 8 Byte repräsentiert mit dem höchsten signifikanten Byte zuerst.
crypto:Hash	Byte-Array in Big-Endian Byte Reihenfolge. Das höchste signifikante Byte ist das nullte Element. Die Anzahl Bytes entspricht der Anzahl resultierender Bytes der Hashfunktion.

> Der korrekte Aufbau und Codierung der entsprechenden Datentypen ist im `CryptoBuilder` enthalten. Dafür werden die Java-Standard-Implementationen für die kryptographischen Verfahren verwendet.

4.4.5 Threshold-Signatur

Für die Implementation der Threshold-Signatur muss das von Victor Shoup beschriebene Verfahren verwendet werden. Es muss das erste beschriebene Protokoll aus der Arbeit („Practical Threshold Signatures“) implementiert werden. Der zu signierende Inhalt durchläuft zuerst die SHA-256 Hashfunktion. Danach wird die ASN.1 Repräsentation des Resultats mit DER kodiert. Als letztes wird noch ein PKCS1 Padding erstellt.

> Die ASN1 Sequenz für den Hash muss in der Applikation selber erstellt werden und danach an den Crypto-Provider von Java übergeben werden. Damit konnte auch erreicht werden, dass die Thresholdsignatur mit dem normalen RSA-Signatur-Provider geprüft werden kann.

4.4.6 Nachrichten

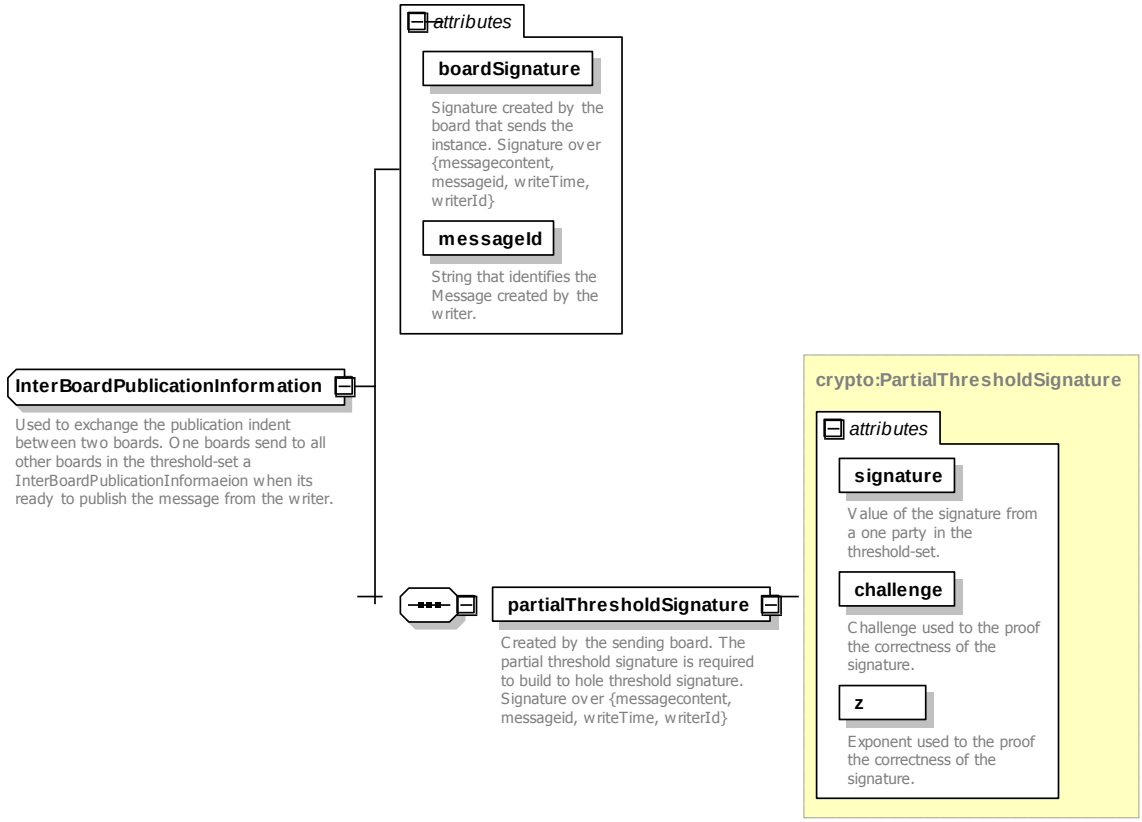
Im Folgenden ist der Aufbau der Nachrichten beschrieben, welche über die beiden Services ausgetauscht werden. Alle Attribute der Nachrichten müssen einen Wert haben. Die Nachrichten sind einem Service zugordnet und werden dort in einer Operation verwendet. Die „Richtung“ gibt an, ob die Nachricht als Input verwendet wird, oder ob die Nachricht das Resultat der Operation ist.

In den Kommentaren zu den Typen, Elementen und Attributen ist beschrieben, welche Werte zulässig sind. Ebenso, welche Werte in die Signaturen einfliessen müssen. Falls eine Komponente diese Spezifikation nicht einhält, dann kann von einer Verletzung der Spezifikation gesprochen werden.

> Alle Verletzungen der Spezifikation werden durch einen `ContractViolationException` ausgedrückt. Für die Prüfung der Spezifikation existiert zu jeder Nachricht ein entsprechender `Contract`.

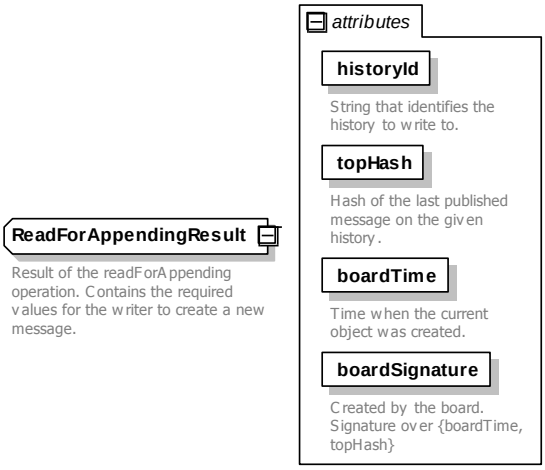
InterBoardPublicationInformation

Service	MessagePublisher
Operation	notifyPublish
Richtung	In



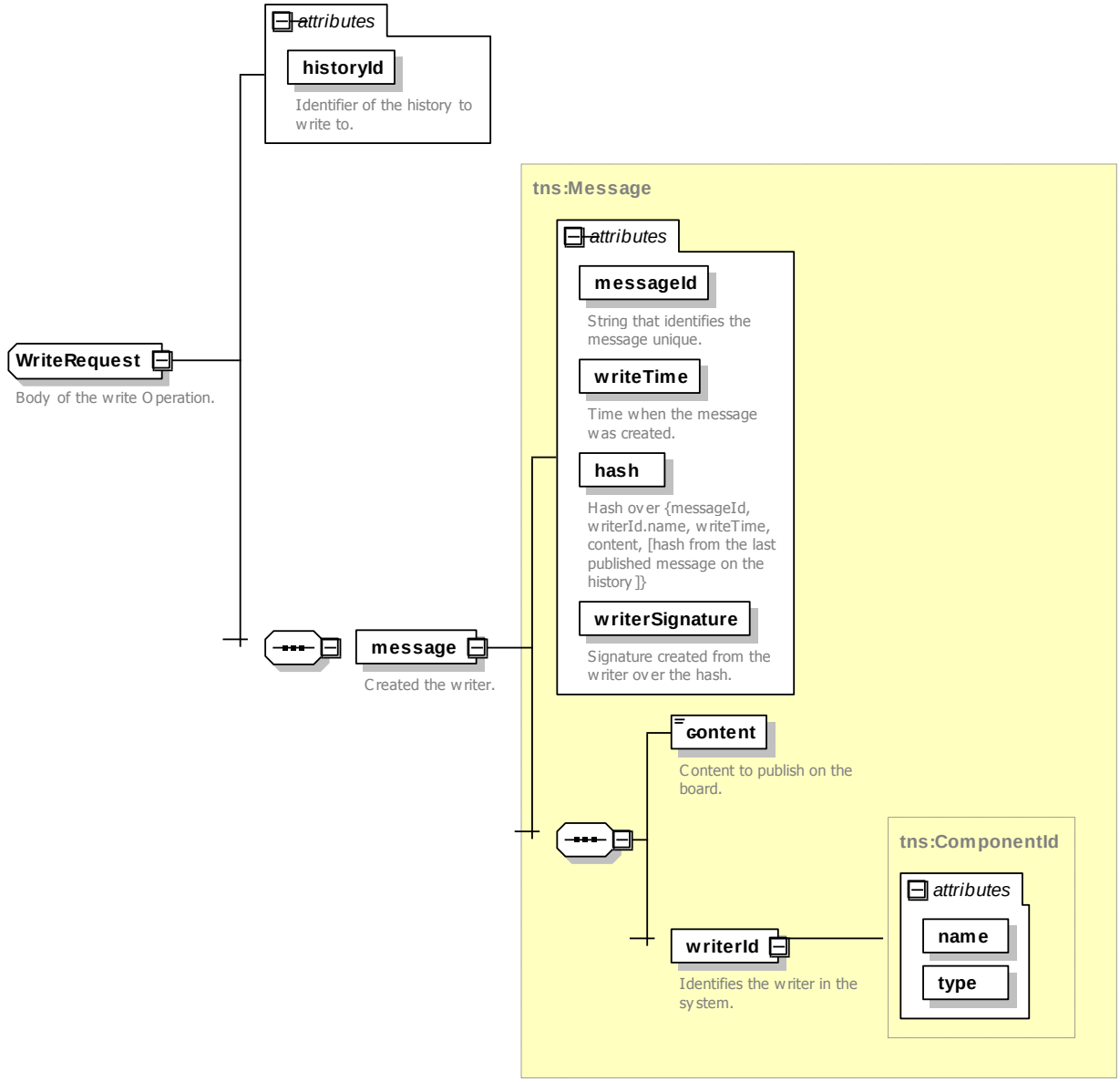
ReadForAppendingResult

Service	MessageAgency
Operation	readForAppending
Richtung	Out



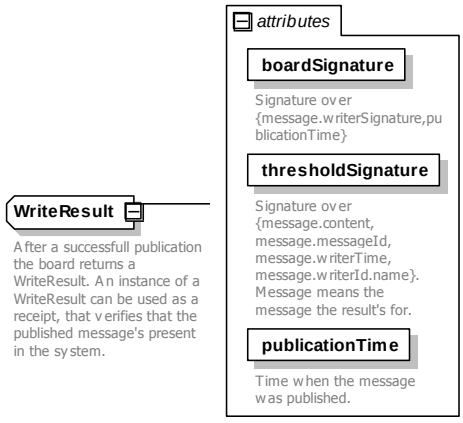
WriteRequest

Service	MessageAgency
Operation	write
Richtung	In



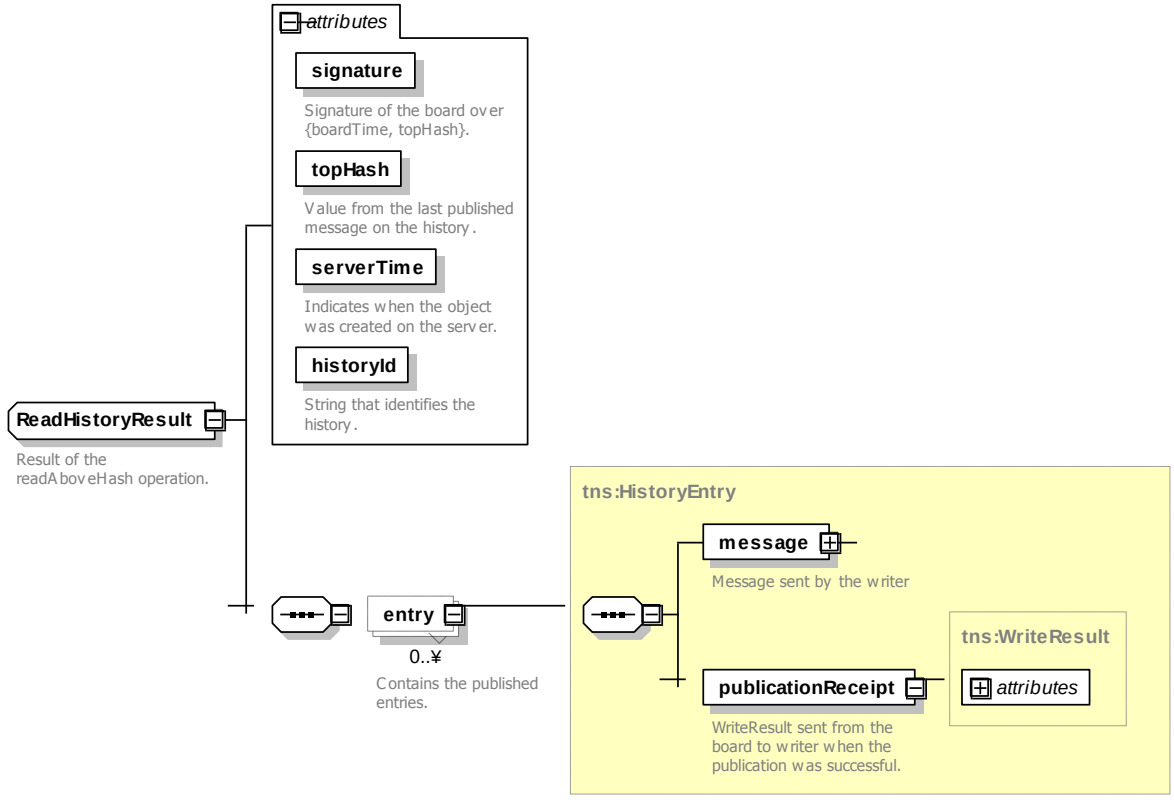
WriteResult

Service	MessageAgency
Operation	write
Richtung	out



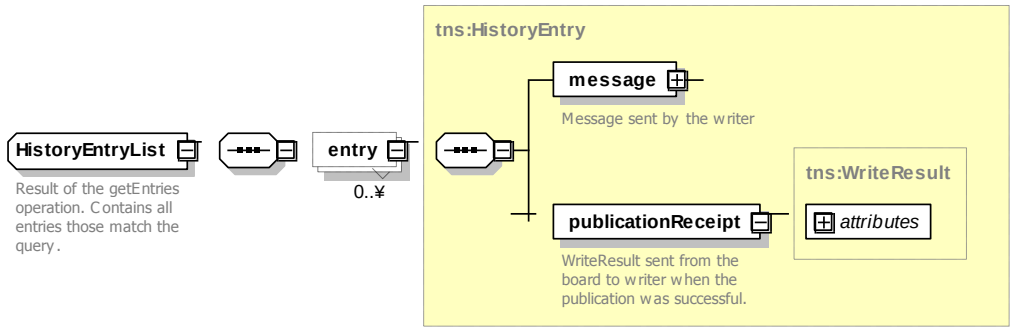
ReadHistoryResult

Service	MessageAgency
Operation	readAboveHash
Richtung	out



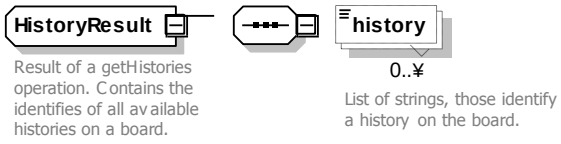
HistoryEntryList

Service	MessageAgency
Operation	getEntries
Richtung	out



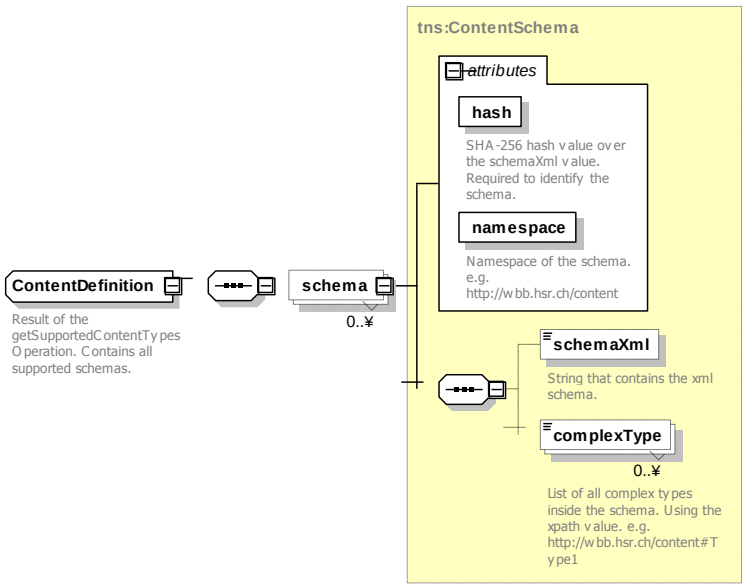
HistoryResult

Service	MessageAgency
Operation	getHistories
Richtung	out



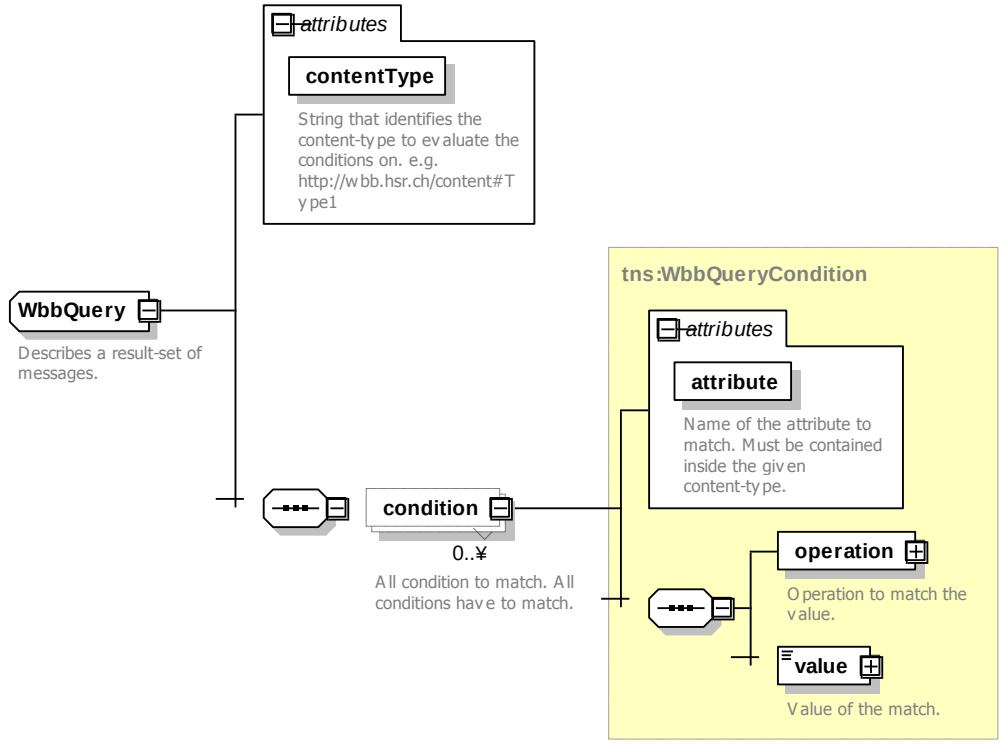
ContentDefinition

Service	MessageAgency
Operation	getSupportedContentTypes
Richtung	out



WbbQuery

Service	MessageAgency
Operation	getEntries
Richtung	In



4.5 Meldungsinhalt

Die Meldung (`Message`) wird von einem Writer erstellt. Dieser Abschnitt beschreibt den Inhalt, welcher eine Meldung beinhalten darf.

Das Board darf nur Inhalt akzeptieren, welcher einem Content-Typen zugeordnet werden kann. Der Content muss gültiges XML sein. Die Zuordnung von einem Inhalt auf seinen Content-Typen passiert über die Element-Definition des Content-Typen in der Content-Typen-Definition. Auch der Inhalt des XML sollte vom Board geprüft werden. Dafür kann er eine XML-Schemavalidierung verwenden. Alle gültigen Schemata liegen in der Content-Typen-Definition.

Beispiel:

Der Writer sendet folgenden Inhalt:

```
 <?xml version="1.0" encoding="UTF-8" standalone="no"?>

<vote xmlns="http://votes.hsr.ch/governance" birthDate="1988-10-
10" familyName="Muster" firstName="Hans" profession="Student"/>
```

In einem XML-Schema der Content-Typen-Definition ist folgender Inhalt vorhanden.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://votes.hsr.ch/governance"
        xmlns:tns="http://votes.hsr.ch/governance"
        elementFormDefault="qualified">
        <complexType name="Vote">
                <attribute name="firstName" type="string" />
                <attribute name="familyName" type="string" />
                <attribute name="birthDate" type="date" />
                <attribute name="profession" type="string" />
        </complexType>
        <element name="vote" type="tns:Vote" />
</schema>
```

Somit kann das Board nun bestimmen, dass der Inhalt auf den Content-Typen, respektive komplexen XML-Typen „*Vote*“ zugeordnet werden kann. Dies geschieht über die Elementdefinition (vote), welche den Typen des komplexen Typen angibt (type=“*tns:Vote*“). Ebenso wird mit der XML-Schemavalidierung gewährleistet, dass das der Wert des Attributs *birthDate* ein gültiges Datum ist.

Zu jedem komplexen Typen in einem der gegebenen XML-Typen werden die dazugehörigen Java-Klassen generiert. Bekommt das Board eine Meldung von einem Writer, dann wird zuerst das geliefert XML validiert. Ist dieses valide, dann wird ein JAXB-Marshaller verwendet, welcher den Inhalt des XML’s in eine Java Objektinstanz umwandelt. Nach diesem Schritt kann die Objektinstanz mit Hilfe der Java Persistence API (JPA) in die Datenbank gespeichert werden.

4.5.1 Content-Typen

Ein Content-Typ umschreibt den Inhalt einer publizierten Meldung. Er kann dabei auch gewisse Validierungsregeln definieren. Jeder Deployer erhält vom Assembler die Content-Typen-Definition. Die Content-Typen sind nichts anderes wie XML-Typen, welche in einem XML-Schema definiert sind. Dabei müssen die unten enthaltenen Teile zwingend im Schema vorhanden sein, um als Content-Typ verwendet werden zu können.

Das XML-Schema welches verwendet wird muss folgendes beinhalten.

1. Type Definition
2. Element Definition welche auf einen Typen verweist.

Als Content-Type wird ein komplexer Typ in einem XML-Schema verwendet. Als Identifikation für einen Content-Typen wird dessen XPath-Name verwendet. Der XPath-Name ist aufgebaut durch den Namespace und dem Name des Typen. Die beiden Strings werden durch eine #-Zeichen getrennt. Beispiel: http://wbb.hsr.ch/entities#Person

Das verwendete XML-Schema darf auf keine anderen Schemas verweisen ausser auf das Standard XML-Schema (http://www.w3.org/2001/XMLSchema).

Die komplexen Typen dürfen nur aus Attributen bestehen. Diese Attribute dürfen jeweils nur auf die simplen Typen aus dem Standard XML-Schema und dem eigenen Schema verweisen.

Mit der `getSupportedContentTypes` Operation gibt es für den Writer die Möglichkeit, die dem Board bekannten Schemas für die Content-Typen abzufragen.

> Details zur Implementation sind unter 5.6 Content-Type (Seite 42).

4.5.2 Identifikation der Meldung

Der Writer ist verantwortlich, dass die Identifikationen seiner erstellten Meldungen immer eindeutig sind. Die Identifikation wird über das Attribut *messageId* des `Message`-Typen mitgegeben. Es kann davon ausgegangen werden, dass der Writer in der Lage ist, für sich selber eindeutige Identifikationen für die Meldungen zu generieren.

Es können mehrere Probleme daraus entstehen, dass der Writer die Identifikation der Meldung selber wählen kann.

Schwachstelle 1
Das erste Problem entsteht, wenn zwei Writer zum ungefähr gleichen Zeitpunkt dieselbe Identifikation für eine Nachricht verwenden. Wenn beispielsweise das Board B1 von einem Writer W1 eine Meldung erhält, dann versucht es die Threshold-Signatur zu bilden. Zum ungefähr gleichen Zeitpunkt sendet der Writer W2 eine Meldung mit derselben Identifikation an das Board B2. Um die Threshold-Signatur bilden zu können, tauschen die Boards B1 und B2 nun Meldungen aus, dass sie die Meldung publizieren wollen. Für B1 und B2 sind die Meldungen vom Writer W1 und W2 identisch, da sie über dieselbe Identifikation verfügen. Die partielle Threshold-Signatur des einen Board kann jedoch vom anderen Board nicht als gültig erklärt werden, da der signierte Wert unterschiedlich ist. Deshalb werden beide Boards ihre erhaltenen Meldungen verwerfen. Somit bietet sich ein möglicher Angriffspunkt an, denn wenn ein korrupter Writer von einem anderen Writer die Identifikation der Meldung während dem Schreiben lesen kann, so kann er eine Meldung mit derselben Identifikation auf die Boards schreiben. Und somit kann es sein, dass beide Meldungen nicht publiziert werden.

Schwachstelle 2
Eine andere Schwachstelle wäre, 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 Threshold-Signaturen und verteilen diese an die übrigen Boards. Die Boards welche die Signatur erhalten stellen jedoch bei der Prüfung fest, dass die Signatur ungültig ist. Das Board kann jedoch nicht feststellen für welchen Meldungsinhalt die Signatur erstellt wurde. Darum bekommt es das Gefühl das sein Partnerboard korrupt ist.

Aus diesen Gründen können die Boards nicht davon ausgehen, dass die Identifikation der Meldung über alle Writer eindeutig ist.

> Für die Boards muss also als eindeutige Identifikation der Meldung das Paar bestehend aus der Identifikation der Meldung vom Writer und der Identifikation des Writers sein.

> Wurde in der Implementation nicht berücksichtigt.

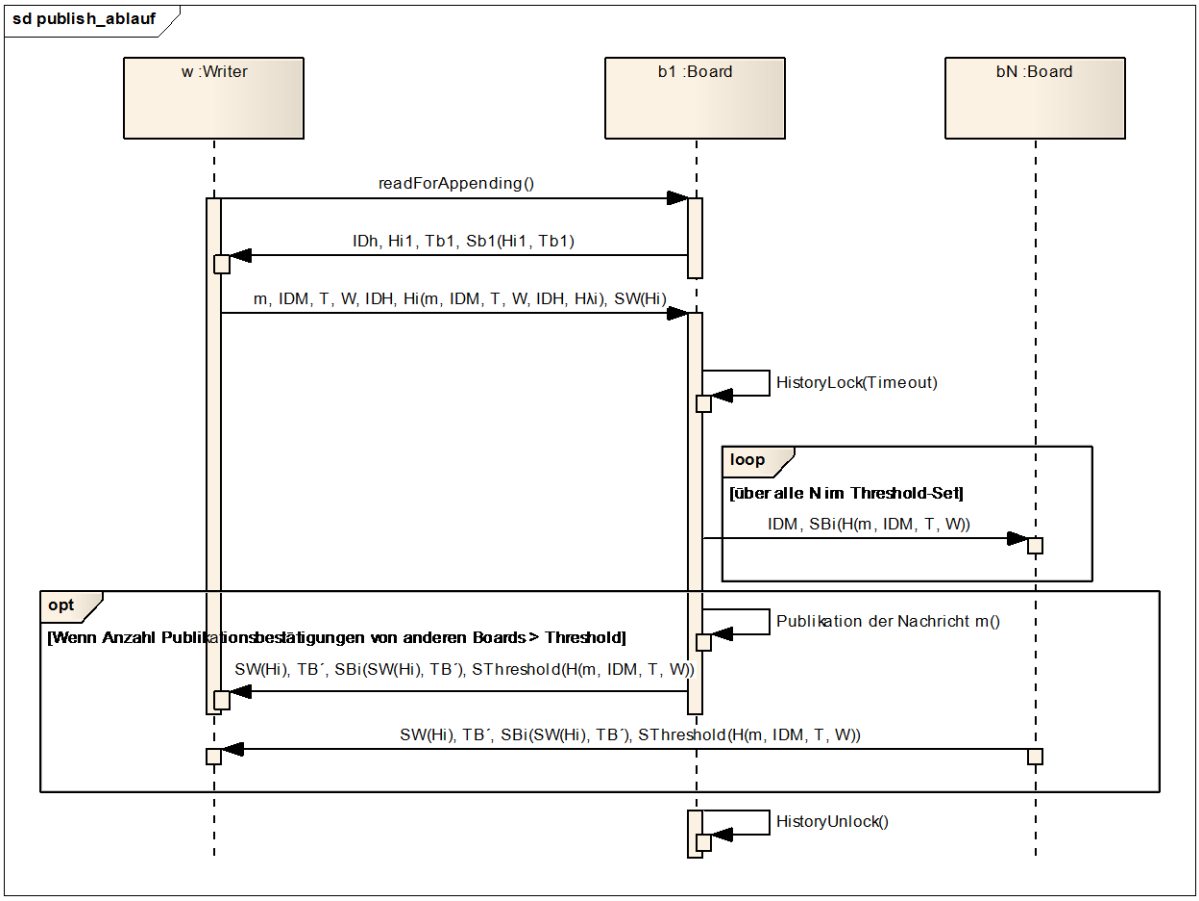
4.6 Publikation

Als Publikation wird der Vorgang bezeichnet, bei dem der Writer eine Meldung auf das Web Bulletin Board schreiben möchte. Wird die Meldung von der Mehrheit der Boards akzeptiert, so wird die Meldung publiziert. Nach dem Publizieren ist garantiert, dass keine Meldung mehr verloren geht, ohne dass es bemerkt wird. Jedes Board legt die Meldung auf seinem Datenspeicher ab.

4.6.1 Ablauf der Publikation

Der gesamte Ablauf der Publikation durchläuft mehrere Schritte.

	Writer	**Board**
1	Aufruf der Methode `readForAppending` auf jedem Board.	Liefert die Identifikation der Historie, auf welche der Writer schreiben soll. Zusätzlich wird der Hash der neusten Meldung auf dieser Historie mitgeliefert.
2	Aufruf der Methode `write` auf die Historie, welche er vom Board erhalten hat.	Prüft die Meldung.
3		Blockiert die Historie und sendet die Information über die eingegangene Meldung über die Methode `notifyPublish` an alle anderen Boards innerhalb des Systems. Wartet bis es mindestens k gültige Bestätigungen von anderen Boards erhalten hat und somit die Threshold-Signatur bilden kann.
4		Stellt dem Writer eine Quittung aus.
5	Wartet bis er mindestens eine gültige Quittung von einem Board erhalten hat. Danach kann er den Publikationsvorgang abrechen und die erhaltene Quittung der Endanwendung weiterreichen.	



4.6.2 Schritt 1: Anfrage

Bevor ein Writer eine Meldung an ein Board senden kann, muss er das Board anfragen, auf welcher Historie die Meldung geschrieben werden soll. Das Board muss dem Writer eine Identifikation der Historie und den Hash-Wert der zuletzt auf der gegebenen Historie publizierten Meldung mitliefern. Ebenfalls muss das Board die Antwort signieren. Das Board erklärt sich mit der signierten Antwort bereit, dass es die Meldung des Writers auf der gegebenen Historie entgegennimmt.

Der Writer verwendet die `readForAppending` Operation. Als Parameter muss er seine eigene Identifikation mitsenden. Als Antwort erhält er ein `ReadForAppendingResult`. Er muss prüfen, ob das Resultat die Spezifikation einhält.

Fehlerfälle

Die folgenden Fehlerfälle können auftreten und vom Writer erkannt werden.

Signatur

Die Signatur der Antwort (`ReadForAppendingResult`) kann nicht verifiziert werden.

➔Das Board sollte einer Ombudsstelle gemeldet werden. Möglicherweise wurden die falschen öffentlichen Schlüssel verteilt oder die Übertragung war fehlerhaft.

Timeout

Das Board liefert keinen Wert innerhalb einer spezifizierten Zeit.

➔Im wiederholten Fall sollte der Writer das Board einer Ombudsstelle melden.

Identifikation Falls dem Board die im Request mitgegebene Identifikation nicht bekannt ist, kann es eine `UnknownComponentException` zurückliefern. Dies kann für den Writer als Prüfung dienen, ob er auch im WBB-System bekannt ist.

➔Falls der Writer von mehreren Boards diese Antwort erhält, sollte er die Ombudsstelle kontaktieren und eine Aufnahme in das WBB-System beantragen.

Das Board sollte garantieren, dass die zur Verfügung stehenden Histories bestmöglich auf die Anfragen der Writer verteilt werden. Möglich ist beispielsweise eine Round-Robin Strategie, die für jeden Aufruf der `readForAppending` Operation die nächste freie Historie zurückliefert. Das Board darf Histories auch mehrfach herausgeben – es wird jedoch nur ein späterer Write-Versuch auf diese Historie durchkommen.

In der Referenzimplementation wurde eine simple Round-Robin Strategie umgesetzt. Dabei wird ebenfalls geprüft, ob eine Historie blockiert ist.

4.6.3 Schritt 2: Schreiben

Die `write`-Operation wird von einem Writer auf allen Boards im System durchgeführt. Sie kann jedoch erst ausgeführt werden, wenn er von dem Board das Resultat der `readForAppending` Operation erhalten hat. Die Antwort der `readForAppending` Operation liefert die Identifikation der Historie auf welche der Writer schreiben soll. Ebenfalls erhält der Writer den Hashwert der letzten Meldung auf der Historie. Dieser Hashwert fliesst in die Erstellung für den Hashwert der neuen Meldung ein. Damit wird die Verkettung der Historie gewährleistet. Die richtige Verkettung der Historie wird bei der Endauswertung geprüft.

Bezüglich der write-Operation enthält die Spezifikation noch offene Punkte. Mehr dazu unter 4.8.2 „Freie Wahl um auf Historie zu schreiben“ (Seite 33).

Fehlerfälle

Publikation Falls die Publikation auf dem Board aus fehlschlägt, kann es eine `WriteFailedException` auslösen. Dieser Fehler ist berechtigt, wenn die gesendete `WriteRequest` Nachricht die Spezifikation verletzt.

➔Falls der Writer von allen Boards diesen Fehler erhält, so sollte es seine gesendeten Nachrichten prüfen. Tritt der Fehler jedoch nur bei einem einzelnen Board auf, so kann der Fehler ignoriert werden.

Identifikation Falls dem Board die im Request mitgegebene Identifikation nicht bekannt ist, kann es eine `UnknownComponentException` zurückliefern. Dieser Fall sollte jedoch nie eintreten, da das Board diesen Fehler bereits bei der vorausgehenden `readForAppending`-Operation auslösen sollte.

➔Falls der Writer von mehreren Boards diese Meldung erhält, sollte es Ombudsstelle kontaktieren und eine Aufnahme in das WBB-System beantragen.

Der Writer sollte für den Schreibvorgang keinen Timeout setzen. Denn sobald der Schreibvorgang gestartet wurde, kann er nicht mehr abgebrochen werden. Der Writer muss jedoch nur auf den erfolgreichen Abschluss eines Schreibvorgangs auf einem Board warten.

4.6.4 Schritt 3: Erstellung der Threshold-Signatur

Nach dem Erhalten und Prüfen einer `write`-Anfrage eines Writers, muss das Board allen anderen Boards innerhalb des Systems mitteilen, dass es die Meldung publizieren wird. Dafür ruft es auf dem MessagePublisher-Service der anderen Boards die Operation `notifyPublish` auf und übergibt eine `InterBoardPublicationInformation`. Diese teilt den anderen Boards mit, dass das Board bereit ist, die Meldung des Writers zu publizieren. Diese Nachricht enthält im Wesentlichen die partielle Threshold-Signatur und die Identifikation der vom Writer erhaltenen Meldung.

Danach muss das Board darauf warten, bis es genügend `InterBoardPublicationInformation` von anderen Boards erhält und danach eine gültige Threshold-Signatur erstellen kann.

Während das Board auf die Antworten der restlichen Boards wartet, muss es die verwendete Historie blockieren. D.h. alle folgenden `write`-Anfragen eines Writers müssen verworfen werden. Dem Writer kann mit der `WriteFailedException` mitgeteilt werden, dass die Historie blockiert ist.

Das Board sollte einen Timeout verwenden, welcher ausgelöst wird, wenn die Threshold-Signatur nicht in einem definierten Zeitfenster erstellt werden konnte. Ansonsten werden zu viele Histories blockiert und es können keine Meldungen mehr publiziert werden. Um dem Writer mitzuteilen, dass das Board bereit dazu wäre die Meldung zu publizieren, aber nicht genügend andere Boards im System, kann das Board ebenfalls die `WriteFailedException` verwenden.

Der Timeout kann vom Board dynamisch gewählt werden. Das Monitoring kann Aufschluss geben, wie lange die Erstellung einer Threshold-Signatur dauert. Der obere Wert kann dann als Timeout verwendet werden. Merkt das Board jedoch, dass zu viele Histories gleichzeitig blockiert sind, so sollte der Timeout kleiner gewählt werden.

Der Timeout für die Bildung der Threshold Signatur kann auf der jeder Board-Komponente einzeln konfiguriert werden.

Die erhaltenen `InterBoardPublicationInformation` werden in einem sogenannten MessageWaitSet abgelegt. Als Identifikation in welches MessageWaitSet eine `InterBoardPublicationInformation` abgelegt wird, wird durch die Identifikation der Meldung bestimmt. Sobald genügend `InterBoardPublicationInformation` erhalten wurde, wird das MessageWaitSet verworfen. Es kann jedoch vorkommen, dass nach der Publikation noch `InterBoardPublicationInformation's` eintreffen. Diese werden ebenfalls in einem MessageWaitSet abgelegt. Um unnötigen Speicherverbrauch zu vermeiden wird zyklisch geprüft ob die MessageWaitSet's noch benötigt werden.

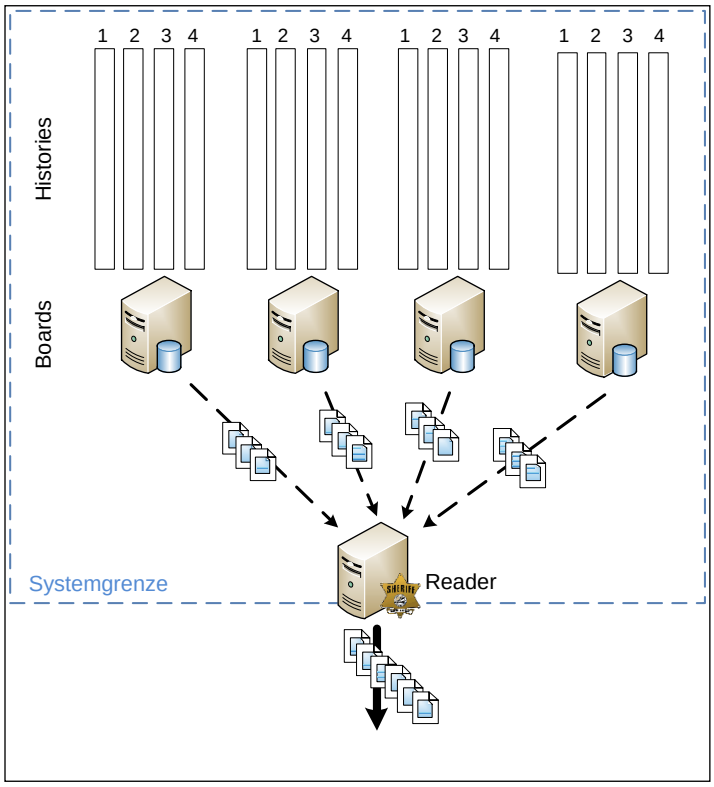
4.6.5 Schritt 4: Ausstellen einer Quittung

Sobald das Board für eine Meldung eine Threshold Signatur erstellen konnte, muss es die Meldung auf der Historie ablegen. Nach dem Ablegen, kann er dem Writer eine Quittung ausstellen, welche den Erfolg der Publikation anzeigt. Diese Quittung liegt in Form einer `WriteResult` Nachricht vor. Der Writer sollte seine versendete `WriteRequest` Nachricht plus die erhalten `WriterResult` Nachricht als Quittung zusammenfassen.

4.7 Endauswertung

Als Endauswertung wird das Zusammentragen aller publizierten Meldungen innerhalb eines WBB-Systems bezeichnet. Dabei kann die Endauswertung zu jedem Zeitpunkt von jedem Reader ausgeführt werden. Sie soll dazu dienen

- alle gespeicherten Meldungen zusammentragen und diese um Duplikate bereinigen
- das verteilte Web Bulletin Board auf dessen Korrektheit hin zu überprüfen



Um dies zu vollbringen muss der Reader alle Meldungen aller Boards zusammentragen, diese validieren und zusammenfügen. Dabei bietet jedes Board zwei Möglichkeiten um die Meldungen abzufragen.

a. *Gesamtabfrage aller gespeicherten Meldungen auf dem Board*
b. *Gefilterte Abfrage von Meldungen*

Als nächstes werden die zwei Varianten erklärt und danach auf die Regeln eingegangen, die für die Endauswertung eines WBB-Systems benötigt werden.

4.7.1 Meldungen im gesamten WBB-System lesen

Da die Histories die ein Board führt öffentlich sind, ist es jedem Reader möglich, die Meldungen abzufragen, die auf den einzelnen Histories gespeichert sind. Der Reader durchläuft dabei alle Boards und deren Histories im gesamten WBB-System und prüft dabei folgendes:

1. Sind noch alle Signaturen valide
2. Sind alle Meldungen auf einer Historie noch korrekt verkettet

Falls nicht mehr alle Einträge einer Historie korrekt verkettet sind, können die Einträge noch verwendet werden, welche unterhalb auf der Historie liegen und noch korrekt sind.

Beim Zusammentragen aller Meldungen hat der Reader auch die Aufgabe, die doppelten Meldungen zu filtern. Das heisst die Meldungen, welche noch auf mehreren Histories publiziert sind, nur einfach zu zählen. Dafür kann er die Identifikation der einzelnen Meldungen verwenden.

Um zu garantieren, dass auch wirklich alle Meldungen zusammengetragen werden konnte, müssten zuvor noch alle ausgestellten Quittungen validiert werden.

> Das Lesen der einzelnen Histories kann parallel stattfinden. Die Histories haben untereinander keinerlei Abhängigkeiten. Der Synchronisationspunkt sollte bei der Prüfung der Identifikation gesetzt werden.

> Die Prüfung der Quittungen entfällt, da der Reader über keinen Speicher für die Quittungen verfügt. Die Prüfung der Histories wird jedoch bei der Endauswertung durchgeführt.

4.7.2 Meldungen gefiltert lesen

Nebst der Möglichkeit alle Meldungen eines Boards auszulesen, muss jedes Board zusätzlich die Möglichkeit bieten, Meldungen nach Content-Typ und optional nach Filterkriterien auf Attributwerten zu filtern.

Diese Operation ist keinen speziellen Restriktionen unterstellt. Es kann also vom Reader nicht angenommen werden, dass die Operation vom Board auch korrekt implementiert wurde. Das will heissen, dass der Reader nachprüfen muss, ob die von den Boards gelieferten Meldungen die Filterkriterien auch wirklich einhalten.

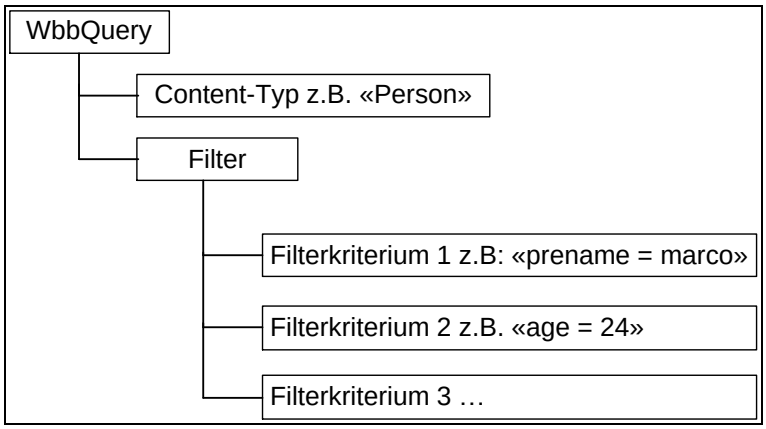
Beim gefilterten Auslesen von Meldungen werden immer nur Meldungen von einem Content-Typ geliefert. Dieser muss dem Operationsaufruf zwingend mitgegeben werden. Neben dem Content-Typ können noch beliebig viele Filterkriterien auf die Attribute des Content-Typ mitgegeben werden. Die maximale Menge aller Meldungen ist die Menge aller Meldungen mit dem geforderten Content-Typen – dies ist der Fall wenn keine Filterkriterien mitgegeben wurden.

Folgende Vergleichsoperatoren für die Filterkriterien sollten unterstützt werden:

Abk.	Typ	Java-Repräsentation	Zulässige Werte
EQ	EqualsQueryCompOperator	==	Alle Datentypen aus dem xs-Namespace.
NEQ	NEqualsQueryCompOperator	!=	Alle Datentypen aus dem xs-Namespace.
GT	GtQueryCompOperator	>	xs:int
GTE	GteQueryCompOperator	>=	xs:int
LT	LtQueryCompOperator	<	xs:int
LTE	LteQueryCompOperator	<=	xs:int

Auf der linken Seite der Bedingung kann nur immer ein Attributname sein und auf der rechten Seite ein konstanter Wert. Die Verknüpfung mehrerer Bedingungen passiert über den logischen UND-Operator. Das Attribut muss im gegeben Content-Typen enthalten sein.

Die Filterkriterien, welche den Content-Typ und die einzelnen Attribut-Filterkriterien beinhalten, werden mit dem Datentyp `WbbQuery` definiert.



Um alle Meldungen zusammenzutragen, die solche speziellen Filterkriterien erfüllen, sendet der Reader die Filterkriterien an alle Boards. Der Reader sollte die erhaltenen Meldungen überprüfen. Dies auf zwei Punkte:

- Stimmen die Signaturen (Threshold und Board) noch.
  - Damit kann ausgeschlossen werden, dass der Inhalt der Meldung modifiziert wurde.
- Stimmt der Inhalt der Meldung mit dem gegeben Filterkriterium überein
  - Ein Board könnte versuchen zusätzliche Meldungen zu übermitteln. Alle Meldungen, welche also nicht mit dem Filterkriterium übereinstimmen, können vom Reader verworfen werden.
    ➔Das Auftauchen einer Meldung, welche nicht den Filterkriterien entspricht, sollte einer Ombudsstelle gemeldet werden.

Die Überprüfung der korrekten Verkettung bleibt beim gefilterten Auslesen der Meldungen aus, da die Meldungen in einer losen Struktur geliefert werden und nicht hintereinander auf einer Historie abgespeichert sein müssen. Was ebenfalls nicht verhindert werden kann ist, dass das Board bewusst gewisse Meldungen nicht an den Reader sendet. Deshalb ist es wichtig, dass bei der Endauswertung alle Boards kontaktiert werden und danach die Schnittmenge der Resultate verwendet wird.

Wenn jedoch der Verdacht besteht, dass ein Board Meldungen zurückhält, dann kann eine komplette Auswertung gemacht werden. Wenn dann Meldungen auftauchen welche einem vorhergehenden Kriterium entsprochen haben, diese aber bei der Abfrage der `getEntries` Operation nicht mitgeliefert worden sind, dann kann dies der Reader der Ombudsstelle melden.

Fehlerfälle

Unbekannter Content-Typ	Der Writer sendet dem Board mit auf welchen Content-Typen sich das Filterkriterium bezieht. Kennt das Board diesen Content-Typen nicht so kann es dies mit einer `NotSupportedComplexTypeNameException` Nachricht anzeigen. ➔Daraufhin kann der Reader mit der Ombudsstelle in Kontakt treten und prüfen ob der Fehler bei sich oder beim Board liegt.
Nicht unterstützte Abfrage	Die `NotSupportedQueryException` Meldung kann vom Board an den Reader zurückgesendet werden, wenn es ein mitgesendetes Filterkriterium nicht interpretieren kann. ➔Liefern mehrere Boards im System diese Meldung, dann sollte der Reader seine Abfrage überprüfen. Ansonsten kann es der Ombudsstelle gemeldet werden.

Die zurückgelieferte Datenmenge einer gefilterten Abfrage kann gross werden. Je nach Filterkriterium können alle Meldungen in einem WBB-System zurückgeliefert werden. Um die Datenmenge einer einzelnen Abfrage zu verkleinern gibt es mehrere Ansätze. Die einzelnen Ansätze werden unter 4.8.3 „Datenmenge von gefilterten Abfragen“ (Seite 33) diskutiert.

4.7.3 Meldungen einer Historie lesen

Bei der Variante „Meldungen im gesamten WBB-System lesen“ werden die Histories aller Boards zusammengetragen. Dazu stellt das Board die Operation `readAboveHash` zur Verfügung, mit welcher ein Reader die nächsten Meldungen auf einer Historie lesen, welche nach der Meldung mit dem mitgegeben Hash publiziert wurden. Das Board kann selber wählen wie viele Meldungen es pro Aufruf zurückliefert. Es ist jedoch verpflichtet jede Antwort zu signieren. In jeder Antwort ist

auch enthalten, welches der Hashwert der obersten Meldung ist. Somit kann der Reader bestimmen, wann er zuoberst auf der Historie angelangt ist, und keine weitere Anfrage starten muss. Als erste Anfrage kann der Reader einen null-Wert mitliefern. Danach muss das Board die untersten Meldungen auf der Historie liefern.

Beispiel:

In der Tabelle nebenan ist der publizierte Inhalt auf einer Historie ersichtlich. Die oben liegenden Meldungen sind zeitlich zuletzt publiziert worden. Der unterste Eintrag dementsprechend zuerst.

Die Anzahl Meldungen, die ein Board pro Anfrage zurückliefert liegt im Beispiel bei 2.

Der Reader möchte den gesamten Inhalt der Historie lesen.

1. Der Reader startet die Anfrage mit einem null-Wert für den Hash
    a. Das Board liefert die Einträge 1&2; Als Top-Hash H6
2. Der Reader stellt fest, dass der Hash H2 nicht der Hashwert des obersten Eintrages ist. Und startet nochmals eine Anfrage; Dieses Mal jedoch mit dem Hashwert H2 als Parameter.
    a. Das Board liefert die Einträge 3 und 4.
3. Der Reader führt Punkt 2 solange aus, bis der Top-Hash mit dem Hashwert einer erhalten Meldung übereinstimmt.

ID	Hashwert
6	H6
5	H5
4	H4
3	H3
2	H2
1	H1

Bei der Implementation sollte darauf geachtet werden, dass nicht zu viel und nicht zu wenige Meldungen auf einmal zurückgegeben werden. Bei zu vielen Meldungen kann es sein, dass der Arbeitsspeicher zu sehr belastet wird und es zu Timeouts während der Kommunikation kommt. Bei wenigen steigt dagegen der Kommunikationsoverhead. Deshalb sollte die Anzahl konfigurierbar sein, um so aufgrund von Informationen aus dem Monitoring die optimale Einstellung zu finden.

Auf dem Board ist konfigurierbar wie viel Einträge pro Abfrage an den Reader zurückgegeben werden.

4.7.4 Verkettung der Historie prüfen

Es ist notwendig die Verkettung der Meldungen auf einer einzelnen Historie zu prüfen. Damit kann erkannt werden, wenn ein Board selber versucht Meldungen auf einer Historie abzulegen oder Meldungen zu modifizieren. Es kann jedoch nicht festgestellt werden, ob die oberste Meldung auf der Historie entfernt wurde. Um dies festzustellen, müssen zusätzlich die erhaltenen Quittungen validiert werden. Das eigentliche Überprüfen der korrekten Verkettung geschieht indem die Hashes aller Einträge in jeder Historie erstellt werden und mit den abgespeicherten verglichen werden.

4.7.5 Validierung einer Quittung

Der Writer erhält vom Board eine Quittung, falls die Publikation der Meldung gelungen ist. Diese Quittungen erhält der Writer von allen Boards. Der Writer ist jedoch nicht verpflichtet auf die Quittungen aller Boards zu warten. Es genügt schon wenn er eine gültige Quittung von einem Board erhalten hat. Es muss einfach garantiert sein, dass die in der Quittung enthaltene Threshold-Signatur gültig ist.

Diese Quittung kann der Writer einem Reader zur späteren Verifizierung übergeben. Dieser Schritt ist notwendig, da ein Board ansonsten immer die oberste Meldung auf einer Historie löschen könnte. Prüft jedoch der Reader immer wieder die erhaltenen Quittungen, so kann entdeckt werden, wenn ein Board Meldungen löscht.

Die Validierung kann aus zwei Gesichtspunkten betrachtet werden. Der eine ist das oben angesprochene Prüfen einer Historie, dass keine Meldungen gelöscht werden. Der andere erlaubt es dem Writer sicherzustellen, dass die publizierte Meldung noch auf dem WBB-System vorhanden ist.

Vorgehensweise zur Prüfung der Quittung.

1. In der Quittung ist enthalten, auf welchem Board und auf welcher Historie die Meldung publiziert wurde.
    a. Der Reader durchläuft von unten her die Historie, bis er die Meldung aus der Quittung wiederfindet. Dabei prüft er die Verkettung der Historie und ob die Meldung aus der Quittung und jener auf der Historie noch identisch sind.
2. Falls bei der Prüfung unter Punkt 1 etwas nicht in Ordnung war, wird die Prüfung auf die Histories aller anderen Boards ausgeweitet. Falls der Writer noch über Quittungen von anderen Boards verfügt, dann müssen nicht alle Histories eines Boards durchlaufen werden. Sondern nur jene welche in den Quittungen vermerkt sind.

> In der Referenzimplementation werden nur die Histories in die Prüfung miteinbezogen, welche auch in der Quittung vermerkt sind. Der im Punkt 2 beschriebene Rückfall wurde nicht implementiert. Das hat zur Folge, dass die Validierung der Quittung fehlschlägt, sobald das Board welches die Quittung ausgestellt hatte, ausfällt.

4.8 Offene Punkte in der Spezifikation

4.8.1 WBB-System kann blockiert werden

Mit einem Testszenario aus der Semesterarbeit konnte gezeigt werden, dass zwei korrupte Writer ein WBB-System teilweise blockieren können. Im Prinzip geht es darum, dass die korrupten Writer nur `write`-Anfragen an eine Anzahl Boards versenden welche kleiner ist als der Threshold. Die Threshold-Signatur kann also nie zustande kommen. Das bemerken die Boards aber erst nach dem Ablauf des Timeouts. Und solange sind die Histories blockiert. Weitere Details sind aus der Semesterarbeit zu entziehen.

Um dies zu verhindern wäre eine Möglichkeit, dass vom Writer aus keine gleichzeitigen Schreib-Operationen zugelassen werden. Somit kann maximal eine Historie pro Writer blockiert werden. Bei wenigen Writern im WBB-System bedeutet das, dass nicht mehr Histories als Writer benötigt werden, da maximal eine Historie pro Writer blockiert werden kann. Ebenso bedeutet es mehr Komplexität für die Writer-Implementation. Deshalb wurde entschieden diese Lösung nicht zu verwenden. Um trotzdem diesen Fall zu verhindern verschob man die Lösung auf eine mögliche Implementation einer Ombudsstelle.

4.8.2 Freie Wahl um auf Historie zu schreiben

Mit der Einführung der `readAboveHash`-Operation kann ein Writer auf eine beliebige Historie schreiben, ohne zuvor die `readForAppending`-Operation zu verwenden. Die `readAboveHash`-Operation liefert nämlich die oberste Meldung auf der Historie inklusive Hash. Mit diesem Hash kann der Writer die `write` Operation auf dem Board ausführen. Somit kann der Writer selber wählen, auf welcher Historie er die Meldung schreiben möchte. Das Board kann jedoch nicht direkt prüfen, ob der Writer auf eine Historie schreiben möchte, welche er als Resultat der `readForAppending` Operation zurückgeliefert hat.

4.8.3 Datenmenge von gefilterten Abfragen

Die Datenmenge einer gefilterten Abfrage kann unter Umständen sehr gross werden. Je nach Filterkriterium können alle Meldungen in einem WBB-System zurückgeliefert werden. Die Idee

waren *Paged Queries,* bei welchen mitgegeben werden, wie viele Resultate man geliefert bekommen will. Im Folgenden werden mehrere Methoden von solchen Paged Queries diskutiert.

Paged Queries

Verwenden wir die Menge aller Meldungen die auf einem WBB-System publiziert wurden und das Filterkriterium erfüllen als M. Dieses M sollte noch aufgeteilt werden. D.h. der Reader kann angeben welche Teilmenge (Subset) er von M haben möchte.

Die Tabelle 1 zeigt die Meldungen, wie sie auf zwei Board verteilt sein könnten.

Board1		**Board2**	
Index	**Meldung**	**Index**	**Meldung**
8	M9	8	
7	M8	7	M9
6	M7	6	M8
5	M6	5	M7
4	M5	4	M6
3	M4	3	M5
2	M3	2	M4
1	M2	1	M3
0	M1	0	M1

Tabelle 1

Methode: Start- und Endindex

Die Teilmenge ist durch einen Start- und Endindex der einzelnen Meldungen innerhalb von M definiert. Jedes Board verwendet den gegeben Start- und Endindex ebenso für die gespeicherten Meldungen.

Dabei können folgende Probleme beobachtet werden:

Fehlende Einträge

Fehlen auf einem Board Meldungen, so vergrössert sich die Resultatliste. Und Einträge, welche bereits zuvor in einem Subset enthalten waren erscheinen nochmals.

Beispiel:

0-2 → M1, M2, M3, M4
3-5 → M4; M5; M6; M7

Im Endresultat erscheint M4 doppelt.

Vertauschte Reihenfolge	Ist der Publikationszeitpunkt der Meldungen vertauscht dann kommt es zu einem ähnlichen Effekt, wie wenn die Einträge fehlen. Fällt nämlich die Indexgrenze zwischen zwei vertauschte Einträge, dann erscheint er in beiden Subsets.

Methode: Publikationszeit

Der Reader spezifiziert den Start und Endzeitpunkt der Publikationszeiten der Meldungen für eine Teilmenge.

Die Publikationszeit des Boards kann nicht verwendet werden, da die Server-Zeiten unterschiedlich sein können. Ebenso kann die Zeit des Writers nicht verwendet werden, da auch diese unterschiedlich sein können.

Methode: Globaler Index

Für die Verwendung eines Index, welcher über alle Boards gleichverteilt ist, müsste die Kommunikation zwischen den Boards erweitert werden, so dass ein globaler Index erstellt werden kann. Da man die Kommunikation zwischen den Board nicht noch vergrössern wollte und die Implementation der Content-Typen nicht zwingend für die Verwendung eines WBB-Systems sind wurde die Anforderung nicht in die Spezifikation aufgenommen.

Mit einem globalen Index könnte man jedoch das Problem lösen.

Cursor

Der Reader handelt mit jedem Board eine Indexstrategie aus. Das Board kann aufgrund von dieser Indexstrategie alle Meldungen von allen seinen Histories zurückliefern, welche zeitlich nach dem vom Reader gegeben Index publiziert wurden. Es ist jedoch nicht möglich denselben Index für alle Boards zu verwenden. Eine geeignet Indexstrategie wäre die Wahl der Publikationszeit, da diese auf dem Board synchron läuft.

Deshalb sollte der Reader einen Cursor verwenden. In diesem Cursor ist gespeichert, welchen Index er bei welchem Board verwendet hat. Der Cursor kann dann unterstützen, um an die nächsten Meldungen zu gelangen.

5 Implementation

5.1 Voraussetzungen

Basierend auf der Simulation aus der vorangegangen Studienarbeit soll in der vorliegenden Bachelorarbeit die Optimierung der Komponenten stattfinden. Zudem soll die Verteilung der einzelnen Komponenten des verteilten Web Bulletin Board auf eine Serverinfrastruktur geschehen. Aufgrund der immer noch vorliegenden Anforderung das Ganze mittels Open Source zu realisieren, wurden sämtliche Komponenten mittels Java implementiert. Die einzelnen Komponenten sollen innerhalb eines J2EE Applikationsserver lauffähig sein, wodurch viele Vorteile der bereits ausgereiften Applikationsumgebung genutzt werden können.

Für die Kommunikation unter den einzelnen Komponenten werden SOAP-Webservices verwendet. Webservices bieten sich aufgrund des einfachen HTTP-Protokolls an, wie dies bereits im Paper von Herr Krummenacher angemerkt wurde. Zudem können die Implementationen der einzelnen Komponenten des verteilten Web Bulletin Board auch in anderen Technologien umgesetzt werden.

5.1.1 Open Source

Die Rahmenbedingungen wurden von der vorausgehenden Arbeit, Semesterarbeit Vertrauenswürdiges und robustes Bulletin Board, übernommen. Darin war auch der Punkt enthalten, dass Open Source Technologien verwendet 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.“*
>
> *(Aufgabenstellung Studienarbeit 2010 Vertrauenswürdiges und robustes Bulletin Board für E-Voting, 2010)*

5.1.2 Referenzimplementation

Die Schnittstellen und Spezifikationen für ein verteiltes Web Bulletin Board sollten nicht nur durch eine Technologie implementiert werden können. Vielmehr sollte es auch möglich sein, dass die verschiedenen Web Bulletin Board Komponenten durch Implementationen auf anderen Plattformen ausgetauscht werden können. Um dies zu gewährleisten muss die Kommunikation zwischen den Komponenten mit plattformübergreifenden Standards passieren.

In der vorliegenden Arbeit ist eine Referenzimplementation der Spezifikation enthalten.

5.2 Anforderungen

Zentrale Anforderung an das WBB System ist es, dass eine Meldung innert kurzer Zeit und ohne grössere Verzögerung auf den Boards gespeichert werden kann. Da die Anzahl paralleler Schreibvorgänge nicht vorausgesagt werden kann und von Anwendungsfall zu Anwendungsfall stark variieren kann, muss das System möglichst gut auf den jeweiligen Anwendungsfalls angepasst werden können. Im Falle des vorliegenden Systems hat man hierzu zwei Faktoren, um die WBB Umgebung an eine erhöhte Nutzlast anzupassen. Einerseits die Anzahl Histories pro Board und andererseits die Anzahl der Komponenten, seien dies die Boards oder die Writer, zu erhöhen.

Die Anzahl Histories anzupassen ist keine grosse Sache, kann jedoch auch nur solange erhöht werden, wie dies die darunterliegende Hardware zulässt. Somit wurde uns schnell klar, dass ein einfaches Deployment unserer Komponenten auf verschiedene physikalische Server möglich sein muss, um das System an den jeweiligen Anwendungsfall anzupassen.

Infolge dessen kam man schnell auf die Idee unsere Komponenten innerhalb eines J2EE Applikationsservers anstelle als Standalone Applikation laufen zu lassen.

5.3 Applikationen

Die Referenzimplementation zu Demonstrationszwecken beinhaltet zwei Applikationen. Die Board-Applikation stellt eine vollständige Implementation einer Board-Komponente dar. Die Writer-Applikation beinhaltet die Funktionalität eines Writers und die eines Readers.

5.4 Architektur

Die verschiedenen Teile der beiden Applikationen sind in Module aufgeteilt. Wobei die Module auch von beiden Applikationen genutzt werden können. Die einzelnen Applikationen sind in einer 3-Schichtenarchitektur aufgebaut.

5.4.1 Writer-Applikation

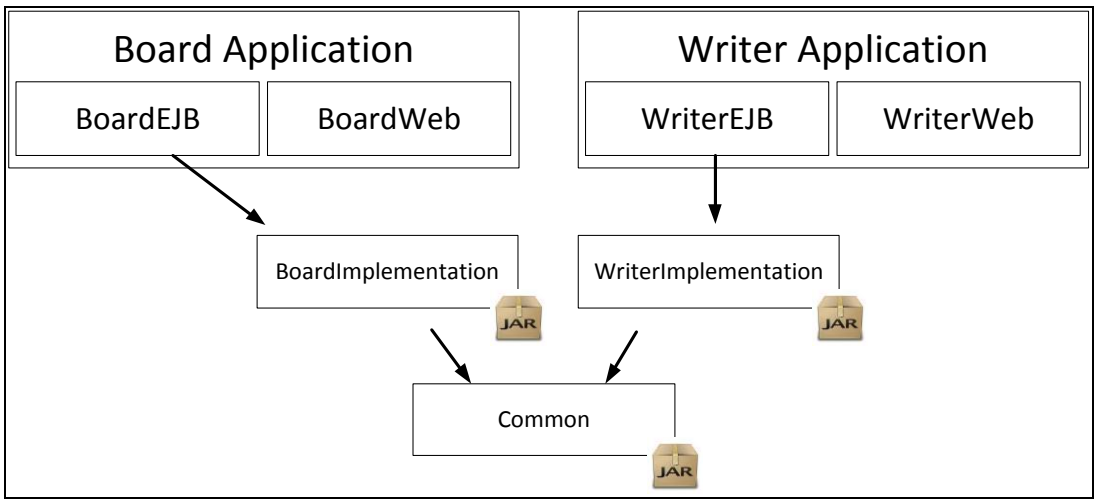
Presentation	•Web-UI
Business	•Writer-Implementation •Reader-Implementation
Data-Access	•Webservice-Client für Kommunikation mit Boards

5.4.2 Board-Applikation

Presentation	•Web-UI •Webservice
Business	•Board-Implementation
Data-Access	•Webservice-Client für Kommunikation mit anderen Boards •Persistierung der Meldungen

5.4.3 Module

Jede Applikation ist in verschiede Module aufgeteilt. Diese Module können auch von verschieden Applikationen verwendet werden. So wird das Common-Modul von allen Applikationen benötigt.



BoardEJB	EJB Modul; Stellt das Board aus der BoardImplementation Bibliothek in den JNDI Namesraum. Ebenso enthält es die Implementation für das Persistieren der Daten über JPA.
BoardWeb	Stellt den SOAP-Webservice zur Verfügung und sucht innerhalb des JNDI Namesraum nach einer Board Implementation.
BoardImplementation	J2EE unabhängige Bibliothek, welche die Grundfunktionalität für ein Board enthält.
WriterEJB	EJB Modul; Stellt den Writer aus der WriterImplementation Bibliothek in den JNDI Namesraum.
WriterWeb	Enthält die Web-Oberfläche für die Demo-Applikation. Sucht innerhalb des JNDI Namesraum nach einer Writer Implementation.
WriterImplementation	J2EE unabhängige Bibliothek, welche die Grundfunktionalität für ein Writer enthält.
Common	Enthält die Klassen, welche von allen Komponenten verwendet werden. Enthält ebenfalls die Implementation des Threshold Signatur Verfahren.

Die oben beschriebenen Module und Bibliotheken sind als eigenständige Eclipse-Projekte definiert.

5.5 Deployment

5.5.1 Produktives System „Verteiltes Web Bulletin Board“

Da das System meist von einer umgebenden Endanwendung gebraucht wird, ist es vorgesehen, dass nur die Boards je auf einem J2EE Applikationsserver installiert sind. Die Komponenten Writer und Reader, dessen Aufgabe das Erstellen und Auslesen von Meldungen aus dem System ist, werden in die Endanwendung integriert. Dabei nutzt diese die vorhandene WBB-API, welche die Funktionen des Writer und Reader zur Verfügung stellt.

Abbildung 1 veranschaulicht den gedachten Produktiveinsatz des verteilten Web Bulletin Board. Dabei bilden die GlassFish Applikationsserver den Kern des Systems während diese durch die Endanwendung über die WBB-API angesprochen werden. Dabei können Writer und Reader dieselbe API verwenden und in beliebiger Anzahl auftreten.

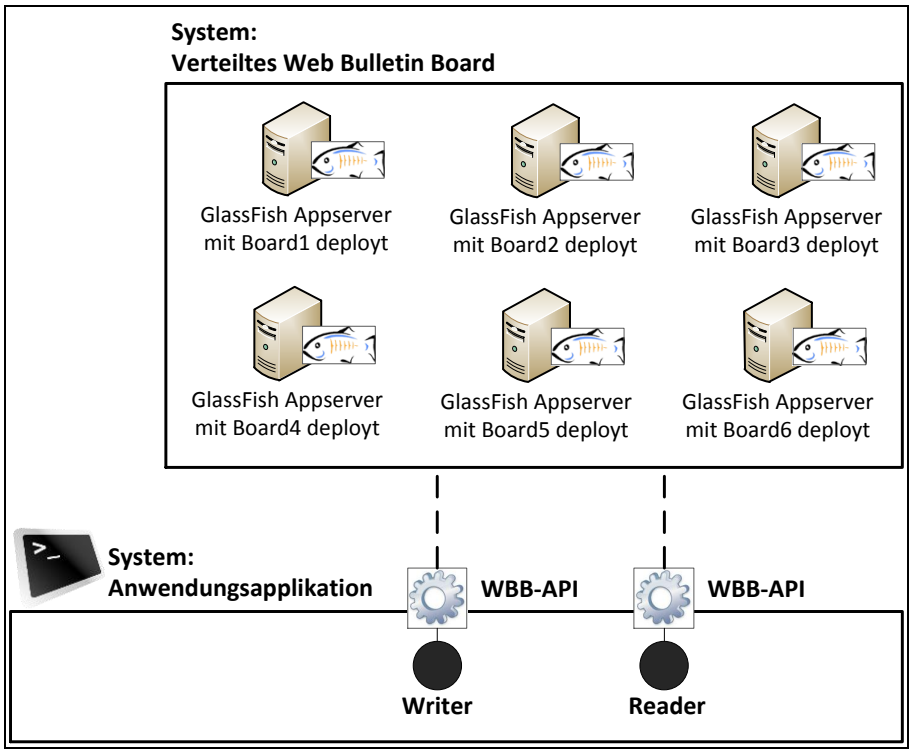


Abbildung 1Produktiveinsatz mit Anwendungsapplikation, welche die API nutzt

5.5.2 System „Verteiltes Web Bulletin Board“ zu Demonstrationzwecken

Zur Demonstration und Veranschaulichung der Funktionalität des verteilten Web Bulletin Board wurde eine Demo-Applikation entwickelt, welche sowohl einen Writer als auch einen Reader simulieren kann. Diese Applikation nutzt wie im produktiven Einsatz die WBB-API um mit den Boards zu kommunizieren und kann ebenfalls auf einem J2EE GlassFish Applikationsserver installiert werden.

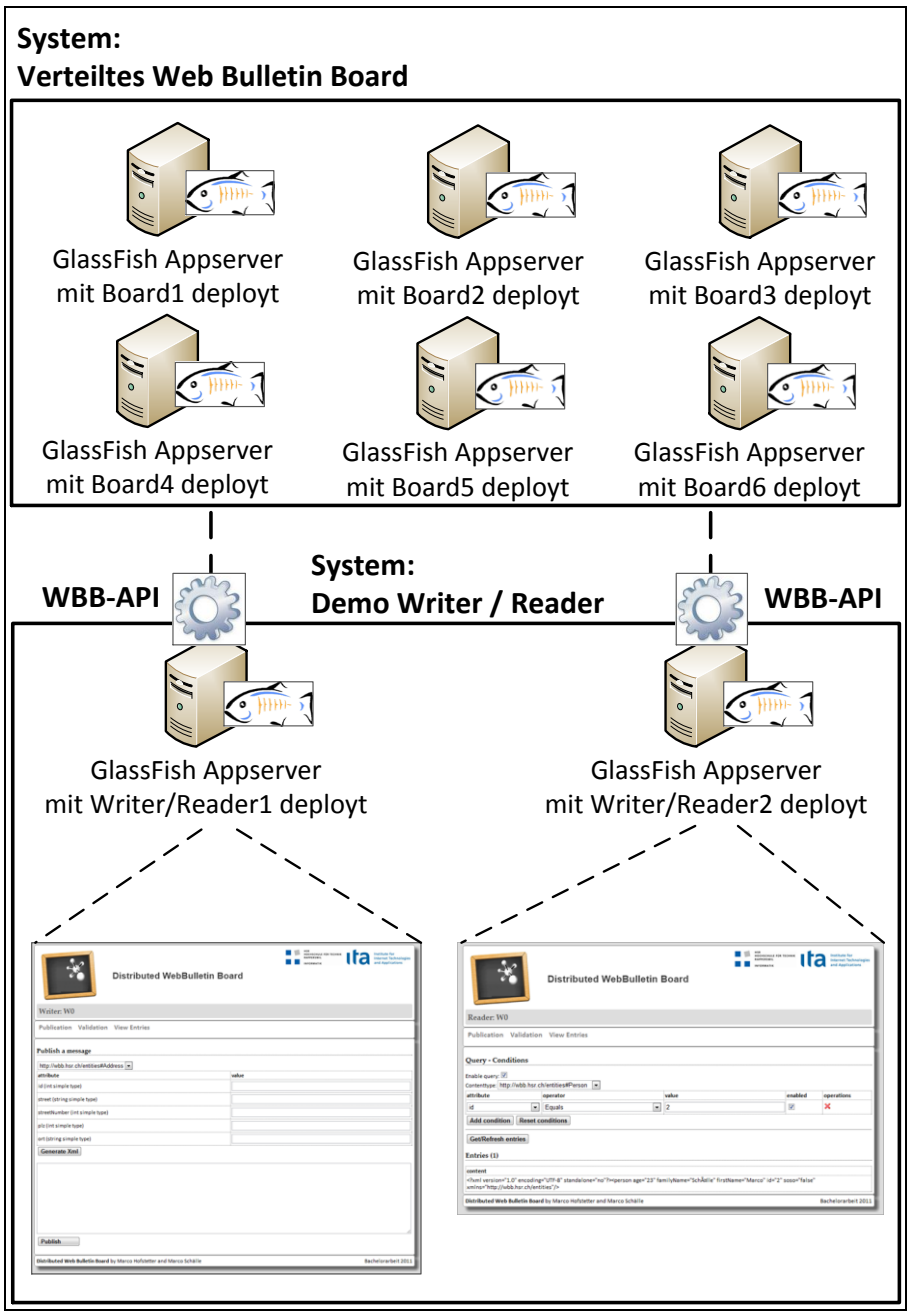


Abbildung 2 Demoapplikation welche eine Endanwendung simuliert

5.6 Content-Typen

Die Spezifikation sieht vor, dass der Inhalt einer Meldung durch sogenannte Content-Typen beschrieben wird.

Die Definition eines Content-Typen soll mittels XML-Schema stattfinden. Dieses XML-Schema wird von einer zentralen Stelle (Assembler) an alle Deployer eines Board oder Writers versendet. Hintergrund der ganzen Idee ist, dass das WBB-System nicht nur als Basis für eine Online-Voting Applikation dienen kann, sondern auch in andere Anwendungen, wie beispielsweise Logging-Systeme genutzt werden kann.

Ebenfalls sieht die Spezifikation vor, dass die publizierten Meldungen über eine Schnittstelle auch gefiltert abgefragt werden können. Über diese Schnittstelle soll der Endanwendung die Möglichkeit geboten werden, gezielt einzelne Meldungen aus dem WBB-System mittels eines Filterkriterium zu lesen ohne alle publizierten Meldungen laden zu müssen.

Man einigte sich pro Content-Typ eine eigene Datenbankstabelle zu führen und diese mit dem dazugehörigen Eintrag aus der bestehenden HistoryEntry – Tabelle zu verknüpfen. Dies sollte es ermöglichen, dass die Abfragen performant ausgeführt werden können. Ebenso bietet jede relationale Datenbank die Möglichkeit an, die Einträge einer Tabelle zu filtern. In der Spezifikation sind im Moment nur rudimentäre Filtermöglichkeiten vorgesehen, welche auch anders implementiert werden könnten. Doch ist es denkbar, dass die Filterkriterien noch erweitert werden. Ebenso bietet diese Lösung die Möglichkeit an, dass man den originalen Inhalt der Meldung 1:1 in der Datenbank ablegen kann.

Die Abbildung 3 zeigt die Struktur einer Datenbank, wie sie sich das Board hält, wenn ein Content-Typ definiert wurde. Und zwar ist ein Content-Typ (ContentType1) mit zwei Attributen (attribute1, attribute2) definiert.

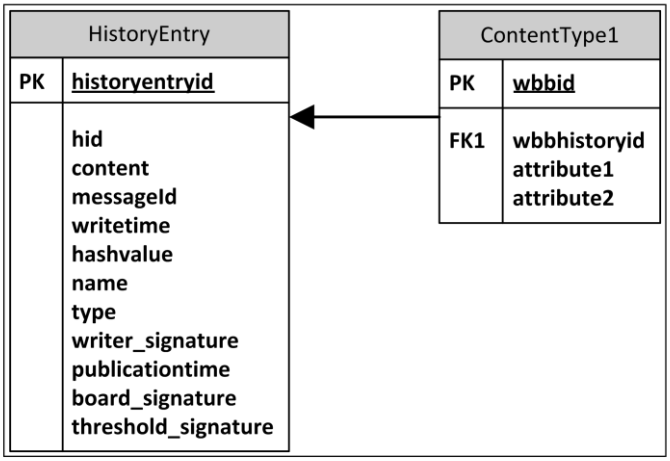


Abbildung 3

In der vorhergehenden Implementation wurde der gesamte Datenbankzugriff mittels eines OR-Mapper realisiert, mit welchem man bequem Java-Instanzen in einer Datenbank persistieren kann. Um dies auch für die Content-Typen zu verwenden, ist es erforderlich, dass der als XML-Schema definierte Content-Typ in eine Java-Klasse umgewandelt wird und diese der Java Runtime sowie dem JPA EntityManager bekanntgemacht wird. Der JPA EntityManager legt die dafür nötigen Tabellen selbst an. Abbildung 4 zeigt schematisch den Ablauf.

Abbildung 4

Die Tabelle 2 zeigt einen Auszug aus der HistoryEntry Tabelle wie er vorkommen kann.

HistoryEntryId : int	**HistoryId : string**	**MessageId : string**	**Content : string**	**[...]**
...	...	...	...	...
2	H5	M2	Inhalt in Form eines Strings	...
3	H3	M3	Freitext	...
4	H2	M4	1234	...
...	...	...	...	...

Tabelle 2HistoryEntry

Mit der neuen Spezifikation, welche vorsieht dass der Inhalt ein gültiges XML-Dokument sein muss, ändert sich der Inhalt der Content-Spalte. Zu sehen in der Tabelle 3. So wird darin das gesamte XML-Dokument, wie es vom Writer erstellt wurde abgelegt. Zusätzlich werden jedoch noch zusätzlich die beiden Tabellen Person (Tabelle 4) und Auto (Tabelle 5) erstellt.

HistoryEntryId : int	**HistoryId : string**	**MessageId : string**	**Content : string**	**[...]**
1	H5	M2	<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?><person familyName="Muster" firstName="Max" xmlns="http://wbb.hsr.ch/entities"/>	...
2	H3	M3	<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?><auto gaenge="5" farbe="rot" plaetze="4" xmlns="http://wbb.hsr.ch/entities"/>	...
3	H2	M4	<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?><person familyName="Kupfer" firstName="Frieda" xmlns="http://wbb.hsr.ch/entities"/>	...
...	...	...	...	...

Tabelle 3 HistoryEntry mit XML-Dokument in Content-Spalte

HistoryEntryId : int	**familyName : string**	**firstName : string**
1	Muster	Max
3	Kupfer	Frieda
...	...	...

Tabelle 4 Person mit Fremdschlüsselbeziehung zu Tabelle HistoryEntry

HistoryEntryId : int	**Gaenge : int**	**Farbe : string**	**Plaetze : int**
2	5	Rot	4
...	...	...	...
...	...	...	...

Tabelle 5 Auto mit Fremdschlüsselbeziehung zu Tabelle HistoryEntry

Die neue Implementation erfüllt die Anforderungen, Daten von unterschiedlichen Typen auf dem Board zu speichern und danach mittels Filterkriterien wieder danach zu suchen. Zudem kann durch

die Definition der Content-Typen mittels XML-Schema gewährleistet werden, dass nur gültige Daten gespeichert werden.

Dabei werden pro Publikation eines Eintrags auf dem Board zwei Datensätze in der Datenbank erstellt. Zum einen der bisherige HistoryEntry und zum anderen der Eintrag in der zugehörigen Tabelle des jeweiligen Content-Typen. Der zweite Eintrag wird zudem über eine Fremdschlüsselbeziehung mit dem Eintrag aus der HistoryEntry-Tabelle verknüpft.

Eine Abfrage mit Filterkriterien wird lediglich auf die Content-Typen-Tabellen abgesetzt und danach wird durch die Fremdschlüsselbeziehung der Eintrag in der HistoryEntry-Tabelle gefunden.

5.6.1 Analyse

Um aus einem XML-Schema eine Klasse zu generieren und diese dem JPA EntityManager bekannt zu machen gibt es diverse Möglichkeiten. Vor der Umsetzung wurden verschieden Lösungen analysiert.

Variante 1: Dynamische Klassen

Es gibt die Möglichkeit mit Bibliotheken wie *Javassist* Klassen während der Laufzeit zu generieren. Mittels dem Java ClassLoader ist es danach möglich, der Java Runtime die neuen Java Klassen bekanntzumachen. Dieser Ansatz kann auch für die Implementation der Content-Typen verwendet werden. Als Vorarbeit muss jedoch erst einmal das XML-Schema geparst werden, um an die Definition des Content-Typen zu gelangen. Danach ist eine eigene Zuordnung von den XML-Typen zu den Java-Klassen erforderlich. Für die generierten Klassen muss danach auch noch die Konfiguration für das Object-Relational Mapping *(orm.xml)* erstellt werden.

Als eigentliches Problem dieser Variante stellte sich heraus, dass die Klasse zwar einfach der Java Runtime, jedoch nicht ohne Weiteres dem JPA EntityManager bekanntgemacht werden konnte. Deshalb war es nicht möglich den Content-Typ in der Datenbank zu speichern und später danach zu suchen.

Variante 2: MOXy

Mittels des Teilprojekts „MOXy“ aus der EclipseLink Implementation ist es möglich, aus einer XML-Schema Definition direkt zur Laufzeit sogenannte *DynamicTypes* zu erzeugen. Diese DynamicTypes können danach wieder verwendet werden, um über die EclipseLink JPA Implementation Instanzen dieser *DynamicTypes* in einen Datenbank abzulegen.

Die Content-Typen werden also zu *DynamicTypes*, welche von der einer Content-Klasse abgeleitet sind (Abbildung 5). Als JPA-Inheritance Strategie wird dabei „*Table per class*“ verwendet. Dies, da für die einfache Read-Abfrage nur die bereits bestehende Tabelle mit den Einträgen verwendet werden kann. Und die Abfragen können über eine Tabelle gemacht werden, womit die Vereinigung von mehreren Tabellen entfällt.

Diese Variante wurde verworfen, da die von GlassFish verwendete EclipseLink Implementation nicht mit der EclipseLink Implementation kompatibel war, welche die Unterstützung für DynamicTypes beinhaltet. Die Möglichkeit besteht zwar, dass eine GlassFish Installation auch mit einer anderen EclipseLink Implementation läuft, jedoch wurde darauf verzichtet. Denn das würde bedeuten, dass man nicht mehr die Standard-Installation verwenden kann.

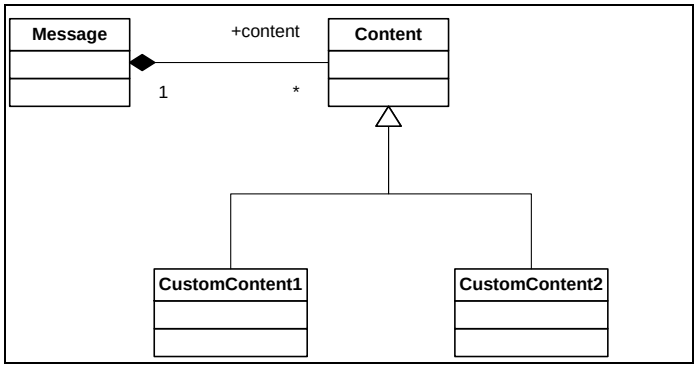


Abbildung 5

Variante 3: Statische Klassen

Aus Erfahrung wusste man, dass das Einbinden von normalen Java-Klassen in eine Anwendung kein Problem darstellt. Den Klassen können Annotationen hinzugefügt werden, welche es erlauben mit Hilfe einer JAXB Implementation, Instanzen von dieser Klasse in ein XML abzubilden. Ebenso stellt es kein Problem dar, um diese Klassen in einem JPA EntityManager zu verwenden und somit die Instanzen zu persistieren.

Die Spezifikation sieht jedoch vor, das die Content-Typen-Definition aus XML-Schemas besteht und nicht aus Java-Klassen. Deshalb musste ein Weg gefunden werden um aus dem XML-Schema die Java-Klassen zu generieren.

Es zeigte sich, dass in der JAXB-Referenzimplementation bereits ein solches Tool enthalten ist. Mit dem Tool *„xjc“* ist es möglich aufgrund eines XML-Schema die Java-Klassen zu generieren, welche für die XML-Datenbindung (d.h. Instanzen aus XML zu erstellen) benötig werden.

Auswertung

Durch diverse Tests der verschieden Varianten zeigte sich, dass die letzte Variante am einfachsten Umgesetzt werden kann. Zwar geht dabei verloren, dass man Content-Typen während der Laufzeit hinzufügen und abändern kann. Doch die Spezifikation sieht auch keinen solchen Fall vor.

Anforderungen	Dynamische Klassen	EclipseLink MOXy	Statische Klassen
Standardisiert	Nein	Ja	Teils
Aufwand	Gross	Gering	Mittel
Fehleranfälligkeit	Hoch	Gering	Gering
Umsetzbar	Nein	Nein	Ja

5.6.2 Umsetzung

Aus der Auswertung der Analyse hat man sich entschieden zum Zeitpunkt des Deployments die Vorbereitungen für die Content-Typen zu treffen. Es kann nämlich davon ausgegangen werden, dass die Content-Typ-Definition vor dem Deployment bereits bekannt sind. Es existiert also keine Anforderung während der Laufzeit neue Content-Typen zu definieren oder bestehende zu Modifizieren.

Das automatisierte Deployment wurde so erweitert, dass es eine jar-Datei generiert welche die Klassen enthält welche die Content-Typen repräsentieren. Ebenso generiert ein Deployment-Task die Konfigurationsdatei (*orm.xml*) welche benötigt wird, um die Datenbank-Tabellen zu den entsprechenden Klassen zu generieren. Abbildung 6 zeigt den Ablauf der Generierung. In der Abbildung 7 ist die Generierung eines Beispielschemas abgebildet.

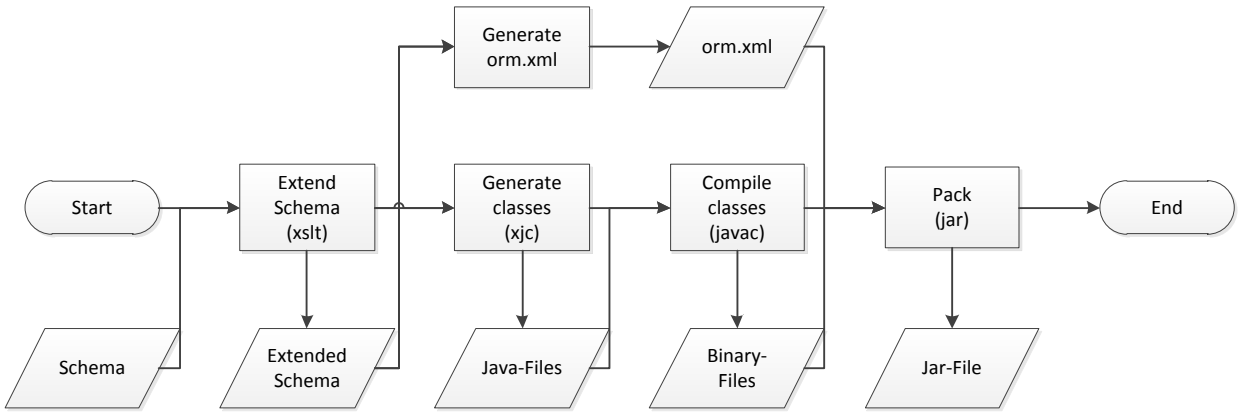


Abbildung 6 Generierung des DynamicEntities.jar

Extend Schema Dieser Prozess nimmt das definierte Schema und erweitert es mit den Attributen wbbId und wbbHistoryId. Aus dem Attribut wbbId wird später in der DB-Tabelle die Spalte, welche als Primärschlüssel verwendet wird. Als Spalte für den Fremdschlüssel wird das wbbHistoryId Attribut verwendet.

Generate classes Der JAXB Binding Compiler (xjc) kann aus einem beliebigen XML-Schema die dazu entsprechenden Java-Klassen generieren.

Compile classes Kopiliert die zuvor generierten Java-Klassen

Generate orm.xml Aus dem erweiterten XML-Schema werden die komplexen Typen sowie deren Attribute ausgelesen. Aus diesen Informationen wird das entsprechende orm.xml generiert.

Pack Die Binärdaten der kompilierten Klassen und das orm.xml werden in eine jar-Datei gepackt.

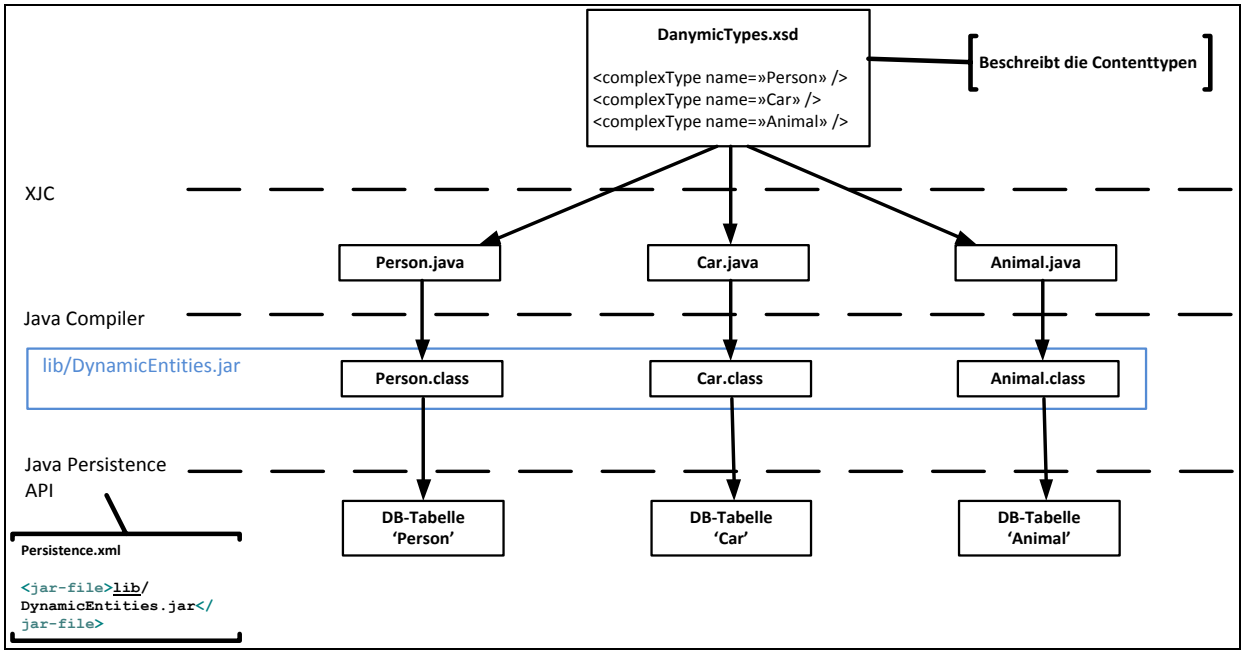


Abbildung 7

Übertragung der Content-Typen

Für die Übertragung von Content, welche zu einem Content-Typen gehören wird gemäss Spezifikation das XML verwendet welches durch das XML-Schema definiert wird. Somit kann das übertragen XML sofort vom Board gegen das Schema validiert werden. Für das Persistieren sieht die Spezifikation nichts vor und es ist dem Board selber überlassen, wie es das XML weiter verarbeitet. Die WBB-API unterstützt eine Instanz in ein XML zu übersetzen, welches danach für die Übertragung verwendet wird. Dazu wird die in JAXB enthaltene Marshalling-Funktionalität verwendet. In Abbildung 8 ist der Ablauf visualisiert.

Persistieren der Content-Typen

Nach dem Erhalt einer Meldung prüft das Board als erstes das erhaltene XML. Danach versucht es aus diesem XML eine Instanz zu erstellen. Dazu verwendet sie die in JAXB enthaltene Unmarshalling-Funktionaliät. Diese Instanz wird danach an den JPA-EntityManager übergeben.

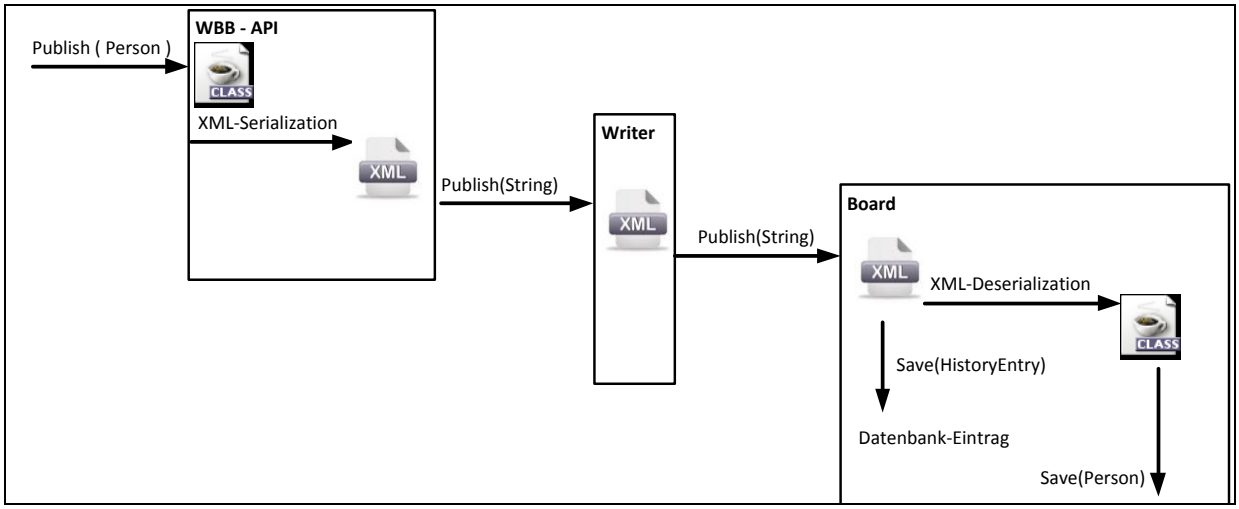


Abbildung 8

Abfrage von Meldungen mittels Filterkriterien

Es können Abfrage von publizierten Meldungen über Filterkriterien, sogenannten Queries getätigt werden. Dabei können sich die Queries nur auf einen Content-Typ beschränkten. Es ist nicht möglich ein Query zu definieren, welches Kriterien aus mehreren Content-Typen enthält.

Die Queries welche vom Writer kommen werden in ein JPA-Query übersetzt. Dadurch ist eine effiziente Abfrage auf der Datenbank möglich. Dadurch, dass für die Erstellung eines JPA -Query die Funktionalität der sogenannten „Criteria Queries“ verwendet wird, könne Angriffe per SQL-Injection ausgeschlossen werden.

Als Resultat eines JPA -Query‘s erhält man eine List von Objektinstanzen welche, die Filterrestriktionen des Queries erfüllen.

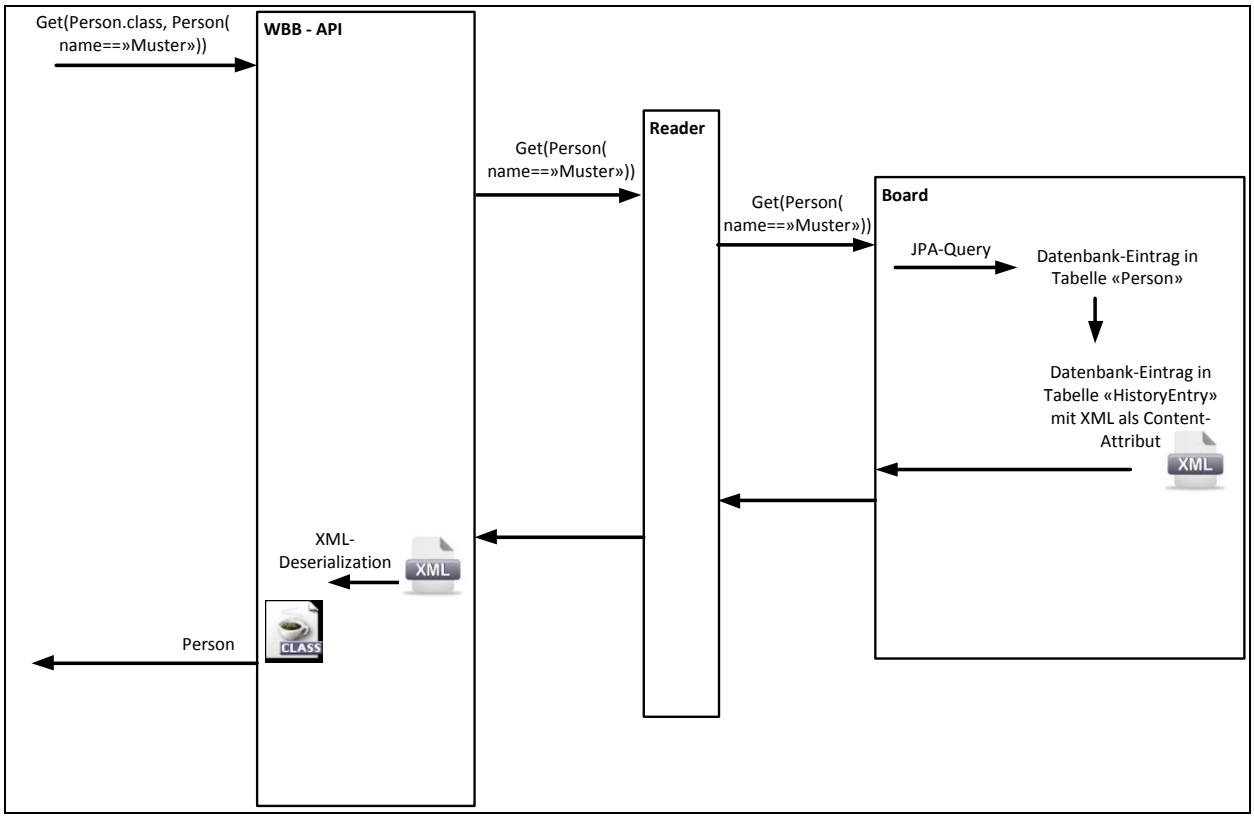


Abbildung 9 Abfrage mittels Query-Syntax

Definition der Content-Typen

Die Definition der Content-Typen geschieht über ein XML-Schema. Dieses erlaubt es nebst der Typenstruktur, d.h die vorhanden Elemente und Attribute, auch Regeln für die Validierung des Inhalts zu spezifizieren. Dieses XML-Schema kann in eine POJO umgewandelt werden und somit in der Datenbank abgelegt werden. Die Wahl fiel auf ein XML-Schema, da damit die Plattformunabhängigkeit gewährleistet ist. Würde man sich auf eine Java-Spezifische Lösung konzentrieren, so verbaut man sich die Möglichkeit, dass es Implementationen eines Boards für andere Plattformen geben kann.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://votes.hsr.ch/parliament"
      xmlns:tns="http://votes.hsr.ch/parliament"
      elementFormDefault="qualified">

      <simpleType name="List">
            <restriction base="string">
                  <enumeration value="SP" />
                  <enumeration value="Gruene" />
                  <enumeration value="SVP" />
                  <enumeration value="GLP" />
                  <enumeration value="CVP" />
                  <enumeration value="FDP" />
            </restriction>
      </simpleType>

      <simpleType name="Position">
            <restriction base="int">
                  <minInclusive value="1"/>
                  <maxExclusive value="32"/>
            </restriction>
      </simpleType>

      <complexType name="Vote">
            <attribute name="list" type="tns:List" />
            <attribute name="position" type="tns:Position"/>
      </complexType>

      <complexType name="Person">
            <attribute name="firstName" type="string" />
            <attribute name="familyName" type="string" />
            <attribute name="birthDate" type="date" />
            <attribute name="profession" type="string" />
            <attribute name="list" type="tns:List" />
            <attribute name="position" type="tns:Position"/>
      </complexType>

      <element name="vote" type="tns:Vote" />
      <element name="person" type="tns:Person" />

</schema>
```

Abbildung 10 Beispiel eines XML-Schemas mit 2 komplexen Typen und Restriktionen zur Validierung

5.7 Benutzeroberflächen

Zu Demozwecken war es gefordert, dass die Komponenten eines verteilten Web Bulletin Board eine Benutzeroberfläche besitzen, um auf deren Funktionalität zuzugreifen. So soll es möglich sein, die auf einem Board gespeicherten Meldungen einzusehen, auf einem Writer eine Meldung zu erstellen und auf einem Reader alle publizierten Meldungen des gesamten Web Bulletin Board auszulesen.

5.7.1 Umsetzung

Da die Komponenten bereits auf einem J2EE Applikationsserver laufen, bot es sich an, die Benutzeroberfläche webbasiert auf dem vorhandenen Applikationsserver zu implementieren. Als Framework zur Umsetzung von Java-basierten Webapplikationen bot sich JavaServer Faces (JSF) an, welches in jedem J2EE Applikationsserver integriert ist, und auf Servlets und der JSP-Technologie basiert. Konkret wurde als Implementation ‚Apache MyFaces' (http://myfaces.apache.org/) verwendet, welche all unsere benötigten Komponenten und UI Elemente unterstützt. Damit liessen sich einfach die bestehenden J2EE Komponenten ansprechen und eine rudimentäre Benutzeroberfläche darüberlegen, welche die Basisfunktionalitäten für den Benutzer verfügbar machen.

Weitere Details und Hilfestellungen zur Anwendung der Benutzeroberfläche können dem Kapitel ‚Anleitungen' – ‚Benutzeroberflächen' entnommen werden.

5.8 Automatisiertes Deployment

5.8.1 Vorausgehende Überlegungen / Anforderungen

Im Verlauf des Projekts wurde ziemlich schnell ersichtlich, dass ein automatisches Deployment nötig wird. Nicht wie in der Semesterarbeit konnte mit einer Umgebung ein gesamtes verteiltes Web Bulletin Board System getestet werden. Es ist erforderlich die Komponenten auf verschiedene Server zu verteilen. Deshalb entschied man sich das Deployment zu automatisieren. Das Kompilieren und zusammenbauen der Komponenten wird dabei automatisch durchgeführt. Dies erlaubt es vor allem während der Entwicklungszeit den aktuellen Stand sofort auf der Umgebung laufen zu lassen. Auch sollen die verschiedenen Einstellungen getestet werden können. Auch mit der Anforderung der Content-Typen, drängt sich ein automatisiertes Deployment auf, da keine einfache Möglichkeit bestand, die Konfiguration der Content-Typen zur Laufzeit vorzunehmen.

5.8.2 Analyse

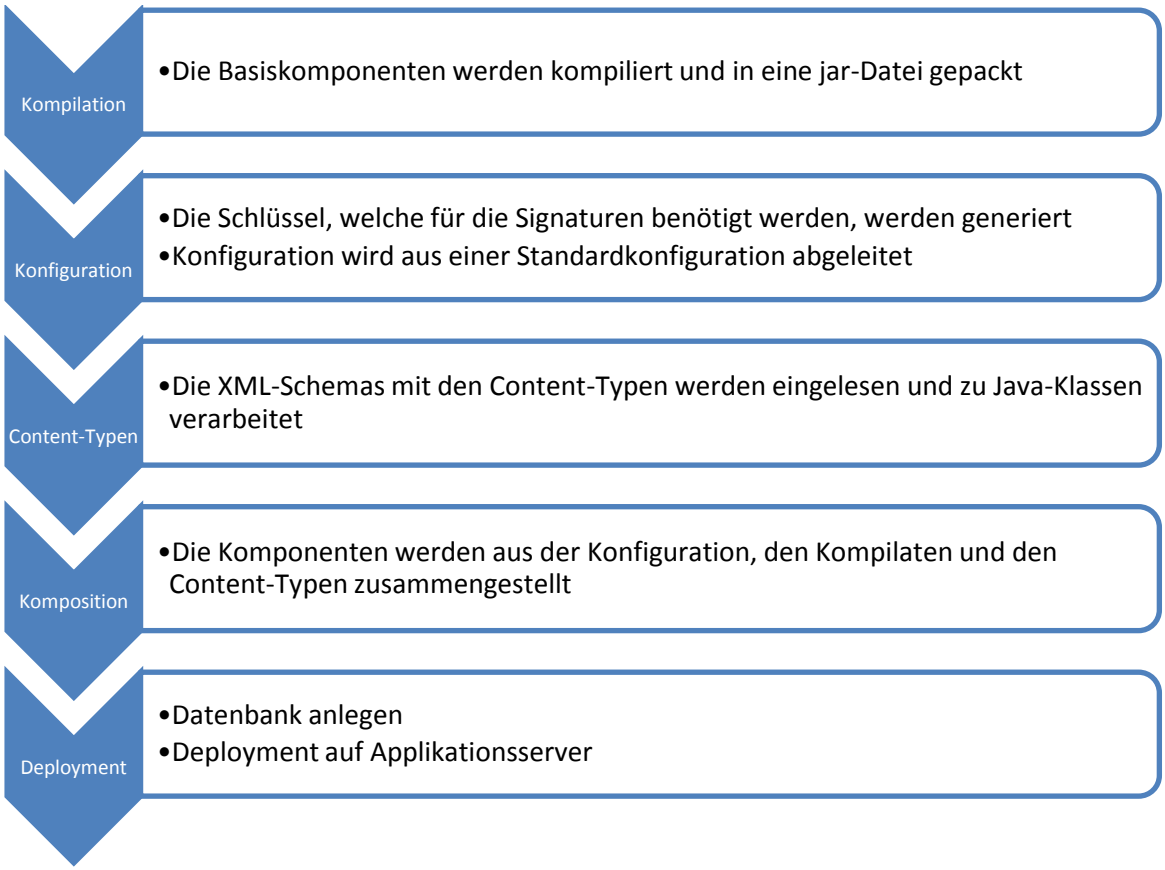
Für die Umsetzung des Deployments fiel der Entscheid auf *Apache ANT*. Dies vor allem, da man dies bereits im vorhergehenden Projekt eingesetzt hat und somit bereits schon Erfahrungen sammeln konnte. Ebenfalls hat man somit direkt die Möglichkeit das Deployment aus der Eclipse-Entwicklungsumgebung zu starten.

5.8.3 Umsetzung

Für das Deployment gibt es ein separates Java Projekt „*Deployment*". Dies enthält alle benötigten Dateien für das Deployment. Zusätzlich ist in jedem Projekt noch ein einzelnes *build.xml* enthalten, welches für die Kompilation und Erstellung des Projektes benötigt wird.

Für die Konfiguration der ausführenden Umgebung gibt es eine einzelne XML-Datei welche alle nötigen Informationen enthält. Für eine detailliert Beschreibung des XML ist eine XML-Schema Datei vorhanden.

Aus dieser XML-Datei werden mit XSL-Transformation die nötigen Artefakte erstellt und auch die Einstellungen ausgelesen. Die Einstellungen werden in eine Property-Datei abgelegt. Somit können sie auch wieder von ANT gelesen und verwendet werden.



5.8.4 Probleme

Probleme traten vor allem bei dem automatischen Deployment in den GlassFish auf. Zwar gibt es speziell ein ANT-Task für diesen Zweck. Dieser ist jedoch noch auf eine ältere Version des GlassFish ausgelegt und funktionierte in der Testumgebung nicht einwandfrei. Er blockierte immer wieder den Ablauf des gesamten Build-Skriptes. Als Lösung wurde direkt das vom GlassFish bereitgestellte Command-Line Tool *„asadmin“* verwendet.

5.9 Tests

5.9.1 Unit-Tests

Für die wichtigsten Klassen wurden Unit-Tests geschrieben. So werden die im Common-Projekt enthalten Signatur-Provider und die Prüfungen der ausgetauschten Nachrichten automatisiert getestet.

5.9.2 Integrationstest

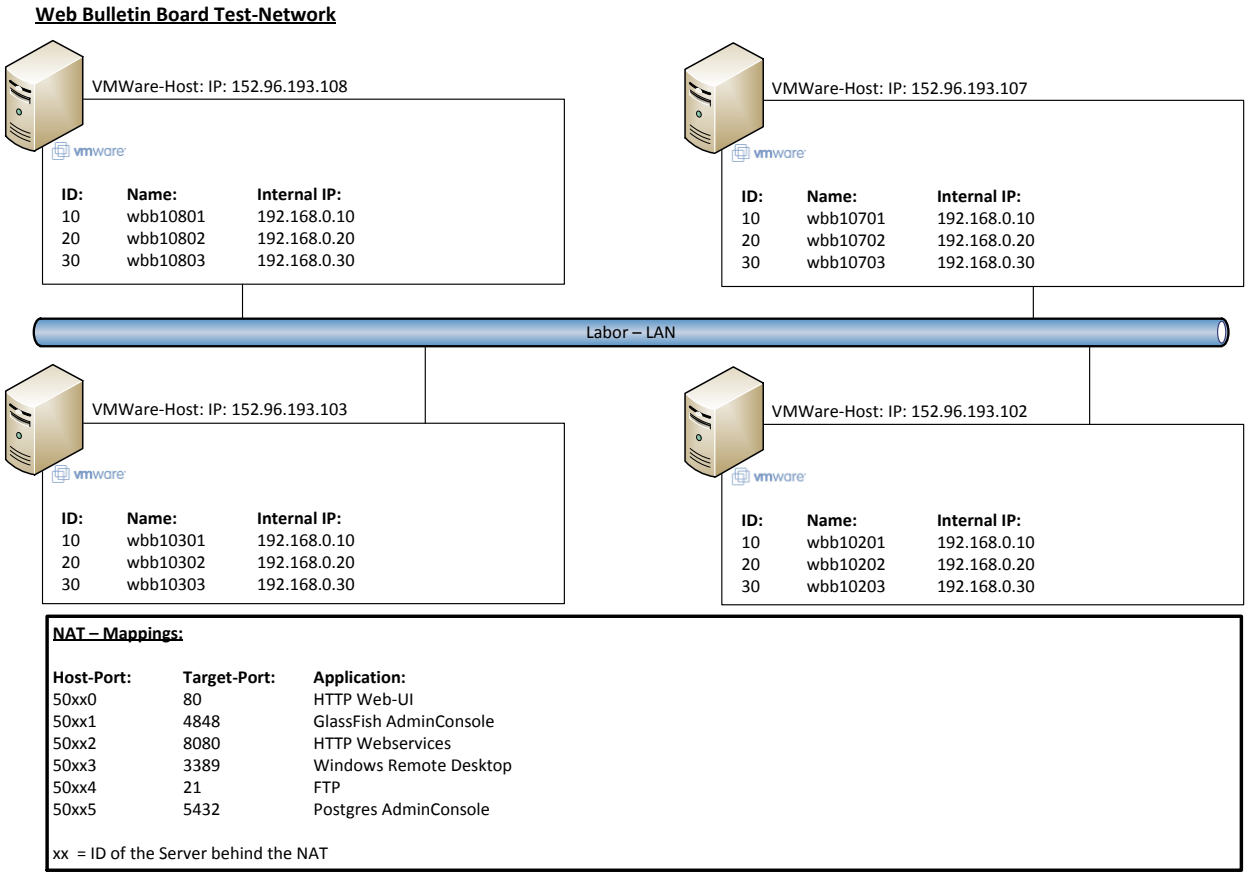
Durch das automatisierte Deployment wurden kontinuierlich Integrationstests auf dem Testsystem durchgeführt. Zur Prüfung wurde ein Test-Client verwendet, welche die wichtigsten Operationen auf dem System durchführt.

5.9.3 Performance-Test

Um Neuerungen in der Implementation auf ihre Performance hin zu testen, wurde ein Referenzsystem aufgesetzt. Auf diesem wurden vordefinierte Szenarien ausgeführt. Da keine Rahmenbedingungen bezüglich der Performance vorliegen, wurden aus den Resultaten der Tests die Punkte extrahiert welche die grösste Ausführungszeit haben. Die Szenarien und endgültigen Testresultate sind nachfolgend beschrieben.

Referenzsystem

Der Aufbau der Testumgebung welche als Referenzsystem herangezogen wird, besteht aus vier dedizierten Computern. Mittels VMWare Server werden pro Computer zusätzlich drei weitere virtuelle Computer simuliert, um genügend Rechner zu haben um verschiedene Szenarien zu simulieren. Auf jedem Computer läuft ein J2EE Applikationsserver (Oracle GlassFish), worauf die einzelnen Komponenten des verteilten Web Bulletin Board installiert werden.



Testszenarien

Im Folgenden werden einzelne Testszenarien unter dem Aspekt der Performance dokumentiert. Ein Szenario setzt sich jeweils aus einer Beschreibung der Konfiguration, des Szenario an sich und dem Resultat mit den damit verbundenen Schlussfolgerungen zusammen.

a) Szenario 1: Publikation einer einzelnen Meldung

Konfiguration

- Anzahl Boards: 6
- Threshold: 4 Boards
- Anzahl Histories pro Board: 10
- Threshold Keysize: 1024bit
- RSA Keysize: 512bit
- Anzahl Writer: 1

Szenariobeschreibung

Ein Writer publiziert eine einzelne Meldung auf das verteilte Web Bulletin Board.

Resultat

Die Publikation einer einzelnen Meldung dauert ungefähr **500ms**. Der am längsten dauernde Zwischenabschnitt ist der Aufruf der Methode write auf den einzelnen Boards. Innerhalb dieser Methode werden unter anderem die partiellen Threshold-Signaturen der anderen Boards mittels Zero-Knowledge-Proof auf ihre Gültigkeit überprüft, um danach bei genügend partiellen Threshold-Signaturen die gesamte Threshold-Signatur zu bilden. Es hat sich herausgestellt, dass ein Entfernen dieser Überprüfung die Dauer einer Publikation enorm verkürzen würde – diese jedoch unabdingbar ist.

Methoden - Messwerte

Zeilenbeschriftungen	Mittelwert [ms]	Maximum [ms]	[ms]
BoardBean	**118**	**1219**	**0**
getEntries-ch.hsr.wbb.api.WbbQuery	0	0	0
getHistories	0	0	0
getSupportedContentTypes	0	0	0
notifyPublish-java.lang.String	154	579	0
readAboveHash-java.lang.String-ch.hsr.wbb.crypto.Hash	0	0	0
readForAppending-java.lang.String	15	32	0
read-java.lang.String	0	0	0
write-java.lang.String-ch.hsr.wbb.messages.WriteRequest	482	1219	0
JPAEntryPersisterBean	**14**	**140**	**0**
getEntries	0	0	0
getEntries-ch.hsr.wbb.api.WbbQuery	0	0	0
getEntries-java.lang.String	0	0	0
getEntries-java.lang.String-ch.hsr.wbb.crypto.Hash-int	0	0	0
getTopHash-java.lang.String	3	16	0
init-ch.hsr.wbb.messages.ContentDefinition	0	0	0
save-ch.hsr.wbb.messages.HistoryEntry	58	140	0
Gesamtergebnis	**71**	**1219**	**0**

b) Szenario 2: Publikation einer einzelnen Meldung mit strengen Schlüsseln

Konfiguration

- Anzahl Boards: 6
- Threshold: 4 Boards
- Anzahl Histories pro Board: 10
- Threshold Keysize: 2048bit
- RSA Keysize: 2048bit
- Anzahl Writer: 1

Szenariobeschreibung

Ein Writer publiziert eine einzelne Meldung auf das verteilte Web Bulletin Board. Alle Komponenten innerhalb des verteilten Web Bulletin Board verwenden dabei sowohl für die RSA Schlüssel als auch für alle Threshold-Schlüssel Schlüsselgrössen von 2048bit.

Resultat

Die Publikation einer einzelnen Meldung dauert ungefähr **2600ms**. Dies zeigt, dass eine Erhöhung der Schlüsselgrösse wie erwartet zu einer enorm längeren Publikationszeit führt. Dieser Faktor könnte durch Auslagerung der Prozesse zur Berechnung der Kryptographie auf externe Komponenten (Hardware) reduziert werden.

Methoden - Messwerte

Zeilenbeschriftungen	Mittelwert [ms]	Maximum [ms]	Minimum [ms]
BoardBean	**449**	**4640**	**0**
getEntries-ch.hsr.wbb.api.WbbQuery	0	0	0
getHistories	0	0	0
getSupportedContentTypes	0	0	0
notifyPublish-java.lang.String-ch.hsr.wbb.messages.InterBoardPublicationInformation	685	3219	0
readAboveHash-java.lang.String-ch.hsr.wbb.crypto.Hash	0	0	0
readForAppending-java.lang.String	100	234	0
read-java.lang.String	0	0	0
write-java.lang.String-ch.hsr.wbb.messages.WriteRequest	1810	4640	0
JPAEntryPersisterBean	**11**	**110**	**0**
getEntries	0	0	0
getEntries-ch.hsr.wbb.api.WbbQuery	0	0	0
getEntries-java.lang.String	0	0	0
getEntries-java.lang.String-ch.hsr.wbb.crypto.Hash-int	0	0	0
getTopHash-java.lang.String	7	31	0
init-ch.hsr.wbb.messages.ContentDefinition	0	0	0
save-ch.hsr.wbb.messages.HistoryEntry	51	110	0
Gesamtergebnis	**249**	**4640**	**0**

c) Szenario 3: Parallele Publikation mehrerer Meldungen

Konfiguration

- Anzahl Boards: 6
- Threshold: 4 Boards
- Anzahl Histories pro Board: 30
- Threshold Keysize: 1024bit
- RSA Keysize: 512bit
- Anzahl Writer: 5
- Threshold Wartezeit: 5000ms

Szenariobeschreibung

5 Writer publizieren parallel je 20 Meldungen auf das verteilte Web Bulletin Board.

Resultat

Die Publikation aller 100 Meldungen dauert ungefähr **50 Sekunden**. Dies ergibt eine Durchschnittliche Publikationsdauer von **500ms** pro Meldung. Verglichen mit der Publikation einer einzelnen Meldung auf dem System ist dies eine Verdoppelung der Zeit. Dies ist auf den Aufbau der Testumgebung zurückzuführen, welcher durch die Virtualisierung der einzelnen Server ans Limit der Rechenkapazität gelangt. Jede einzelne Komponente auf einem eigens dedizierten Server laufen zu lassen würde hier sicherlich Abhilfe schaffen.

Messwerte

Zeilenbeschriftungen	Mittelwert [ms]	Maximum [ms]	Minimum [ms]
BoardBean	**1039**	**10328**	**0**
getEntries-ch.hsr.wbb.api.WbbQuery	0	0	0
getHistories	0	0	0
getSupportedContentTypes	0	0	0
notifyPublish-java.lang.String	375	5109	0
readAboveHash-java.lang.String-ch.hsr.wbb.crypto.Hash	0	0	0
readForAppending-java.lang.String	74	1750	0
read-java.lang.String	0	0	0
write-java.lang.String-ch.hsr.wbb.messages.WriteRequest	2927	10328	0
JPAEntryPersisterBean	**25**	**532**	**0**
getEntries	0	0	0
getEntries-ch.hsr.wbb.api.WbbQuery	0	0	0
getEntries-java.lang.String	0	0	0
getEntries-java.lang.String-ch.hsr.wbb.crypto.Hash-int	14	93	0
getTopHash-java.lang.String	9	172	0
init-ch.hsr.wbb.messages.ContentDefinition	0	0	0
save-ch.hsr.wbb.messages.HistoryEntry	45	532	0
Gesamtergebnis	**633**	**10328**	**0**

6 Ausblick / Offene Punkte

Nicht alles, was anfangs gedacht war, konnte in dieser Arbeit umgesetzt werden. In dieses Kapitel sind Ansätze enthalten, welche nicht vollständig umgesetzt worden sind. Die Erklärung dafür ist unter „Projektmanagement“ (Seite 69) enthalten.

6.1 Administration / Konfiguration

Die einzelnen Komponenten sollen von extern während der Laufzeit konfiguriert werden. Der Board-Host kann aufgrund von Erkenntnissen welche er aus dem Monitoring gewonnen hat, die Einstellungen in der Applikation anpassen.

Die einzelnen Komponenten können in den einzelnen EJB-Applikationskonfiguration konfiguriert werden. Darin enthalten sind die für die Komponenten unterschiedlichen veränderbare Parameter enthalten.

Alle Komponenten

Schlüssel	
crypto.hashAlgorithm	Algorithmus welcher für die Berechnung des Hash-Wertes verwendet wird. Z.B. MD2, MD5, SHA-1, SHA-256, SHA-384, SHA-512
crypto.signatureAlgorithm	Algorithmus welcher für die Berechnung und Validierung der Signaturen verwendet wird. Z.B. RSA
connections	Verbindungsinformationen zu den anderen Komponenten im WBB System.
keys	Schlüssel für die Erstellung der eigenen Signaturen sowie für die Validierung der Signaturen von anderen Komponenten.

Board-Spezifisch

Schlüssel	
board.nrOfHistories	Anzahl verwendeter Histories für das Board
board.thresholdWaitTimeout	Zeit in Millisekunden bis ein Timeout ausgelöst wird bei dem Versuch ein Threshold Signatur zu erstellen.
board.maxReadCount	Anzahl Meldungen, welche maximal bei dem Aufruf der read-Operation zurückgeliefert werden.

6.2 Monitoring

Unter dem Aspekt des Monitorings wird vorgesehen, dass die einzelne Komponente innerhalb eines WBB Systems von aussen auf dessen Zustand überprüft werden kann. Probleme während dem laufenden Betrieb können somit erkannt werden und eventuelle Massnahmen ergriffen werden. Die Monitoring-Daten müssen somit für den Host der Komponente zugänglich und verständlich aufbereitet sein.

6.2.1 Analyse

Aufgrund der Anforderung, dass die Monitoring-Daten auch von extern einsehbar sein müssen, kommen mehrere Varianten in Frage. Eine mögliche ist, dass die Daten über eine Weboberfläche von der Komponente selber visualisiert werden. Die zweite Möglichkeit ist Entwicklung einer Desktop-Applikation, welche über eine Schnittstelle mit den einzelnen Komponenten kommuniziert.

Für die Administration und das Monitoring von Java-Applikationen wird vielfach die JMX-Schnittstelle verwendet. JMX ist ein erweiterbares Tool welches erlaubt über eine spezifizierte Schnittstelle die Administrations- und Monitoring-Daten über ein Netzwerk auszutauschen. Weiter gibt es Tools, wie z.B. *jsconsole*, die eine rudimentäre Darstellung der gelieferten Daten unterstützten. Diese Tools sind jedoch recht technisch aufgebaut und erfordern fundiertes Wissen über die überwachte Applikation, um die Resultate richtig zu deuten. Deshalb wurde es als erforderlich erachtet, die gelieferten Daten in einer separaten Applikation zu visualisieren. Dadurch kann mehr auf die für die Applikation relevanten Daten eingegangen werden.

Um eigene Daten über die JMX-Schnittstelle zu liefern ist die Implementation eines eigenen, sogenannten MXBean‘s oder MBean‘s nötig. Auf der Seite der Client-Applikation werden aus dem Framework heraus Funktionen angeboten, welche die einfache Konsumation der von den MXBean gelieferten Daten anbieten.

Der GlassFish Server selber bietet ebenfalls MBean‘s an, welche Daten über den Zustand der einzelnen Komponenten liefern.

Mit der Portierung der Applikation zu J2EE Komponenten können Interceptors verwendet werden. Das Konzept der Interceptors erlaubt es Methodenaufrufe auf Business-Objekte abzufangen. Somit können Daten welche für das Monitoring verwendet werden in eigens implementierten Interceptors generiert werden.

Im Weiteren wurden die relevanten Zähler für die einzelnen Komponenten analysiert.

Writer

- Publikationsintervall
- Auslastung
- Informationen über die Publikationsdauer
- Fehlgeschlagene Schreibvorgänge auf Board n

Board

- Timeouts während der Erstellung der Threshold-Signatur
- Meldungen pro Historie
- Auslastung der einzelnen Histories
- Durchschnittliche Zeit für Kommunikation mit Board n
- Versuche des Writers auf eine blockierte Historie zu schreiben

6.2.2 Umsetzung

Der verwendete J2EE Applikationsserver (GlassFish) verfügt schon selber über in JMX-Schnittstelle. Über diese können Monitoring Daten ausgelesen werden. Das Monitoring kann in der GlassFish Administrationsoberfläche oder über das Administrations-Tool *asadmin* für die verschiedenen Komponenten eingeschaltet werden. Danach können über ein Tool wie die *jconsole* die Monitoring-Daten ausgelesen werden. Ebenso sind die Daten über die Administrationsoberfläche des GlassFish ersichtlich. Nützlich sind vor allem die verfügbaren Monitoring-Daten über die einzelnen Bean-Methoden, welche eine Aussage über die Verarbeitungszeit geben können.

Weitere Informationen über die Monitoring Unterstützung im GlassFish-Server kann aus dessen Dokumentation entnommen werden.

Mit genügend Kenntnissen über die vorliegende Arbeit können auch die geschrieben Log-Meldungen für die Behebung von Störungen oder zu Monitoring-Zwecken verwendet werden.

Für die Performance-Tests wurde in dieser Arbeit ein separates Projekt angelegt, in welchem eine Implementation vorliegt um die Ausführungszeiten der Methoden auf den Board-Komponenten zu analysieren. Die Implementation verwendet dazu die JMX Schnittstelle auf dem GlassFish.

7 Anleitungen

7.1 Deployment

In dieser Anleitung wird erläutert wie ein verteiltes Web Bulletin Board System in der Demonstrationsvariante unter einem Windows Betriebssystem von Grund auf aufgesetzt werden kann. Es wird davon ausgegangen, dass für jede Komponente des verteilten Web Bulletin Board ein eigener GlassFish Applikationsserver und ein eigener Datenbankserver zur Verfügung steht. Analog ist es jedoch auch möglich mehrere Komponenten auf dem gleichen Server laufen zu lassen.

Die folgenden Komponenten werden für das Deployment benötigt.

- Java Developement Kit 6 (http://www.oracle.com)
- Apache ANT (http://ant.apache.org/)
- GlassFish Server Open Source Edition 3.1 (http://glassfish.java.net/)
- Postgres SQL – Driver (http://www.postgresql.org/)

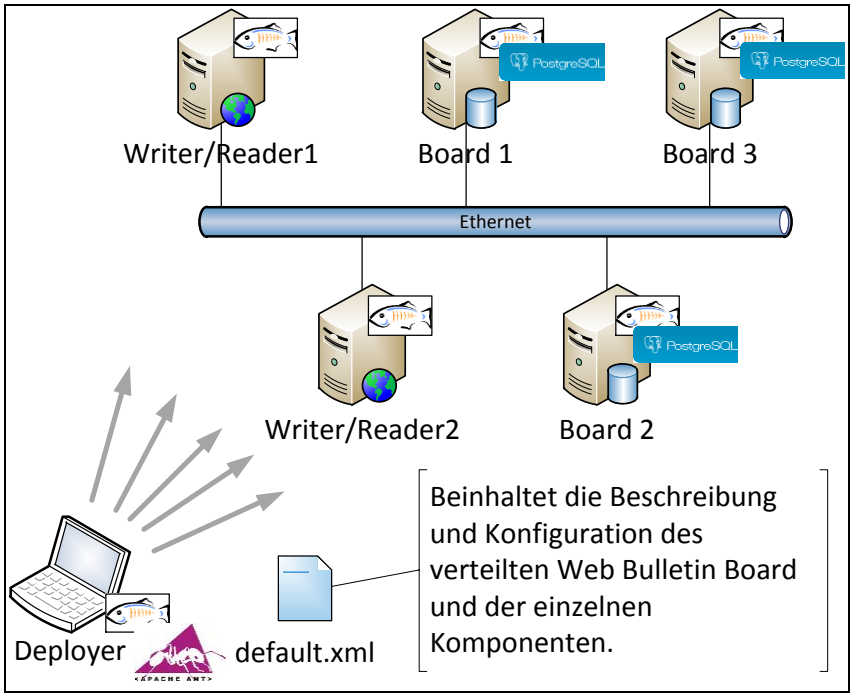
Die folgenden Komponenten werden für das Betreiben eines Boards benötigt.

- Java Developement Kit 6 (http://www.oracle.com)
- GlassFish Server Open Source Edition 3.1 (http://glassfish.java.net/)
- Postgres Datenbankserver (http://www.postgresql.org/)
- Postgres SQL – Driver (http://www.postgresql.org/) (Muss ins lib-Verzeichnis des GlassFish Server gelegt werden)

Die folgenden Komponenten werden für das Betreiben eines Writers benötigt.

- Java Developement Kit 6 (http://www.oracle.com)
- GlassFish Server Open Source Edition 3.1 (http://glassfish.java.net/)

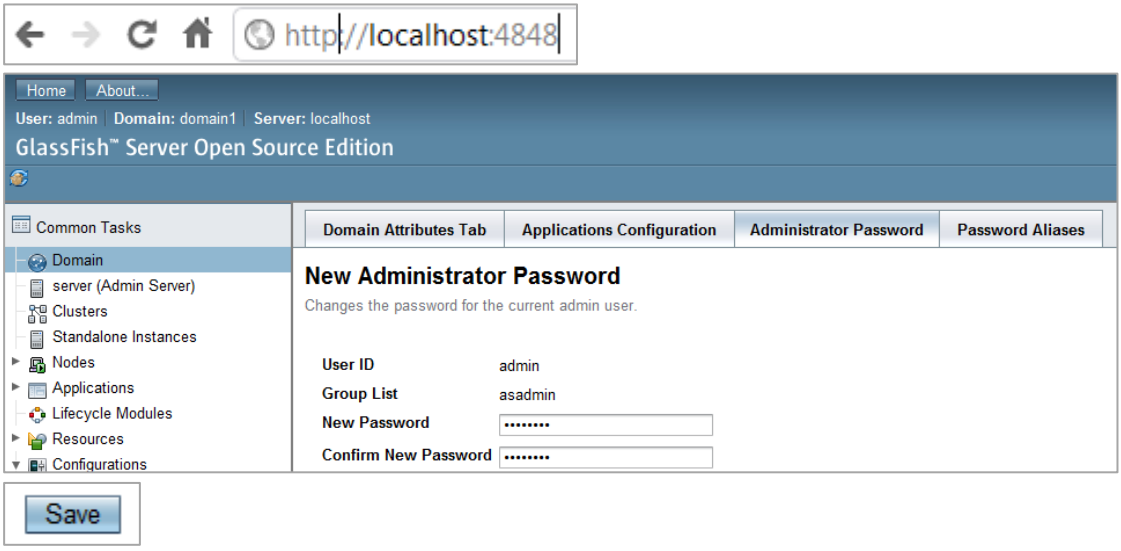
Im Verlauf der weiteren Anleitung wird vorausgesetzt, dass die benötigten GlassFish J2EE Applikationsserver (Server Open Source Edition) und Postgres Datenbankserver korrekt installiert sind. Für das automatische Deployment ist es zudem erforderlich, dass auch auf dem Computer von dem aus das Deployment ausgeführt wird, ein GlassFish Applikationsserver installiert ist und dass dieser Computer auch eine aktive Internetverbindung besitzt.



Benötigte GlassFish Voreinstellungen

Folgende Einstellungen müssen auf einer laufenden Glassfish-Installation getätigt werden um das automatisierte Deployment zu verwenden:

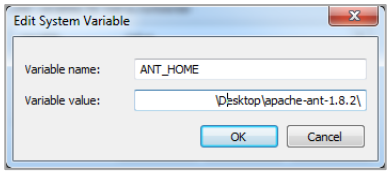
1. Administrator Password für Domain setzen



2. Programm Asadmin in einem CommandLine Tool starten (Im Installationsverzeichnis des GlassFish Applikationsserver z.B. *C:\glassfish3\bin>asad*min)
   Danach müssen folgende Befehle ausgeführt werden:
   - login *(admin / neu gesetztes Passwort)*
   - enable-secure-admin
   - restart-domain

Anleitung

Folgende Anleitung setzt ein Web Bulletin Board System mit den Standardeinstellungen auf. Vor dem Ausführen des build Skript, sollte vergewissert werden, dass der GlassFish mit dem gegebenen Login und Passwort auch erreichbar ist. Ebenso die Posgres Datenbank.

1. build.properties anpassen • GlassFish um Applikation zu builden	wbb.glassfish=C:\\glassfishv3\\glassfish\\
2. Systemvariable ‚ANT_HOME' setzen	

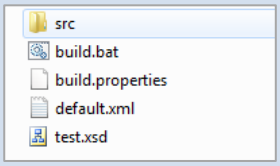
3. Content-Typen mittels XML Schema definieren

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://votes.hsr.ch/parliament"
    xmlns:tns="http://votes.hsr.ch/parliament"
    elementFormDefault="qualified">

    <complexType name="Person">
        <attribute name="firstName" type="string" />
        <attribute name="familyName" type="string" />
        <attribute name="birthDate" type="date" />
        <attribute name="profession" type="string" />
    </complexType>

    <element name="person" type="tns:Person" />
</schema>
```

test.xsd



4. WBB – Umgebung in default.xml definieren
   - App-Server
   - DB-Server
   - Komponenten (Boards (Name = Bx), Writers (Name="Wx"))
   - XML-Schemas der Content-Typen einbinden

Bsp. Applikationsserver

```
<applicationServer name="wbb10"
    host="192.168.10.10" port="8080">
    <administration port="4848" user="admin"
    password="pw" />
</applicationServer>
```

Bsp. Datenbankserver

```
<databaseServer name="wbb10"
    host="192.168.10.10" port="5432">
    <administration user="postgres"
          password="pw" />
    <database user="wbb" password="pw" />
</databaseServer>
```

Bsp. Board

```
<board name="B0">
    <database server="wbb10" database="wbb0"/>
    <application server="wbb10"
          contextRoot="B0" />
</board>
```

Bsp. Writer

```
<writer name="W0">
    <application server="wbb11"
          contextRoot="W0" />
</writer>
```

Bsp. XML-Schemas einbinden

```
<schemas>
    <schema location="a.xsd" />
    <schema location="b.xsd" />
    <schema location="c.xsd" />
</schemas>
```

5. Ausführen ‚build.bat'

Im tmp Verzeichnis sind alle generierten Dateien enthalten. Darin finden sich die ear-Dateien welche für ein

	manuelles Deployment verwendet werden können. Die connections.properties Datei kann für die Board API verwendet werden. Ebenso kann aus dem keys Ordner ein keyFile.ini von einem Writer für die API genommen werden.

7.1.1 Deployment-Tasks

Um die unten angegeben Tasks auszuführen können sie als optionale Parameter dem build.bat Skript angegeben werden. Standardmässig wird der deploy_all Task ausgeführt.

deploy_all *[default]*	Führt das Deployment für alle Komponenten aus, welche in der Deployment-Beschreibung enthalten sind.
undeploy_all	Entfernt alle Komponenten, welche in der Deployment-Beschreibung enthalten sind.
all_in_one	Führt sequentiell zuerst den undeploy_all, clean, deploy_all Task aus. Dieser Task kann verwendet werden um neue Änderungen auf das System zu verteilen.
clean	Entfernt alle Dateien welche durch das Deployment-Skript erstellt wurden.
build_all	Erstellt alle J2EE Application-Files. Mit diesen kann danach das Deployment manuell auf einen Application-Server durchgeführt werden.

7.2 Benutzeroberflächen

Zur Demonstration des Funktionsumfangs eines verteilten Web Bulletin Board wurde für alle Komponenten eine Benutzeroberfläche ausgearbeitet, welche über einen Webbrowser aufgerufen werden kann. So können über die Weboberflächen des Boards die Meldungen eingesehen werden, welche auf dem jeweiligen Board abgespeichert sind.

Grundsätzlich ist es vorgesehen, dass einzig die Boards auf einem J2EE Applikationsserver laufen. Die Writer und Reader Komponenten werden jeweils in die Endanwendung integriert, welche die zur Verfügung gestellte API des Web Bulletin Board benutzen. Zu Demo-Zwecken werden jedoch beim automatisierten Deployment auch die Writer Komponenten auf einen Applikationsserver installiert. Diese beinhalten sowohl die Writer- als auch die Reader-Funktionalitäten.

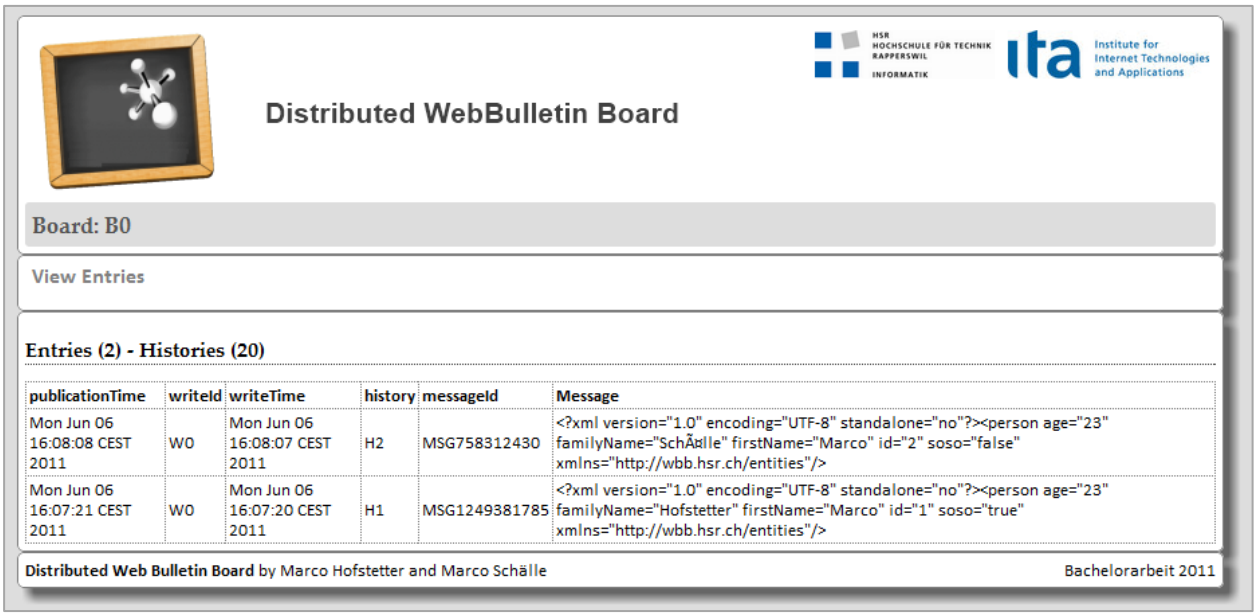
Zu den Writer-Funktionalitäten zählt, dass neue Nachrichten auf dem verteilten Web Bulletin Board publiziert werden können, während bei dem Reader einzelne Quittungen auf ihre Gültigkeit geprüft werden, aber auch alle Boards validiert werden können. Zusätzlich können Abfragen mit optionalen Filterkriterien abgesetzt werden. Diese liefert als Resultat alle Meldungen im System, welche dem Filter entsprechen.

Die URL für den Zugriff auf die Weboberflächen setzt sich wie folgt zusammen:

> URL = ‚IP vom J2EE -Server‘:‘Default-HTTP-Port von Glassfish‘/‘Gesetzer ContextRoot der Komponente‘/
> *Bsp: 192.168.10.10:8080/B0/*

7.2.1 Board Benutzeroberfläche

Über die Benutzeroberfläche eines Boards können die auf dem jeweiligen Board publizierten Meldungen eingesehen werden. Zu jeder Meldung werden die Publikationszeit auf dem Board, die ID des Writer der die Meldung publiziert hat, die Erstellungszeit beim Writer, die Historie auf der die Meldung gespeichert ist, die ID der Meldung selbst und den eigentlichen Inhalt der Meldung als XML angezeigt.



publicationTime	writeId	writeTime	history	messageId	Message
Mon Jun 06 16:08:08 CEST 2011	W0	Mon Jun 06 16:08:07 CEST 2011	H2	MSG758312430	<?xml version="1.0" encoding="UTF-8" standalone="no"?><person age="23" familyName="SchÃ¤lle" firstName="Marco" id="2" soso="false" xmlns="http://wbb.hsr.ch/entities"/>
Mon Jun 06 16:07:21 CEST 2011	W0	Mon Jun 06 16:07:20 CEST 2011	H1	MSG1249381785	<?xml version="1.0" encoding="UTF-8" standalone="no"?><person age="23" familyName="Hofstetter" firstName="Marco" id="1" soso="true" xmlns="http://wbb.hsr.ch/entities"/>

7.2.2 Demo Writer/Reader Benutzeroberfläche

Zu Demozwecken wurde eine Testoberfläche geschrieben, welche die WBB-API anspricht. Diese demonstriert sowohl die Funktionalität eines Writers als auch die eines Readers.

Writer Funktionalität

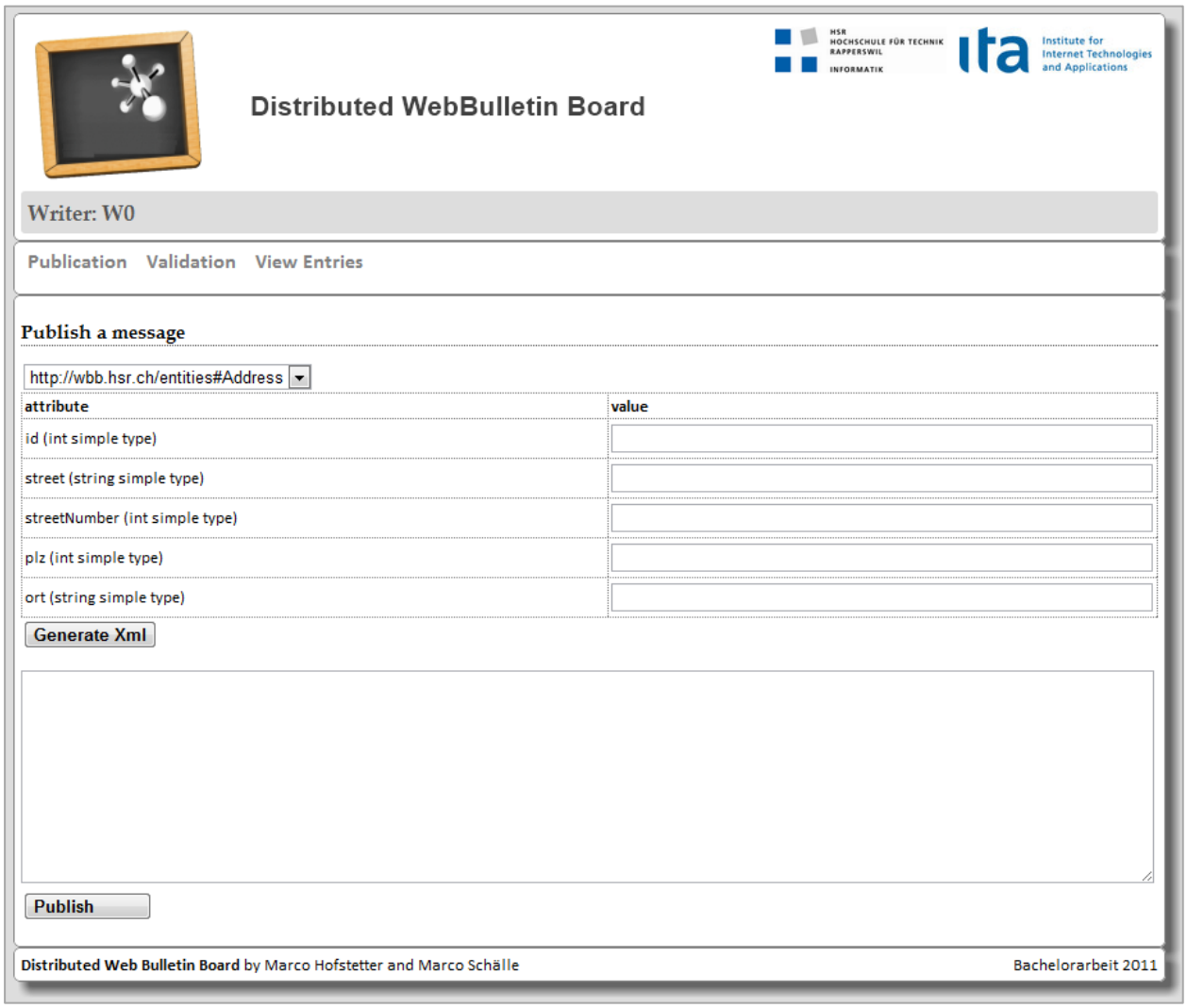
- Register ‚*Publication*'

Reader Funktionalität

- Register ‚*Validation*'
- Register ‚*View Entries*'

Writer Funktionalität

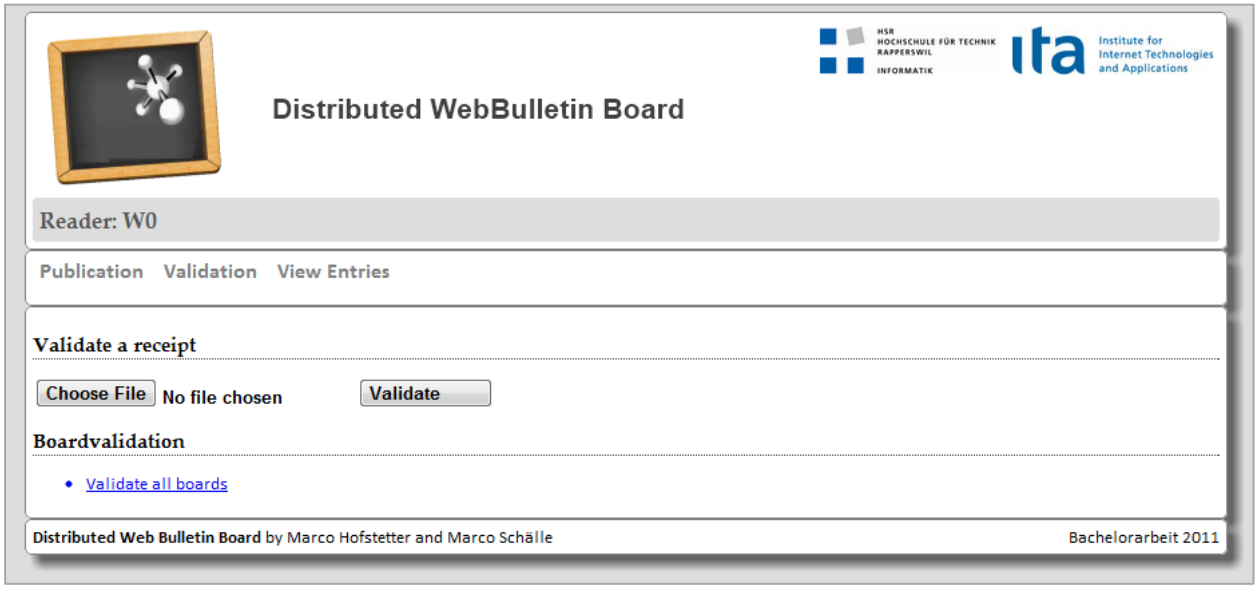
Über den Writer können neue Meldungen auf das verteilte Web Bulletin Board publiziert werden. Dabei wird der Inhalt einer Meldung grundsätzlich immer als XML repräsentiert. Die Weboberfläche bietet jedoch die Hilfe an, durch Auswahl des gewünschten Content-Typen die dazugehörigen Attribute auszufüllen. Danach kann der XML-Inhalt" generiert werden.



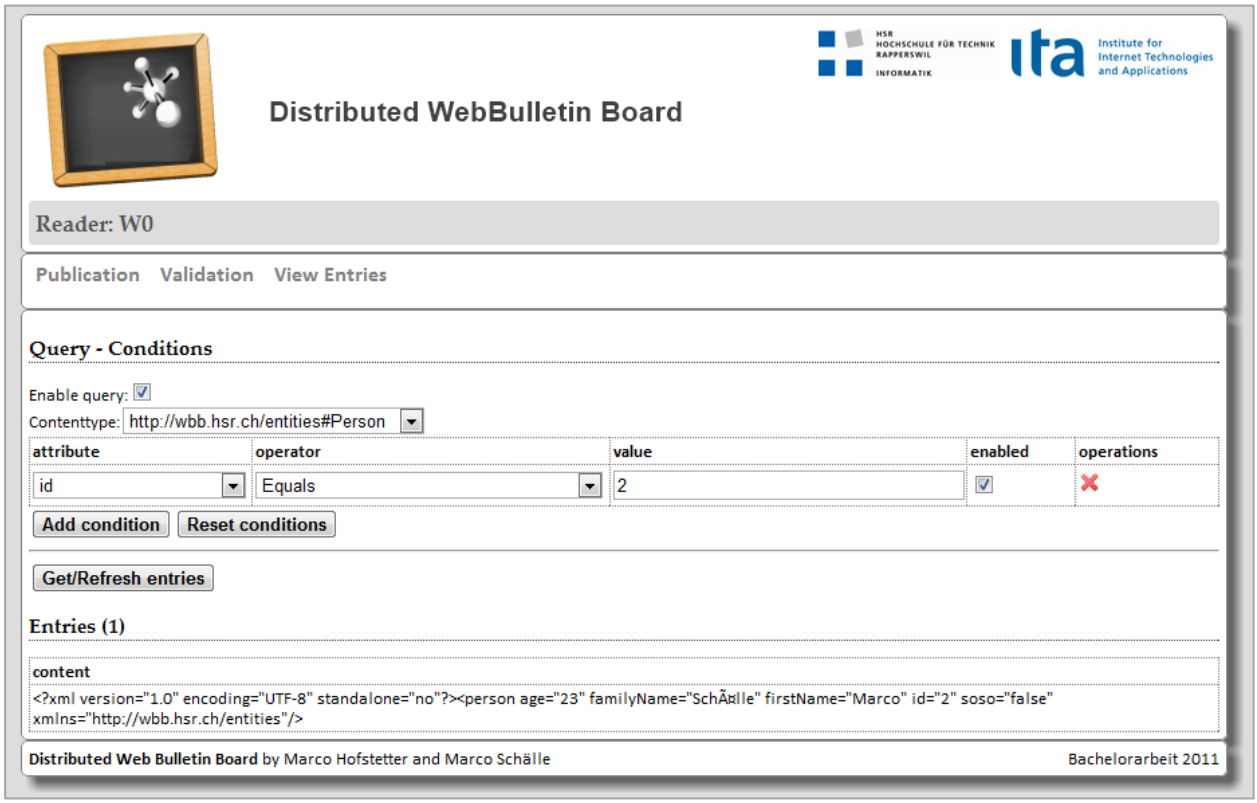
Nach erfolgreicher Publizierung einer Meldung besteht die Möglichkeit die Quittung lokal zu speichern. Dies erlaubt die spätere Validierung der Quittung – sprich, ob die dazugehörige Meldung noch unverändert auf den Web Bulletin Board gespeichert ist.

Reader Funktionalität

Über den Register ‚Validation' kann eine zuvor gespeicherte Quittung zu einer Meldung validiert werden. Dabei wird geprüft, ob die Meldung noch unverändert auf dem verteilten Web Bulletin Board gespeichert ist.
Es ist auch möglich alle Boards des WBB zu überprüfen. Dabei wird bei sämtlichen Meldungen die Korrektheit der Signatur, der Threshold Signatur, des Hashwertes (auch ob sie noch in der ursprünglichen Reihenfolge vorliegen) geprüft.



Über den Register ‚View Entries' können die auf dem Web Bulletin Board gespeicherten Meldungen abgefragt werden. Dabei besteht die Möglichkeit die Abfrage mit einem Filter zu versehen – den ‚Query-Conditions'. Es lässt sich damit nach einem bestimmten Content-Typen suchen. Es ist auch möglich nach Content-Typen mit bestimmten Attributwerten zu filtern.

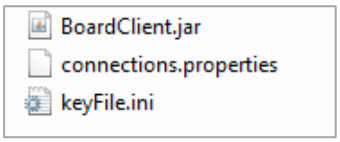


7.3 API

Einer Endanwendung welche ein verteiltes Web Bulletin Board zur verteilten und robusten Datenspeicherung verwenden will, steht eine Schnittstelle (API) zur Verfügung, um auf die Funktionen des WBB zuzugreifen. Sei dies das Publizieren eines neuen Eintrages, die Prüfung auf Korrektheit aller bestehenden Einträge oder das Abfragen nach spezifischen Einträgen.

Zur Verwendung der Schnittstelle benötigt man lediglich die Java-Library BoardClient.jar sowie die zwei Dateien „*connections.properties*“ und „*keyFile.ini*“. Diese beiden Dateien werden auch beim automatisierten Deployment generiert.

- *BoardClient.jar*: Bildet die eigentliche API ab
- *Connections.properties*: Die Endpunktangaben der Boards der WBB Umgebung mit welchen kommuniziert werden muss.
- *keyFile.ini*: Beinhaltet alle zur Prüfung der Signaturen benötigten öffentlichen RSA-Schlüssel der Kommuniktionspartner und den eigenen privaten RSA-Schlüssel zur Erstellung von Signaturen. Auch der öffentliche Schlüssel für die Threshold-Signatur ist enthalten.




```
B0=http://192.168.10.10:8080/B0/
B1=http://192.168.10.11:8080/B1/
B2=http://192.168.10.11:8080/B2/
B3=http://192.168.10.20:8080/B3/
B4=http://192.168.10.21:8080/B4/
B5=http://192.168.10.22:8080/B5/
```

connections.properties

```
keyConfig.boards.B0.publicKey.exponent=10001
keyConfig.boards.B0.publicKey.modulus=8e852e09ac69e86efcd7e1b6f6b58fa4dde56e
keyConfig.boards.B0.publicThresholdKey.position=0
keyConfig.boards.B0.publicThresholdKey.verificationKey=6b396c94e5188372b1e2a
...
keyConfig.writers.W1.publicKey.exponent=10001
keyConfig.writers.W1.publicKey.modulus=d0740112e23694d49c56580b5d5ddedefd42f
...
keyConfig.privateConfig.componentId.name=W0
keyConfig.privateConfig.componentId.type=WRITER
keyConfig.privateConfig.privateKey.exponent=4f7eab555c518d48ec379452c8dd9440
keyConfig.privateConfig.privateKey.modulus=abba16698af8108f53ffaac27ebeeaaa8
keyConfig.privateConfig.publicKey.exponent=10001
keyConfig.privateConfig.publicKey.modulus=abba16698af8108f53ffaac27ebeeaaa8e

keyConfig.thresholdInfo.keyLength=1024
keyConfig.thresholdInfo.numberOfAuthorities=6
keyConfig.thresholdInfo.publicKey.exponent=10001
keyConfig.thresholdInfo.publicKey.modulus=a9d1fe62835ab7a607b7d9bd307c6d94b1
keyConfig.thresholdInfo.publicKey.verificationKey=62ed7581
keyConfig.thresholdInfo.threshold=4
```

keyFile.ini

BoardClient

void init()	Initialisiert den BoardClient. Dabei werden die zwei Dateien 'connections.properties' und ‚keyFile.ini', welche für die Benutzung benötigt werden, im Ordner ‚META-INF' gesucht.
void init(InputStream, InputStream)	Initialisiert den BoardClient. Dabei besteht die Möglichkeit die zwei Einstellungsdateien via zwei InputStream's mitzugeben.
PublicationReceipt publish(Object)	Das Objekt wird innerhalb der WBB Umgebung gespeichert. Dabei ist wichtig dass es sich um einen den Boards bekannten Content-Typ handelt. Als Resultat liefert die Methode ein PublicationReceipt mit welchem später überprüft werden kann, dass der Eintrag noch unverändert auf den Boards gespeichert ist.
ValidationResult testReceipt(PublicationReceipt)	Mit dieser Methode lässt ein PublicationReceipt validieren. Es wird überprüft, dass die dazugehörige Meldung noch unverändert auf den entsprechenden Boards abgespeichert ist.
ValidationResult testBoards()	Es werden alle Meldungen auf allen Boards auf ihre Gültigkeit überprüft. Dabei werden parallel alle Einträge auf den einzelnen Histories auf die korrekte Signatur, Threshold-Signatur und Hash-Verkettung überprüft.
List<HistoryEntry> getAllEntries()	Liefert alle Einträge innerhalb der WBB Umgebung. Dabei wird jeder Eintrag nur einmal ausgegeben.
<T> getEntries(Class<T>, WbbQuery)	Liefert alle Einträge innerhalb der WBB Umgebung, welche die Conditions innerhalb des mitgegebenen Query erfüllen.
Void finish()	Beendet den BoardClient, wobei gebrauchte Ressourcen wieder freigegeben werden.

Beispiele

```
@XmlRootElement(namespace="http://wbb.hsr.ch/entities")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Person")
public class Person {

    @XmlAttribute
    protected Integer id;
    @XmlAttribute
    protected String firstName;
    @XmlAttribute
    protected String familyName;
    @XmlAttribute
    protected Integer age;
    @XmlAttribute
    protected Boolean soso;
```

```
// Initliaisieren
BoardClient client = new BoardClient();
client.init();

// Publizieren
Person person = new Person();
person.setFamilyName("Mouse");
person.setFirstName("Mickey");
person.setAge(24);
person.setSoso(false);
person.setId(6);
client.publish(person);

// Einträge abfragen
client.getAllEntries();

// Einträge via Query abfragen
WbbQuery query = new WbbQuery(Person.class)
    .addCondition("firstName", WbbQuery.EQ, "Mickey");

Collection<Person> results = client.getEntries(Person.class, query);

// Finish
client.finish();
```

8 Projektmanagement

Die Planung des Projekts wurde in Blöcke aufgeteilt, welche stichwortartig erklärt werden:

- Technische Analyse und Aufsetzen der Testumgebung
  - Das Aufsetzen der Testumgebung nahm relativ viel Zeit ein, hat sich jedoch für die Integrationstests ausgezahlt.
- Offene Punkte aus SA beheben
  - Ziel war bekannt und wurde im geplanten Zeitrahmen umgesetzt.
- Portierung auf J2EE
  - Mangelhafte Dokumentation des J2EE Applikationsserver-Standards erschwerte die Umsetzung
  - Konfigurationsprobleme sind aufgetreten
    - Anzahl gleichzeitiger HTTP-Requests
    - DB-Pool Konfiguration
    - Mehrere gleiche Applikationen auf gleicher Server-Instanz
    - Auslesen der Konfiguration
    - JNDI
- Content-Typen
  - Eingeschobene Analysephase bei Bekanntgabe der neuen Anforderung.
  - Erforderte eine Neuplanung des Projekts →Umsetzung „Monitoring“ und „Administration“ blieb aus

8.1 Projektplan

Wochenplanung	22.02-28.02	29.02-07.03	08.03-14.03	15.03-21.03	22.03-28.03	29.03-04.04	05.04-11.04	12.04-18.04	19.04-25.04	26.04-02.05	03.05-09.05	10.04-16.05	17.05-23.05	24.05-30.05	31.05-06.06	07.06-13.06	14.06-17.06
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
Einarbeitung	10	65															
Technische Analyse	40		20														
Offene Punkte aus SA																	
Zweifache Erstellung von Hash			50														
MessageWaitSet‘s werden nicht gelöscht					20												
Writer blockiert					5												
Identifikation der Nachricht als Ursache für Probleme					10												
Component prüfen bei Methodeneintritt					10												
Readerimplementation verbessern					25												
Portierung auf J2EE																	

Projektstruktur aufsetzen						30											
Komponenten als EJB						40	40	10									
Pseudo DI durch DI von Container ersetzen							20	10									
Webservice-Schnittstelle						10		20									
Komponenten Unittests portieren							10										
Quittungsausstellung																	
Analyse/Konzept							20										
Implementatoin Ausstellen einer Quittung								20									
Implementation Validieren einer Quittung									20								
ContentTypes																	
Analyse/Konzept				80	20												
Implementation										50	40	20	40				
Monitoring JMX																	
Konzept für Monitoring/Counters									10								
JMX Bean implementieren	**Implementation aus Zeitgründen und der Priorität der ContentTypes ausgelassen!**																
Monitoring Client																	
Administration																	
Konzept Administration												20					
Administrations Client	**Wurde mit den Web-UIs umgesetzt**																
Automatisiertes Deployment																	
Konzept										30							
Implementation											40	30	60				
Web-Uis																	
Mockups								30									
Board-UI Implementation									10					20			
Writer-UI Implementatoin									20					20			
Reader-UI Implementation									20					10			
Testing																	
Unittest			10	10													
Integrationstest							10			10				50			
Aufsetzen Testumgebung	50	25				10											

Projektmanagement																
Installationsanleitung														20		
Dokumentation		10	20	10	10	10		10	20	10	20	30		60	80	70
Vortrag																30
Poster / Abstract														20	20	
Milestones					1			2			3			4		5

Milestones

M1: Offene Punkte aus SA behoben

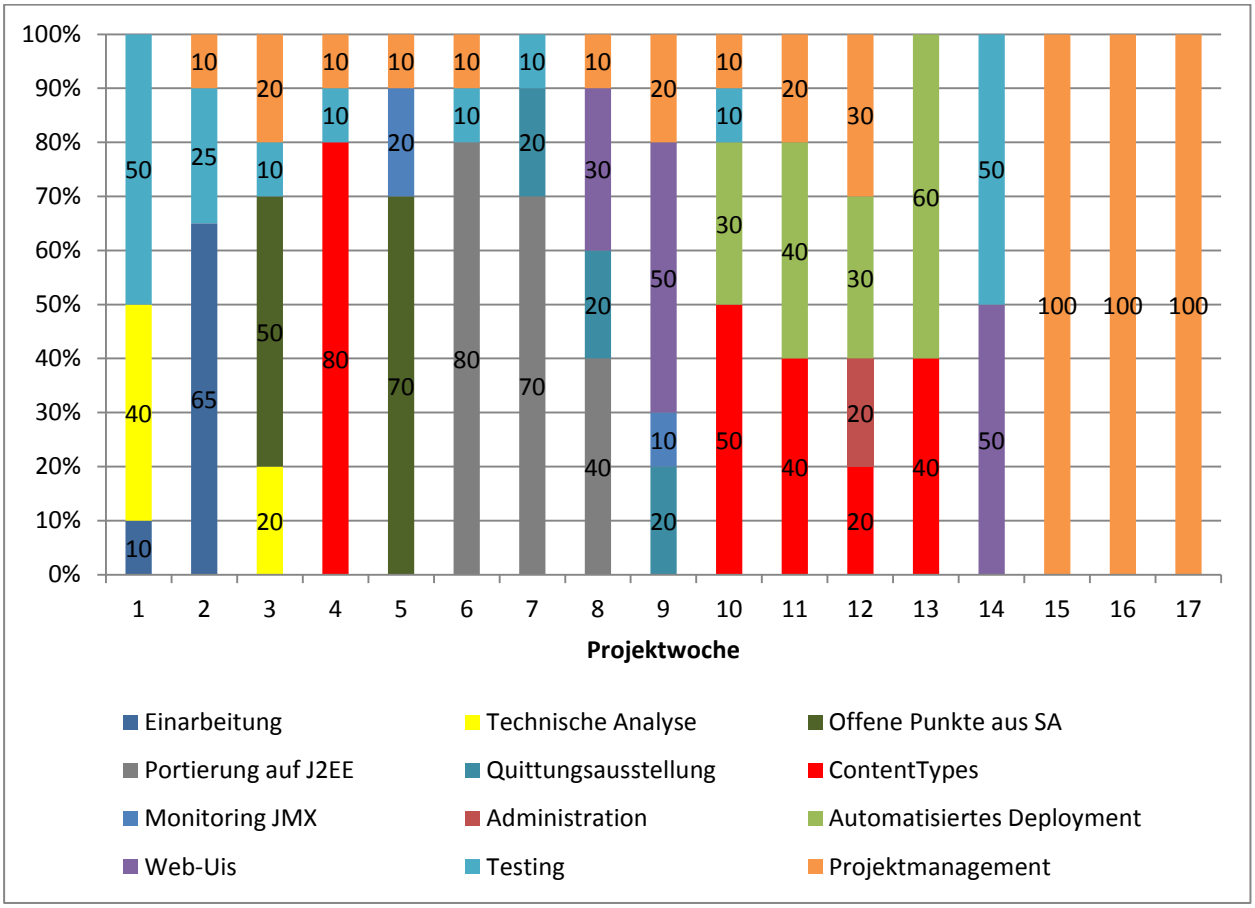
M2: Komponenten auf J2EE portiert / Kommunikation über Netzwerk funktioniert

M3: Umsetzung der Content-Typen abgeschlossen

M4: Automatisiertes Deployment und Benutzeroberflächen funktionieren

M5: Abgabe

Verteilung der Arbeitsthemen auf die Projektwochen



8.2 Erfahrungsbericht Marco Hofstetter

Projektverlauf

Da die Bachelorarbeit eine Folgearbeit unserer eigenen Semesterarbeit war, fiel die fachliche Analyse zu Beginn der Arbeit nicht mehr so gross aus. Sie wich einer technischen Analyse, in welcher wir die richtigen Frameworks und Tools für unsere Arbeit evaluierten. Wir wussten von Anfang an ziemlich genau, was wir innerhalb der Bachelorarbeit erreichen wollten. So wurde die Umsetzung der WBB Komponenten als J2EE Applikationen eine ziemlich zeitintensive Angelegenheit, bis das Ergebnis so aussah, wie wir uns dies vorgestellt hatten. Als grösste Schwierigkeit erwies sich, die mangelhafte Dokumentation des GlassFish Applikationsserver und der ganzen J2EE Technologie im Allgemeinen zu interpretieren. Auch das anfängliche Aufsetzen einer Testumgebung im Labor verschlang ziemlich viel Zeit.

Das unvorhergesehene Auftauchen der neuen Anforderung, publizierte Einträge auf dem WBB mit Filtern wieder abzufragen, warf unsere Planung kurzerhand wieder um. Einerseits mussten vorgesehene Themen aus zeitlichen Gründen gestrichen werden, andererseits musste nochmals eine Analyse im laufenden Projekt vorgenommen werden. Die Umsetzung der neuen Anforderung war eine Herausforderung und technisch auch spannend, dennoch waren wir bis zum Schluss nicht ganz davon überzogen, ob die Anforderung auch wirklich ins Konzept des Web Bulletin Board passt.

Während des ganzen Projekts war es mitunter ein wichtiger Punkt, dass die entwickelte Implementation auch möglichst effizient verwendet werden kann. So floss ein beachtlicher Anteil der ganzen Projektzeit in das automatisierte Deployment. Was auf den ersten Blick als nebensächlich angesehen werden könnte, erweist sich beim näheren Hinsehen jedoch als unabdingbares Hilfsmittel um ein verteiltes Web Bulletin Board aufzusetzen und effiziente und regelmässig durchgeführte Integrationstests zu ermöglichen.

Arbeit im Team

Das Arbeiten im Team hat gut funktioniert. Durch die fixen Arbeitszeiten bestand genügend Zeit, wichtige Fragen gemeinsam auszudiskutieren und den weiteren Verlauf des Projekts zu planen. So wusste jeder, womit der andere jeweils gerade beschäftigt war. Wenn möglich wurden die Arbeitspakete auch nach dem Interesse der einzelnen Teammitglieder aufgeteilt. Auch die Zusammenarbeit mit dem Betreuer und externen Personen verlief einwandfrei.

Fazit

Als grösste Schwierigkeit der Bachelorarbeit stellte sich heraus, die Begeisterung aus der Semesterarbeit beizubehalten. Was anfänglich nach einer reinen Portierung der WBB-Simulation aus der Semesterarbeit, hin zu einer J2EE Applikation aussah, stellte sich rasch als technische Herausforderung heraus. Es ging so weit, dass ich zwischendurch manchmal das Gefühl hatte, man würde sich in den technischen Details verlieren – zumal die Umsetzung der J2EE Komponenten auch eine ziemlich aufwendige Konfigurationsangelegenheit ist. Es entstanden relativ schnell viele lose Enden, welche gegen Schluss hin jedoch zu einem, aus meinen Augen, guten Endergebnis gebündelt werden konnten.

8.3 Erfahrungsbericht Marco Schälle

Projektverlauf

Dadurch, dass wir bereits zuvor die vorhergehende Semesterarbeit zu diesem Thema realisiert hatten, mussten wir uns nicht mehr in die Thematik einarbeiten und konnten schnell mit der Arbeit loslegen. Gewisse Überlegungen hatten wir uns bereits schon nach dem Abschluss der Semesterarbeit gemacht. Trotzdem stellte sich die Portierung der Semesterarbeit auf einen J2EE Applikationsserver als nicht so einfach dar wie anfangs geglaubt. Wir hatten zwar beide die theoretischen Vorkenntnisse, um so etwas zu realisieren. Jedoch mussten wir merken, dass die Konzepte der J2EE Umgebung nicht immer 1:1 für unsere Lösung verwendet werden kann. Auch war es nicht einfach, eine Testumgebung aufzusetzen, welche unseren Anforderungen genügte.

Das automatisierte Deployment war nicht von Anfang in diesem Umfang geplant. Es lohnte sich jedoch mehr Zeit in das Deployment zu investieren. Die Testumgebung konnte somit schneller aktualisiert werden. Und somit wurden die Integrationstests für neue Funktionen regelmässiger durchführt.

Die Spezifikation für die Verwendung der Content-Typen stellte für uns keine Probleme dar. Die Umsetzung verlief aber nicht mehr so zügig. Wir benötigen viel Zeit für die Evaluation der besten Lösung. Viele Bibliotheken versprachen eine saubere Lösung, wie beispielsweise *EclipseLink MOXy.* Doch sobald wir die Lösung in unser Gesamtsystem integrieren wollten, mussten wir feststellen, dass nicht mehr alles so wunderbar funktionierte.

Teilweise verloren wir auch die Grundkonzepte eines verteilten Web Bulletin Board aus den Augen. So verbrauchten wir nach mir zu viel Zeit an den Content-Typen, obwohl dieses Konzept nicht dringend notwendig für ein WBB ist.

Arbeit im Team

Fix definierte Arbeitszeiten ermöglichten uns, dass wir gemeinsam am Projekt arbeiten konnten. Dies erleichterte vor allem die Kommunikation untereinander. Die zu vor definierte Arbeitspakete konnten wir gut auf unsere einzelnen Interessengebiete aufteilen. Wir ergänzten uns gut und konnten konstruktive Diskussionen führen. Es dauerte zwar teilweise lange, bis wir einen Konsens fanden, waren aber danach beide von der Lösung überzogen. Damit erreichten wir auch die von uns gewünschte Qualität.

Fazit

Es war schön zu sehen, dass wir die geleistete Arbeit aus dem vorausgehenden Projekt, wie erwartet in die vorliegende Arbeit übernehmen konnten. Dies zeigte mir, dass die gelernten theoretischen Konzepte aus dem Studium, auch in der Praxis funktionieren.

Der Stand des Projektes entspricht meinen Erwartungen. Zwar gibt es immer noch offene Punkte um das System zu 100% produktiv einzusetzen. Aber das Produkt ist soweit, dass man es testweise einsetzen kann.

9 Schlussfolgerung

Alle Anforderungen aus der Aufgabenstellung wurden umgesetzt. Es wurde sogar mehr umgesetzt als zu Beginn der Arbeit definiert war. So enthält das Endprodukt noch zusätzlich die Implementation der Content-Typen und das automatisierte Deployment eines kompletten verteilten Web Bulletin Board. Das System ist funktionsfähig und reagiert auf Störungen wie erwartet. In der Arbeit ist eine Spezifikation enthalten, welche es erlaubt eine eigene Komponente zu implementieren und diese in das bestehende System einzubeziehen. Auch hilft die Spezifikation um das System eines verteilten Web Bulletin Board besser zu verstehen. Aufgrund der Bemerkungen sollte es möglich sein, die resultierende Software zu verstehen und evtl. weiterzuentwickeln. Ebenso ist eine Web Service Schnittstelle spezifiziert, welche es erlaubt das System von anderen Plattformen zu nutzen.

Bezüglich Softwarequalität und der Stabilität des Systems im Allgemeinen ist es uns gelungen durch regelmässige Tests, seien dies Unit-Tests im Detail oder Integrationstests und Performance-Tests über das ganze verteilte Web Bulletin Board, ein konstant hohes Niveau zu erlangen und zu behalten. Auch hier ist zu erwähnen, dass dies für allfällige Erweiterungen sehr von Nutzen ist. Um die Integrationstest überhaupt erst regelmässig auszuführen, war ein effizientes Deployment des Systems nötig, um hier den Zeitaufwand nicht unnötig in die Höhe zu treiben. Obwohl das eigens dazu implementierte Deployment-Skript nicht unbedingt zur Umsetzung des Systems an sich gehört, war es mitunter ein sehr wichtiger Faktor um das ganze Projekt überhaupt erfolgreich durchzuführen.

Nach unserer Ansicht bedarf es aber trotzdem noch einigem Aufwand um die Software produktiv einzusetzen. So sind in der Spezifikation noch Punkte enthalten, welche nicht umgesetzt worden sind oder noch weitere Analyse voraussetzen. Dazu sollte auch eine realistische Endanwendung mit einbezogen werden um die tatsächlichen Anforderungen auch wirklich zu testen. Oft mussten wir bei der Implementation Annahmen treffen, da keine solche Endanwendung mit konkreten Anforderungen vorliegt. So kam es dann ja auch, dass wir uns zu Demonstrationszwecken eine Webapplikation entwickelten, um zumindest die Hauptfunktionalitäten eines verteilten Web Bulletin Board visuell zu demonstrieren. Dies verhalf uns jedoch auch während der Entwicklungszeit auf einfache Art und Weise schnell den Zustand einer Komponente einzusehen - sei dies zum Beispiel die Einsicht aller gespeicherten Meldungen auf einem Board, ohne dabei direkt mit einem Datenbank-Tool auf die Datenbank zugreifen zu müssen.

Beim Entscheid der Technologien sind wir immer noch überzogen, dass wir die richtige Wahl getroffen haben. Es war schwierig, aber natürlich auch spannend, sich in all die neuen Technologien einzuarbeiten. Wir beiden hatten bereits die theoretischen Vorkenntnisse um die Technologien zu verwenden, dennoch fehlten beide die Erfahrung in der Praxis.

Abschliessend kann man sagen, dass es uns gelungen ist, relativ viele unterschiedliche und komplexe Anforderungen gut und im Detail zu erfüllen. Dabei gelang es uns auch mit einer einfachen API die Komplexität des dahinterliegenden verteilten Web Bulletin Board für die benutzende Applikation zu kapseln und verbergen.

10 Glossar

Ant	Von Apache geschriebenes Werkzeug zum automatisierten Erzeugen von ausführbaren Computerprogrammen.
Board	Die Teilkomponente ‚Board' ist für die Persistenz der publizierten Meldungen verantwortlich.
Deployment	Deutsch Softwareverteilung; Prozess zur Installation von Software PC/Server in Betrieben
Eclipse	Integrierte Entwicklungsumgebung für die Programmiersprache Java
Endanwendung	Als Endanwendung wird eine Applikation bezeichnet, welche ein Web Bulletin Board System verwendet, um Daten abzulegen.
Historie	Meldungen werden innerhalb des Boards auf eine Historie gespeichert. Dabei kann ein Board mehrere Histories besitzen.
J2EE	Java Platform, Enterprise Edition; Spezifikation einer Softwarearchitektur
J2EE Application Server	Laufzeitumgebung für J2EE-Komponenten
JAXB	Java Architecture for XML Binding; Ermöglicht es Daten aus einer XML-Schema-Instanz an eine Instanz zu binden.
JAX-WS	Java API for XML - Web Services; API zum Erstellen von Webservices.
JNDI	Java Naming and Directory Interface; Ein Namesdienst liefert zu einem bestimmten Namen ein Objekt.
JPA	Java Persistence API; Vereinfachte Übertragung von Objekten. Ist in der J2EE Spezifikation enthalten.

Meldung	Meldungen werden vom Writer erstellt und auf den einzelnen Boards publiziert. Die Meldung wird innerhalb des Boards auf einer Historie gespeichert. Die Meldung beinhaltet einen Content welcher durch einen Content-Typen definiert wird.
Partielle Threshold-Signatur / Teilsignatur	Mehrere partielle Threshold-Signaturen ermöglichen die Erstellung einer verteilten Threshold-Signatur.
Reader	Die Teilkomponente ‚Reader' dient als Prüfkomponente innerhalb des Web Bulletin Board. Es kann alle Meldungen im System als auch einzelne Quittungen auf ihre Korrektheit hin überprüfen. Ebenfalls dient die Reader als Schnittstelle zur Abfrage nach Meldungen welche auf dem Web Bulletin Board publiziert sind.
SOAP	Simple Object Access Protocol; Netzwerkprotokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht werden können
Threshold	Minimale Anzahl Boards 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.
Threshold-Set	Eine Gruppe von Boards, auf welche eine Meldung m redundant publiziert wird.
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 Boards die an der Signatur teilnahm mindestens dem Threshold des entsprechenden Threshold-Sets entspricht.
Verteiltes Web Bulletin Board, Web Bulletin Board, WBB	Das System als solches betrachtet wird Verteiltes Web Bulletin Board genannt. Es besteht aus den Komponenten Board, Writer und Reader.
Writer	Die Teilkomponente ‚Writer' dient als Schnittstelle zur Erstellung von neuen Meldungen auf dem verteilten Web Bulletin Board. Sie ist zuständig für die parallele Publizierung der Meldungen auf mehreren Boards und die abschliessende Erstellung und Herausgabe der Quittung.
Wsdl	Web Services Description Language; Beschreibungssprache für Webservices. Definiert den Austausch der Nachrichten über XML.
XML	Extensible Markup Language; ist eine Sprache zur Darstellung strukturierter Daten in Form von Textdaten. Wird für den plattformunabhängigen Austausch von Daten verwendet.

11 Literaturverzeichnis

Heather, J., & Lundin, D. (2009). *The append only WBB.*

Hofstetter, M., & Schälle, M. (2010). *Studienarbeit: Vertrauenswürdiges und robustes Bulletin Board für E-Voting.*

Krummenacher, R. (2010). *Umsetzung eines Web-Bulletin-Boards.*

Shoup, V. (2000). *Practical Threshold Signatures.*

Steffen, A. (2010). *Aufgabenstellung Studienarbeit 2010 Vertrauenswürdiges und robustes Bulletin Board für E-Voting.*

12 Anhänge

- Sourcecode (Ordner ‚Source' auf CD)
- DeploymentSkript (Ordner ‚Deployment' auf CD)
- API (Ordner ‚API' auf CD)
- Poster
- Sitzungsprotokolle
- Literatur (Ordner ‚Literatur' auf CD)