

STUDIENARBEIT

SYNCHRONISATION VON BESICHTIGUNGSINFORMATIONEN

Jan Eelbo

Studiengang Informatik-Ingenieurwesen
Technische Universität Hamburg-Harburg

Betreuung

Prof. Dr. Joachim W. Schmidt
Arbeitsbereich Softwaresysteme
Technische Universität Hamburg-Harburg



Hamburg, im Juli 2003

Inhaltsverzeichnis

1	Einleitung	1
1.1	Ziel der Arbeit	2
1.2	Gliederung	2
2	Thematische Einführung	5
2.1	Portale	5
2.2	Die Plattform infoAsset Broker	8
2.2.1	Architektur und Konzepte	8
2.2.2	Objektmodell	9
2.2.3	Das WIPS IT1.1 Informationsportal.....	12
2.3	Das Synchronisationsmodell für die Plattform infoAsset Broker	12
2.3.1	Klassen des Synchronisationsmodells	12
2.3.2	Der Synchronisationsvorgang.....	15
3	Case Study : Synchronisation von Informationen für die Schiffsbesichtigung	19
3.1	Ausgangssituation.....	19
3.2	Anforderungen der Synchronisation.....	20
3.3	Anwendungsfälle	22
4	Entwurf und Erweiterung des Synchronisationsmodells	27
4.1	Konfliktfreiheit bei der Synchronisation	27
4.1.1	Untersuchung des Konfliktpotentials	28
4.1.2	Alternative 1 : Drei Portale und ein zentraler ID-Server.....	29
4.1.3	Alternative 2 : Drei Portale und lokale ID-Generatoren	30
4.1.4	Alternative 3 : Zwei Portale und Synchronisation durch Hauptverwaltung	32
4.1.5	Alternative 5 : Zwei Portale und Synchronisationsdaten per Email.....	32
4.1.6	Alternative 6 : Zwei Portale und verschieden Asset-Versionen.....	33
4.1.7	Entwurf einer Lösung für die Konfliktfreiheit	33
4.2	Lineare Versionierung von Schiffsdaten	35
4.3	Erweiterungen des Synchronisationsmodells	36
4.3.1	Definieren der Objekthülle	37
4.3.2	Erkennen von Zustandsänderungen korrespondierender Assets	38

5	Realisierung und Evaluation	41
5.1	Realisierung.....	41
5.1.1	Die Klasse ReferenceMapping.....	41
5.1.2	Realisierung des Synchronisationsprozesses.....	43
5.1.3	Erweiterungen	45
5.1.4	Speicherung von zwei Zeitstempeln.....	46
5.1.5	Anpassung des AssetSerializers	46
5.1.6	Sonderbehandlung für AssetListen	46
5.1.7	Implementierung der Klasse ShipUpdateRequest.....	48
5.1.8	Lineare Versionierung der Schiffsdaten.....	50
5.1.9	Die Benutzerschnittstelle.....	50
5.2	Evaluation.....	55
6	Zusammenfassung und Ausblick	57
6.1	Zusammenfassung	57
6.2	Ausblick.....	58
	Anhang A – Referenzbäume für alle Anwendungsfälle	61
	Anhang B – Klassendiagramme zum Besichtigungsprozess	65
	Anhang C – Entwicklungsumgebung	67
	Literaturverzeichnis	69

Abbildungsverzeichnis

2.1	Taxonomie von Portalen nach [Weg02]	6
2.2	Merkmale von Unternehmensportalen aus [Weg02]	7
2.3	Architektur-Übersicht der Plattform infoAsset Broker nach [Weg00]	9
2.4	Konzeptuelles Modell der Broker-Services nach [Weg02]	10
2.5	Generisches Objektmodell infoAsset Broker aus [Leh02]	11
2.6	Konzeptuelles Klassendiagramm – Searches im infoAsset Broker	11
2.7	Konzeptuelles Synchronisationsmodell aus [Leh02]	13
2.8	Konzeptuelles Klassendiagramm zum Synchronisationsvorgang aus [Leh02]	14
2.9	Informationsaustausch zwischen den Portalen aus [Leh02]	15
2.10	Abbilden korrespondierender Objekte durch die Mapping-Tabelle [Leh02]	16
2.11	Attribute der Klasse AssetSynchronisationState [Leh02]	16
2.12	Synchronisationsphase I zwischen den Portalen [Leh02]	17
2.13	Synchronisationsphase II zwischen den Portalen [Leh02]	18
3.1	Ausgangssituation	20
3.2	UML-Anwendungsfalldiagramm – Synchronisation	22
3.3	UML-Anwendungsfalldiagramm – Synchronisationseinstellungen	23
3.4	UML-Anwendungsfalldiagramm – Änderungen an Schiffsdaten bearbeiten	24
4.1	Drei Portale und zentraler ID-Server	29
4.2	Drei Portale und lokale ID-Generatoren	30
4.3	Mapping-Tabelle globale/lokale Asset-IDs	31
4.4	Konzeptuelles Klassendiagramm – ShipUpdateRequest	34
4.5	Lineare Verkettung der ShipUpdateRequests pro Schiff	35
4.6	Konzeptuelles Klassendiagramm – Lineare Versionierung von Schiffsdaten	36
4.7	Referenzbaum – Vorschriften	37
4.8	Beispiel – Erkennen von Zustandsänderungen	39
5.1	Objektdiagramm ReferenceMapping und HashMaps	42
5.2	UML-Objektdiagramm – Synchronisationslink	44
5.3	Konzeptuelles Klassendiagramm – AssetListen	47
5.4	Interface AssetURIReplacer	47
5.5	Klassendiagramm ShipUpdateRequest	48
5.6	Screenshot – Synchronisationseinstellungen	51
5.7	Screenshot – Besichtigungsauftrag vor der Besichtigung	52
5.8	Screenshot – Besichtigungsauftrag nach durchgeführter Besichtigung	53
5.9	Screenshot – Änderungsantrag	54
5.10	Screenshot – Bearbeitung eines Änderungsantrages	54

A.1	Referenzbaum – Besichtigungen	61
A.2	Teil-Referenzbaum – Schiff.....	62
A.3	Referenzbaum – Vorschriften	63
A.4	Referenzbaum – ShipUpdateRequests	63
B.1	Klassendiagramm Besichtigungen.....	65
B.2	Klassendiagramm Schiffsdaten.....	66

Tabellenverzeichnis

4.1	Konfliktuntersuchung	28
C.1	Systemeigenschaften der Entwicklungsumgebung	67
C.2	Entwicklungstools	67

Danksagung

Ich möchte mich für das interessante Thema und die gute Betreuung am Arbeitsbereich Softwaresysteme bedanken. Insbesondere danke ich Prof. W. Joachim Schmidt, Patrick Hupe und Axel Wienberg. Außerdem möchte ich dem Germanischen Lloyd für die Zusammenarbeit danken, insbesondere bedanke ich mich bei Herrn Uwe Langbecker.

Kapitel 1

Einleitung

Die Welt wird digital. Fast alle Informationen, die wir in unserem täglichen Leben benötigen, sind in irgendeiner Form schon digital verfügbar. Dies betrifft sowohl private wie auch kommerziell verwertbare Informationen. Das Internet ist dabei zur wohl wichtigsten Informationsquelle der Gesellschaft geworden. Die Hälfte aller Bundesbürger nutzt nach aktuellen Schätzungen das Internet (siehe dazu [Emnid]). In anderen Ländern ist diese Quote noch höher. Vor dem Hintergrund weiter steigender Nutzerzahlen wird es immer wichtiger, Informationen zu kategorisieren und strukturiert im Internet zur Verfügung zu stellen. Angetrieben wird dieser Prozess durch viele Unternehmen, die unternehmensspezifische Inhalte und Dienste ihren Kunden sowie Mitarbeitern und Interessenten zur Verfügung stellen wollen.

Eine Möglichkeit dies zu tun, bieten Informationsportale. Informationsportale stellen ihren Nutzern Inhalte zur Verfügung, die diese durch Nutzung eines Standard-Web-Browsers (z.B. Microsoft Internet Explorer [InEx] oder Netscape Navigator [NeNav]) abrufen können. Informationsportale können einen personalisierten Zugang bieten, so dass dem Nutzer gerade die auf seine Bedürfnisse zugeschnittenen Inhalte präsentiert werden. Viele Unternehmen setzen gerade diesen personalisierten Zugang für ihre Mitarbeiter ein, so dass diese schnell an die für ihre Arbeit wichtigen Informationen und Inhalte gelangen.

Im Rahmen des Forschungsprojekts WIPS IT1.1 (siehe auch [wips]) zwischen dem Arbeitsbereich Softwaresysteme der Technischen Universität Hamburg-Harburg ([sts] und [tuhh]) und dem Germanischen Lloyd ([gl]) ist ein solches Informationsportal erstellt worden, das WIPS IT1.1 Informationsportal (WIPS steht für „Wettbewerbsvorteile durch informationstechnische Produktsimulation im Schiffbau“). Der Germanische Lloyd ist ein Unternehmen, das die Einteilung von Schiffen in bestimmte Klassen vornimmt und allgemeine Zertifikate für die zugehörigen Klassen erteilt. Ohne eine gültige Klasseneinteilung ist der Betrieb eines Handelsschiffes (auch Kreuzfahrtschiffes) kaum möglich, da zahlreiche nationale und internationale Regelungen und Vorschriften eine Klasseneinteilung erfordern. Zur Einteilung in diese Klassen müssen die Schiffe in regelmäßigen Abständen von Mitarbeitern des Germanischen Lloyd, so genannten Schiffsbesichtigern, inspiziert werden.

Im Rahmen des Forschungsprojektes stellt das WIPS IT1.1 Informationsportal den Schiffsbesichtigern alle für diese Arbeit notwendigen Informationen (Besichtigungsinformationen) zur Verfügung (z.B. Besichtigungsaufträge, Schiffsdaten, Eignerdaten, bisherige Besichtigungen). Dabei arbeitet das zentrale WIPS Informationsportal in der Zentrale des Germanischen Lloyd in Hamburg und enthält sämtliche Informationen zu Schiffsbesichtigungen und Schiffen, die beim Germanischen Lloyd klassifiziert sind. Die Schiffsbesichtiger betreiben lokale Informationsportale an ihren jeweiligen Besichtigungsstandorten (verschiedene internationale Häfen), wobei jedoch die lokalen Informationsportale nur die Besichtigungsinformationen speichern, die für die Schiffsbesichtiger am jeweiligen Standort notwendig sind.

Um die Besichtigungsinformationen auf den lokalen und dem zentralen Portal abzugleichen, bzw. Änderungen (z.B. Ergebnisse einer Besichtigung) zu übermitteln, muss zwischen den Portalen eine Synchronisation stattfinden. Für diese Aufgabe soll im Rahmen dieser Studienarbeit eine Lösung entwickelt und realisiert werden.

1.1 Ziel der Arbeit

In der vorliegenden Arbeit soll eine Lösung zur Synchronisation der Besichtigungsinformationen zwischen den Informationsportalen entwickelt werden. Das Entwickeln einer Lösung umfasst auch ihre prototypische Realisierung inklusive einer Benutzerschnittstelle.

Lokal und zentral wird dabei das WIPS IT1.1 Informationsportal eingesetzt, welches auf der Portalplattform *infoAsset Broker* der Firma *infoAsset* ([iA]) basiert. Für den *infoAsset Broker* ist bereits ein generisches Synchronisationsmodell entwickelt worden, welches nach Möglichkeit auch in der vorliegenden Arbeit verwendet werden soll (siehe dazu auch [Leh02]).

Zur Entwicklung einer Lösung ist es notwendig, eine fachliche Anforderungsanalyse von Nutzungsszenarien durchzuführen, damit die Lösung den Bedürfnissen der späteren Nutzer (Schiffsbesichtiger) entspricht. Aus den gefundenen Anforderungen wird dann ein Entwurf erstellt und anschließend realisiert.

1.2 Gliederung

Kapitel 2 erläutert zunächst den Begriff der Portale mit dem Ziel, eine Bedeutung des Begriffs Informationsportal zu liefern. Anschließend wird eine Einführung in die technische Basis dieser Studienarbeit, nämlich die zu Grunde liegende Portalplattform *infoAsset Broker*, geliefert, sowie die Verbindung zum WIPS IT1.1 Informationsportal aufgezeigt. Zum Schluss wird das in [Leh02] entwickelte Synchronisationsmodell für den *infoAsset Broker* beschrieben.

Kapitel 3 behandelt die Anforderungsanalyse für die Synchronisation von Besichtigungsinformationen. Es werden die in Frage kommenden Anwendungsfälle beschrieben, sowie die Anforderungen zusammengetragen, die eine Lösung erfüllen sollte.

In Kapitel 4 wird darauf aufbauend eine konkrete Lösung entworfen. Dazu werden zunächst verschiedene Entwurfsalternativen aufgezeigt. Im Anschluss wird dann ein konkreter Entwurf präsentiert, sowie daraus resultierende Erweiterungen des existierenden Synchronisationsmodells besprochen.

Kapitel 5 behandelt die Realisierung des Entwurfs. Es behandelt die Implementierungsdetails sowie Änderungen an den Basisklassen des WIPS Informationsportals, die notwendig geworden sind. Außerdem wird in Kapitel 5 eine Evaluation der realisierten Lösung geliefert.

Zum Abschluss gibt Kapitel 6 eine Zusammenfassung über die Studienarbeit und beschreibt im Ausblick Erweiterungs- und Verbesserungsmöglichkeiten des realisierten Prototyps.

Kapitel 2

Thematische Einführung

In diesem Kapitel soll eine Einführung in die thematischen Grundlagen dieser Studienarbeit geliefert werden. Dazu wird im ersten Abschnitt der Begriff Portal und seine Bedeutung in der Informatik näher erläutert. Im zweiten Abschnitt wird eine kurze Einführung in die technische Plattform, die dieser Studienarbeit zugrunde liegt, geliefert. Hier handelt es sich um die Portalplattform *infoAsset Broker*. Im dritten und letzten Abschnitt wird eine Einführung in das für die Plattform infoAsset Broker entwickelte Synchronisationsmodell geliefert.

2.1 Portale

Dieser Abschnitt beschäftigt sich mit dem Begriff des *Portals*. Es wird eine kurze Einführung, sowie Übersicht über die bekannten Arten von Portalen geliefert. In [Dud00] wird ein Portal beschrieben als:

„[Haupt]eingang; [prunkvolles] Tor; auch EDV Website, die als Einstieg ins Internet dient“

Es ist sicher letztere Bedeutung (EDV Website), die den Begriff des *Portals* in der Informatik schon näher erläutert. Es existieren aber weitaus genauere Definitionen des *Portals*, aus denen die folgende in [KLT00] entnommen ist:

„Ein *Web-Portal* ist eine Website im World-Wide-Web, die Informationen aus verschiedenen, ausgewählten Quellen zusammenfasst und ihren Nutzern über einen Standard-Web-Browser einen (personalisierten) Zugang mittels Suche und/oder Navigation von Verzeichnisstrukturen bietet, gegebenenfalls ergänzt um redaktionellen Inhalt, Funktionalität zur Kommunikation und/oder Informationsverarbeitung.“

Der Begriff des *Web-Portals* soll im Rahmen dieser Arbeit synonym mit dem Begriff des *Portals* verwendet werden. Portale lassen sich in *vertikale* Portale und *horizontale* Portale trennen. In [KLT00] ist diese Unterscheidung recht allgemein definiert:

„Ein *horizontales* Web-Portal deckt einen weiten Bereich von Themen ab, während ein *vertikales* Web-Portal auf einen eingeschränkten Themenbereich in größerer Detailliertheit fokussiert.“

Das Standardbeispiel für ein *vertikales* Portal ist Amazon.de. Es ist spezialisiert auf den Verkauf von Büchern, Tonträgern und Software. Inzwischen allerdings versucht auch Amazon etwas „horizontaler“ zu werden. Dort werden mittlerweile auch Foto-, Elektronik-, Küchen- und Haushaltsartikel zum Verkauf angeboten. Typische Vertreter für *horizontale* Portale sind T-Online, AOL, Lycos, Yahoo oder MSN. Allen diesen Portalen ist gemeinsam, dass sie eine breite Zielgruppe ansprechen wollen, indem sie ein vielfältiges Themenangebot präsentieren. Aus der Unterscheidung von *vertikalen* und *horizontalen* Portalen ergibt sich die folgende in [Weg02] entwickelte Taxonomie von Portalen:

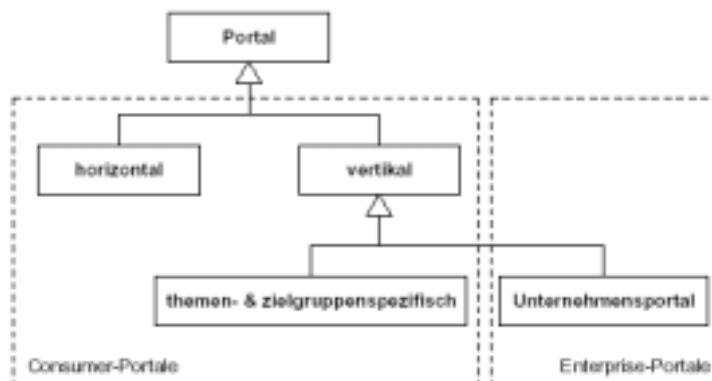


Abbildung 2.1: Taxonomie von Portalen nach [Weg02]

Danach wird in [Weg02] der Begriff des Unternehmensportals als spezielles vertikales Portal eingeführt, dessen Adressaten sowohl die Öffentlichkeit (Medien, Investoren) als auch Kunden, Partner und eigene Mitarbeiter des Unternehmens sind. Es wird auch hier synonym zu Enterprise-Portal gesehen. Die Definition wird wie in [Weg02] auch in dieser Arbeit aus [KLT00] übernommen:

„Ein *Unternehmensportal* ist eine Web- oder Intranet- oder Extranet-Site, die unternehmensrelevante Informationen aus verschiedenen, ausgewählten Internet-Quellen, Datenbanken, und anderen unternehmensspezifischen digitalen Quellen zusammenfasst und ihren Nutzern - einer eingeschränkten, mit dem Unternehmen verbundenen Zielgruppe – über einen Standard-Web-Browser oder spezielle Software einen personalisierten Zugang zu diesen Informationen mittels Suche und/oder Navigation von Verzeichnisstrukturen bietet, ergänzt um redaktionellen Inhalt, Funktionalität zur Kommunikation und Informationsverarbeitung.“

In [Weg02] wird die Funktionalität von Unternehmensportalen weiter differenziert, nach der Zielgruppe, die sie ansprechen. Danach gibt es vier verschiedenen Portalarten:

1. Allgemeine Unternehmens-Web-Sites, die der bloßen Information der Öffentlichkeit (Medien, Investoren und Interessenten) über das Unternehmen und der Vorstellung seiner Produkte oder Leistungen, sowie Kontaktmöglichkeiten dienen.
2. Kundenportale, die der engeren Bindung von Kunden – meistens Firmenkunden – an das Unternehmen dienen, um die Verkaufsprozesse zu unterstützen und zu verbessern. Sie bieten meist eine personalisierte Informationsversorgung. Als Schlagwort wird in [Weg02] hierfür Kundenbeziehungsmanagement (Customer Relationship Management, CRM) angeführt.
3. Partnerportale, die als Zielgruppe eigene Mitarbeiter des Einkaufs oder Verkaufs, etablierte Geschäftspartner und Zulieferer adressieren. Sie dienen der Unterstützung und Verbesserung von elektronisch unterstütztem Lieferketten-Management (Supply Chain Management, SCM), sowie von elektronischen Marktplätzen und Anwendungsvermietung (Application Service Providing, ASP).
4. Wissensportale, die zum Wissensmanagement dienen und als Zielgruppe die eigenen Mitarbeiter haben.

Das führt zu der folgenden Tabelle, die die verschiedenen Merkmale von Unternehmensportalen noch einmal gegenüberstellt:

	WebSite	Kundenportal	Partnerportal	Wissensportal
Paradigma	B2P	B2C	B2B	B2E, E2E
Medium	Internet	Internet, (Extranet)	Extranet, (Internet)	Intranet, (Extranet)
Zweck	Marketing, allg. Information	CRM, e-Commerce	SCM, ASP, e-Procurement	Wissensmanagement
Zielgruppe	Öffentlichkeit	Firmenkunden, Einzelkunden	Einkauf/Verkauf, Geschäftspartner, Zulieferer	Mitarbeiter
Orientierung	extern		intern	

Abbildung 2.2: Merkmale von Unternehmensportalen aus [Weg02]

Die Paradigmen aus Abbildung 2.2 bedeuten:

- *B2P*: Abkürzung für „Business to public“. Adressat dieser Art von Unternehmensportal ist die Öffentlichkeit.
- *B2C*: Abkürzung für „Business to consumer“. Adressaten dieser Art von Unternehmensportal sind die Kunden des Unternehmens.
- *B2B*: Abkürzung für „Business to business“. Adressaten dieser Art von Portalen sind andere Unternehmen.
- *B2E, E2E*: Abkürzungen für „Business to experts“ bzw. „Experts to experts“. Adressaten dieser Art von Portalen sind die eigenen Mitarbeiter eines Unternehmens.

Als letztes soll der Begriff des *Informationsportals* näher erläutert werden. In [Weg02] werden die Wissensportale noch weiter differenziert, wobei der Begriff des *Enterprise Information Portals* eingeführt wird. Diese beiden Begriffe werden im Folgenden synonym verwendet. Nach [Weg02] dient ein *Informationsportal* der Erschließung und Verwaltung expliziten Wissens in Form von Informationen und Inhalten aus verschiedenen Internet-Quellen oder unternehmensinternen Datei-Servern, Nachrichten, Dokumenten und Faxen. Es integriert und präsentiert die strukturierten und unstrukturierten Inhalte. Dabei sind viele der Funktionen, die dafür notwendig sind, auch in horizontalen Portalen zu finden. Funktionen, die darüber hinausgehen, sind die Kategorisierung und möglicherweise automatische Klassifikation der Inhalte. Es kommt auch hier ein personalisierter, rollenbasierter und uniformer web-basierter Zugang zum Einsatz.

Es soll noch einmal erwähnt werden, dass es sich bei einem *Informationsportal* um ein Unternehmensportal handelt (siehe [Weg02]). Allerdings wird in [MaLe02b] ein *Persönliches Informationsportal* als dualer Ansatz zum Unternehmensportal eingeführt. Es dient der persönlichen Speicherung und Verwaltung von digitalen Informationen aus verschiedenen Anwendungsbereichen. Über die gleichen Portalfunktionen wie beim Unternehmensportal kann der Privatnutzer die gespeicherten persönlichen Inhalte auch zur kooperativen Nutzung zur Verfügung stellen.

2.2 Die Plattform infoAsset Broker

Die im Rahmen der Studienarbeit entwickelte Lösung basiert auf der Verwendung des *infoAsset Brokers* als Portal-Plattform. Dazu soll hier kurz die Architektur, die eingesetzten Konzepte, sowie das Objektmodell dieser Plattform vorgestellt werden. Zum Schluss wird noch kurz die Verbindung zum WIPS IT1.1 Informationsportal aufgezeigt.

2.2.1 Architektur und Konzepte

Der infoAsset Broker bildet nach [Weg00] eine Standardsoftware für die Realisierung von Informationsportalen. Die mit dieser Software zu realisierenden Portale umfassen Wissensportale mit Fokus auf Klassifizierung und Strukturierung von Inhalten zur effizienten Informationsgewinnung sowie kommerzielle Unternehmensportale, um Mitarbeitern und Kunden Inhalte durch verschiedene auch kostenpflichtige Dienste verfügbar zu machen.

Der infoAsset Broker ist in einer Mehrschichten-Architektur realisiert. Die unterste Ebene bildet die Speicherungsschicht. Diese Schicht speichert die Inhalte und gewährleistet für die darüber liegenden Schichten einen transparenten Zugriff auf die Inhalte. Von der Art des Datenspeichers wird dabei abstrahiert. Dies ist im Regelfall ein Datenbank-System, kann aber auch eine einfache Datei sein.

In der Schicht der Anwendungslogik werden die Dienste realisiert. Hier befinden sich auch die Schnittstellen (*Interfaces*) zur Nutzung der Dienste durch Systemkomponenten. Der Zugriff auf die gespeicherten Inhalte der Speicherungsschicht erfolgt rollenbasiert für verschiedene Benutzer und Benutzergruppen über diese Dienste.

Die obersten Schichten bildet die Präsentationsschicht. Ihre Aufgabe ist es den Kunden bzw. Nutzern des Systems eine geeignete Web-Oberfläche zur Nutzung der Dienste zur Verfügung zu stellen, sowie die Benutzerinteraktionen auf die Dienste der darunter liegenden Schicht der Anwendungslogik abzubilden.

Die folgende Abbildung 2.3 zeigt die Schichten-Architektur des infoAsset-Brokers.

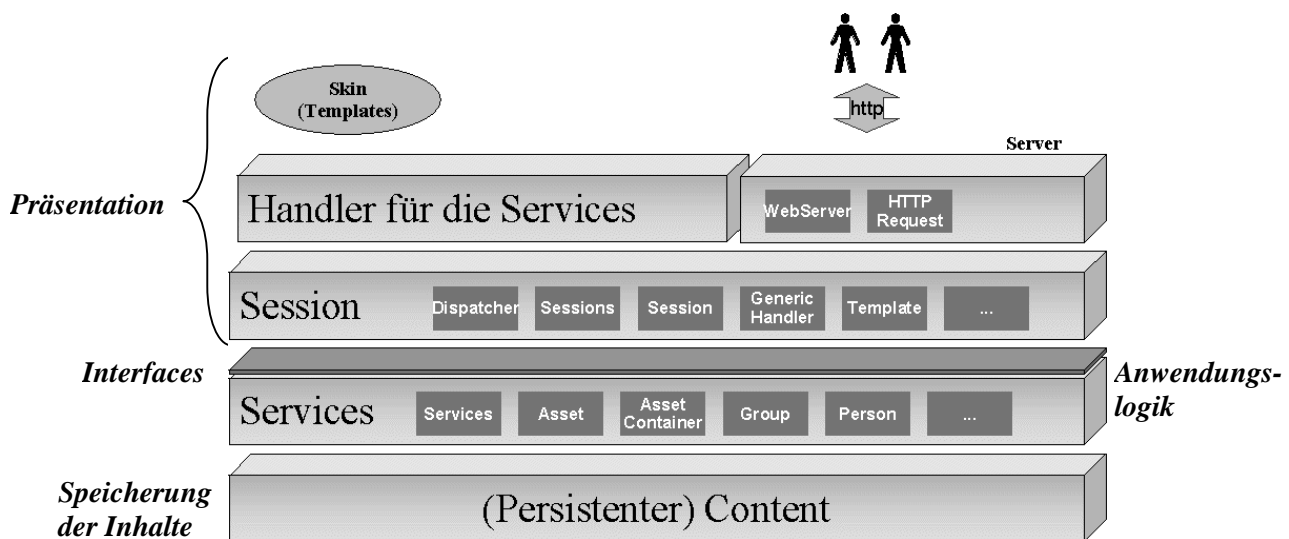


Abbildung 2.3: Architektur-Übersicht der Plattform infoAsset Brokers nach [Weg00]

Nach der Architektur sollen nun kurz die wichtigsten eingesetzten Konzepte beschrieben werden.

Assets

Assets sind Objekte der Anwendungslogik. Sie repräsentieren Informationsobjekte verschiedener Entitäten. (z.B. Dokumente, Personen, Aufträge) Assets implementieren das Interface *Asset* und können persistent gespeichert werden. Alle Assets eines Typs werden von einem sog. *AssetContainer* verwaltet.

Handler

Handler bearbeiten Anfragen an den infoAsset Broker in der Präsentationsschicht. Es gibt zwei Arten von Handlern: sichtbar und unsichtbar. Sichtbare Handler können den Systemzustand ändern und erzeugen eine Ausgabe durch Template-Substitution. Unsichtbare Handler können nur den Systemzustand ändern und die Anfrage weiterleiten. Allen Handlern aber gemein ist, dass sie von einer gemeinsamen Oberklasse *GenericHandler* erben.

Templates

Templates sind Output-Formatvorlagen. Sie treten in Form von HTML- oder WML-Vorlagen für die Präsentation auf. Sie können Platzhalter für den dynamischen Inhalt von Seiten enthalten. Dabei sind drei Arten von Platzhaltern möglich: Platzhalter für textuelle Ersetzung, bedingte Platzhalter (Ausgabe nur in bestimmten Fällen) und Listenplatzhalter (Wiederholung von Teilen). Handler substituieren die Platzhalter mit Inhalten und können Eingabedaten als Request-Parameter von den Nutzern (aus Formularen) verwenden.

2.2.2 Objektmodell

Im infoAsset Broker befinden sich die *Assets* oder Informationsobjekte in der Schicht der Anwendungslogik. In der folgenden Abbildung 2.4 sind Asset-Klassen durch eine schwarz

gefärbte linke obere Ecke gekennzeichnet. Es gibt verschiedene Asset-Typen, die im Folgenden als *AssetKinds* bezeichnet werden (z.B. Personen, Verzeichnisse, Gruppen).

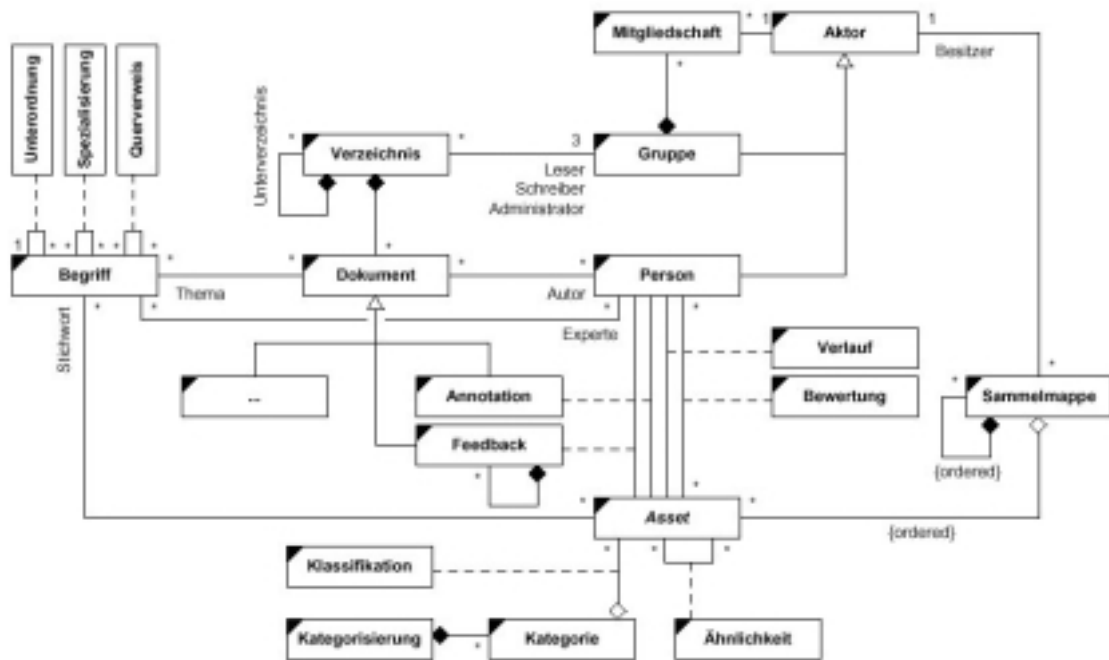


Abbildung 2.4: Konzeptuelles Modell der Broker-Services nach [Weg02]

Wie man sieht, sind fast alle der dargestellten Klassen *Assets*, stellen aber jeweils eine Spezialisierung der abstrakteren Klasse *Asset* dar. Zwischen den verschiedenen Assettypen gibt es zahlreiche assoziative und hierarchische Beziehungen. Besonders wichtig sind hier nach [Weg02] die Beziehungen zwischen Personen und allgemeinen *Assets*, da sie, und dies ist auch unmittelbar einleuchtend, eine entscheidende Voraussetzung für verschiedene Personalisierungsmöglichkeiten darstellen. Durch die Beziehungen zwischen Begriffen und anderen Assettypen wird eine Zuordnung von Schlagworten und eine allgemeine Erschließung sämtlicher gespeicherter Informationsobjekte möglich. Möglichkeiten zur Kategorisierung, Klassifikation und Suche von ähnlichen *Assets* werden durch die gleichnamigen Klassen (siehe Abbildung 2.4) genauso ermöglicht.

Assets werden durch ein Attribut *id* eindeutig innerhalb des Containers identifiziert. Der letzte Änderungszeitpunkt eines jeden *Assets* wird ebenfalls in einem Attribut festgehalten. (*lastModification*). Das folgende generische Objektmodell (Abbildung 2.5) zeigt die Verwaltung der *Assets* in *AssetContainers*. Dabei werden in einem *AssetContainer* nur *Assets* desselben Typs gespeichert. Der *AssetContainer* ist zuständig für die Verwaltung des Lebenszykluses der *Assets*, also für das Erstellen, Löschen, sowie auch die Suche nach *Assets*.

Die *Assets* unterscheiden sich nun nach dem *AssetKind* (oder Objekttyp), den sie darstellen. Über *AssetKind* werden dem *Asset* weitere Attribute zugeordnet. Die können einerseits Wert-Attribute sein (*ValueAttribute*), die einen einzelnen elementaren Wert speichern (int, String, Date), sowie Referenzattribute (*ReferenceAttribute*), die Referenzen zu anderen *Assets* speichern. Leider werden die Referenzattribute im zugrunde liegenden Version 1.x der Plattform infoAsset Broker nicht verwendet. Stattdessen wird die *id* eines *Asset* als ValueAttribut vom Typ String gespeichert. Erst durch [Ger02] fanden diese Attribute im

infoAsset Broker in Version 2.0 Anwendung. Das WIPS IT1.1 Informationsportal basiert aber noch auf Version 1.x.

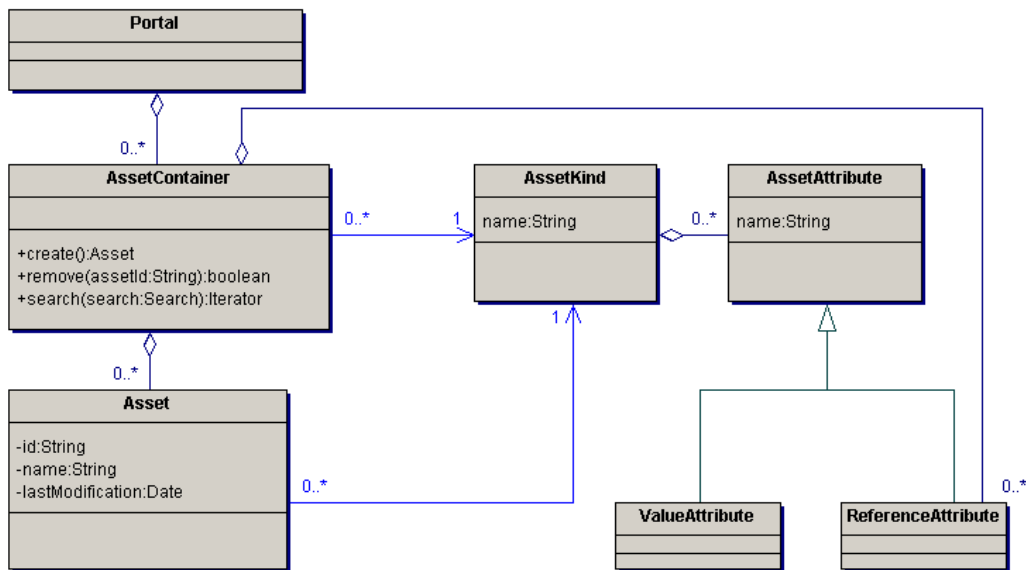


Abbildung 2.5: Generisches Objektmodell infoAsset Broker aus [Leh02]

Über den AssetContainer wird auch die Suche (*Search*) nach Assets ermöglicht. Wenn die Attribute zu suchender Assets bestimmte Kriterien erfüllen sollen, so lassen diese sich über die *SearchCriteria*s angeben. Die Qualität eines Suchergebnisses wird über vordefinierte Merkmale beurteilt (*SearchRanking*) und entsprechend sortiert. Das Suchergebnis wird durch die Klasse *SearchResult* repräsentiert. Eine Übersicht über das Konzept der Searches liefert die folgende Abbildung 2.6

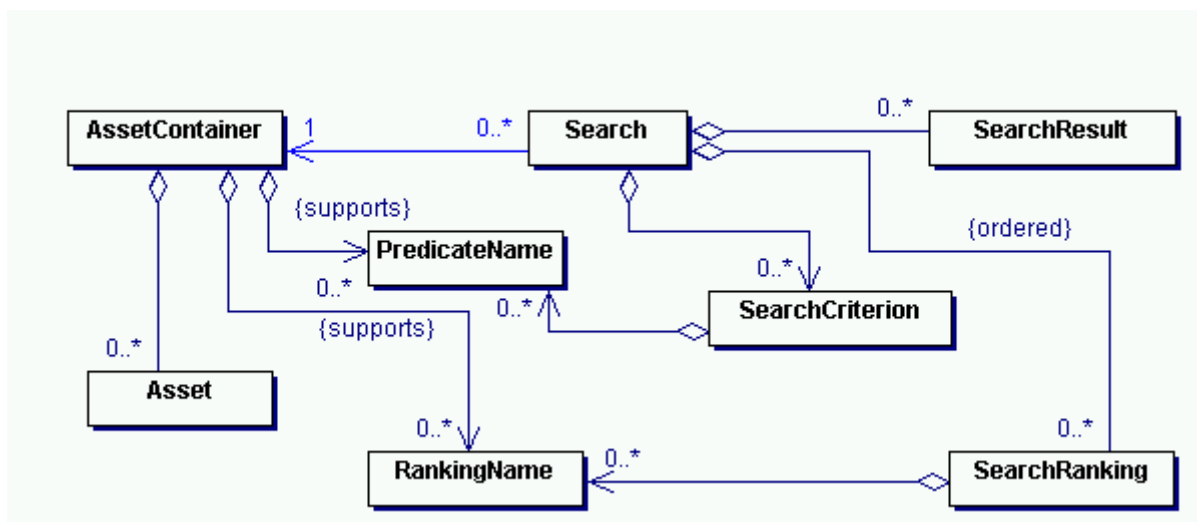


Abbildung 2.6: Konzeptuelles Klassendiagramm – Searches im infoAsset Broker

2.2.3 Das WIPS IT1.1 Informationsportal

Das WIPS IT1.1 Informationsportal entstand im Rahmen des WIPS IT1.1 Projektes (siehe auch [wips]), das vom Arbeitsbereich STS der Technischen Universität Hamburg-Harburg in Zusammenarbeit mit dem Germanischen Lloyd in Hamburg durchgeführt wurde. Dabei steht WIPS für „Wettbewerbsvorteile durch informationstechnische Produktsimulation im Schiffbau“. Als Portalplattform dient dabei der infoAsset Broker. Er wurde um Dienste erweitert, wie zum Beispiel Besichtigungsbestellung durch Reeder, Verwaltung von Besichtigungsaufträgen, Abfragen von Schiffsinformationen. Im Rahmen dieser Studienarbeit kommt das WIPS IT1.1 Informationsportal als konkretes Portal zum Einsatz. Es soll an dieser Stelle noch einmal der Unterschied zwischen dem infoAsset Broker und dem WIPS IT1.1 Informationsportal deutlich gemacht werden: Das WIPS IT1.1 Informationsportal ist ein konkretes Portal, während der infoAsset Broker eine Portalplattform darstellt.

2.3 Das Synchronisationsmodell für den infoAsset Broker

Anstelle eines Portals wurde in [Leh02] auch die Möglichkeit besprochen, dass es zentrale Portale und persönliche bzw. lokale Portale gibt. Die Informationen zwischen diesen Portalen müssen abgeglichen werden. Wenn die Informationsbestände eines Portals auf einem anderen repliziert werden, so gibt es Informationsobjekte auf beiden Portalen, die aber inhaltlich nur ein Objekt darstellen. Ändern sich die Zustände dieser Informationsobjekte nun einseitig, so repräsentieren diese *korrespondierenden* Informationsobjekte verschiedene Versionen eines Inhalts-Objekts. In diesem Falle ist ein Informationsaustausch zwischen den Portalen notwendig, um die Unterschiede dieser Informationsobjekte auszugleichen. Dieser Ausgleich der Unterschiede wird als *Synchronisation* bezeichnet. In [Leh02] ist ein Synchronisationsmodell zur Synchronisation zwischen zwei infoAsset Broker-Portalen entwickelt worden. Es wurde als weiterer Dienst der Broker-Architektur hinzugefügt und soll im Folgenden kurz erläutert werden, da dieses Synchronisationsmodell als Grundlage in der vorliegenden Studienarbeit verwendet wurde. Dazu werden zunächst die wichtigsten Klassen des Modells angesprochen. Anschließend wird die Funktionsweise dieses Synchronisationsmodells erläutert.

2.3.1 Klassen des Synchronisationsmodells

Das Modell unterscheidet die Portale danach, welches Portal den Synchronisationsvorgang initiiert. Das initiiierende Portal wird als lokales Portal, Client-Portal, Synchronisations-Client oder Client-System bezeichnet, während das andere Portal als Remote-Portal oder zentrales Portal bezeichnet wird. Die folgende Abbildung 2.7 zeigt das konzeptuelle Modell des Synchronisations-Clients.

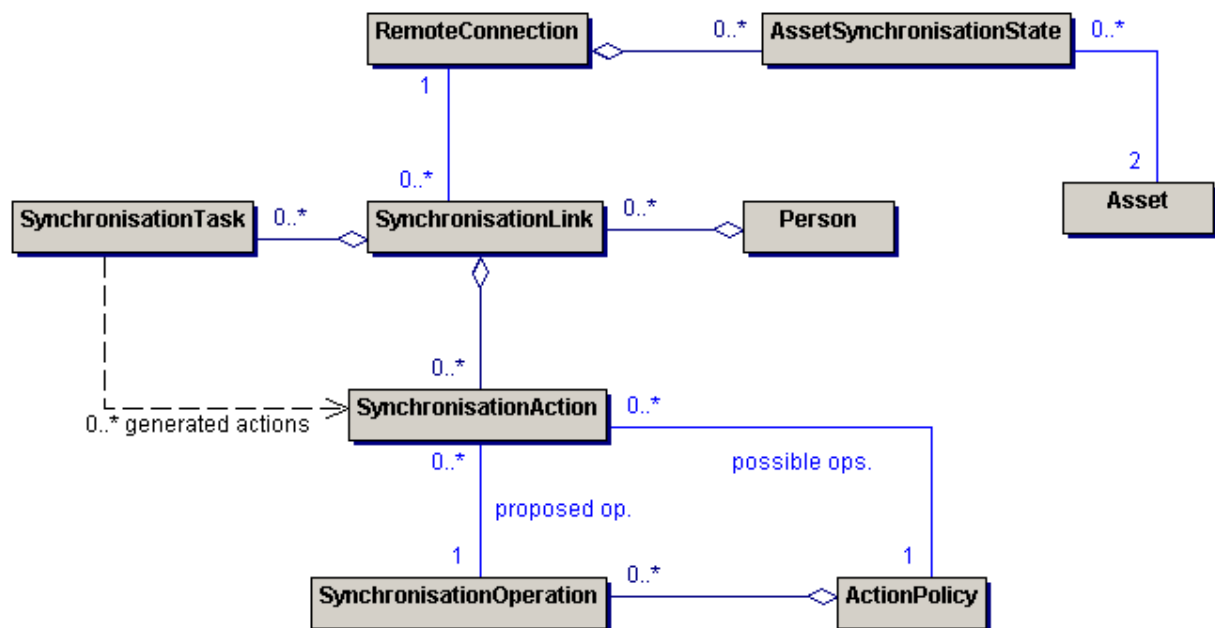


Abbildung 2.7: Konzeptuelles Synchronisationsmodell aus [Leh02]

Es soll nun eine kurze Beschreibung der abgebildeten Klassen folgen. Details dieses Modells sind in [Leh02] nachzulesen.

- *SynchronisationLinks* modellieren Synchronisationsverbindungen mit einem oder mehreren Remote-Portalen. Sie bilden den Ausgangspunkt einer jeden Synchronisation und sind einem Benutzer des Portals (*Person*) zugeordnet. Ferner enthalten sie Synchronisationseinstellungen für den Synchronisationsvorgang.
- Die Angaben zum Remote-Portal werden durch die Klasse *RemoteConnection* repräsentiert. Außerdem implementiert diese Klasse die Kommunikation mit dem Remote-Portal.
- *AssetSynchronisationState*-Objekte werden durch *RemoteConnection* erstellt und verwaltet je zwei bereits synchronisierte, korrespondierende *Assets* (Informationsobjekte).
- *SynchronisationTasks* modellieren durchzuführende Synchronisationsaufgaben für eine Menge von Informationsobjekten gleichen Typs. Dabei kann durch ein dem *SynchronisationTask* zugeordnetem *Search*-Objekt (in Abbildung 2.7 nicht gezeigt) die Menge der zu synchronisierenden Assets bestimmt werden. Ist kein *Search*-Objekt zugeordnet, werden alle Assets dieses Typs synchronisiert.
- *SynchronisationActions* modellieren eine konkret auszuführende Synchronisationsaktion. Die Aktionen ergeben sich aus den Zustandsänderungen der korrespondierenden Assets.
- Die *SynchronisationActions* beziehen sich auf genau eine *ActionPolicy*, die angibt, welche Synchronisationsoperationen (*SynchronisationOperations*) für korrespondierende Informationsobjekte in Frage kommen. Beispiele für Synchronisationsoperationen sind „lokal überschreiben“, „remote überschreiben“ und „überspringen“.

Eine weitere wichtige Klasse ist die *SynchronisationEngine*. Sie steuert auf dem Client- sowie dem Remote-Portal den Synchronisationsvorgang. Die folgende Abbildung 2.8 zeigt die Klasse und zugehörige Klassen.

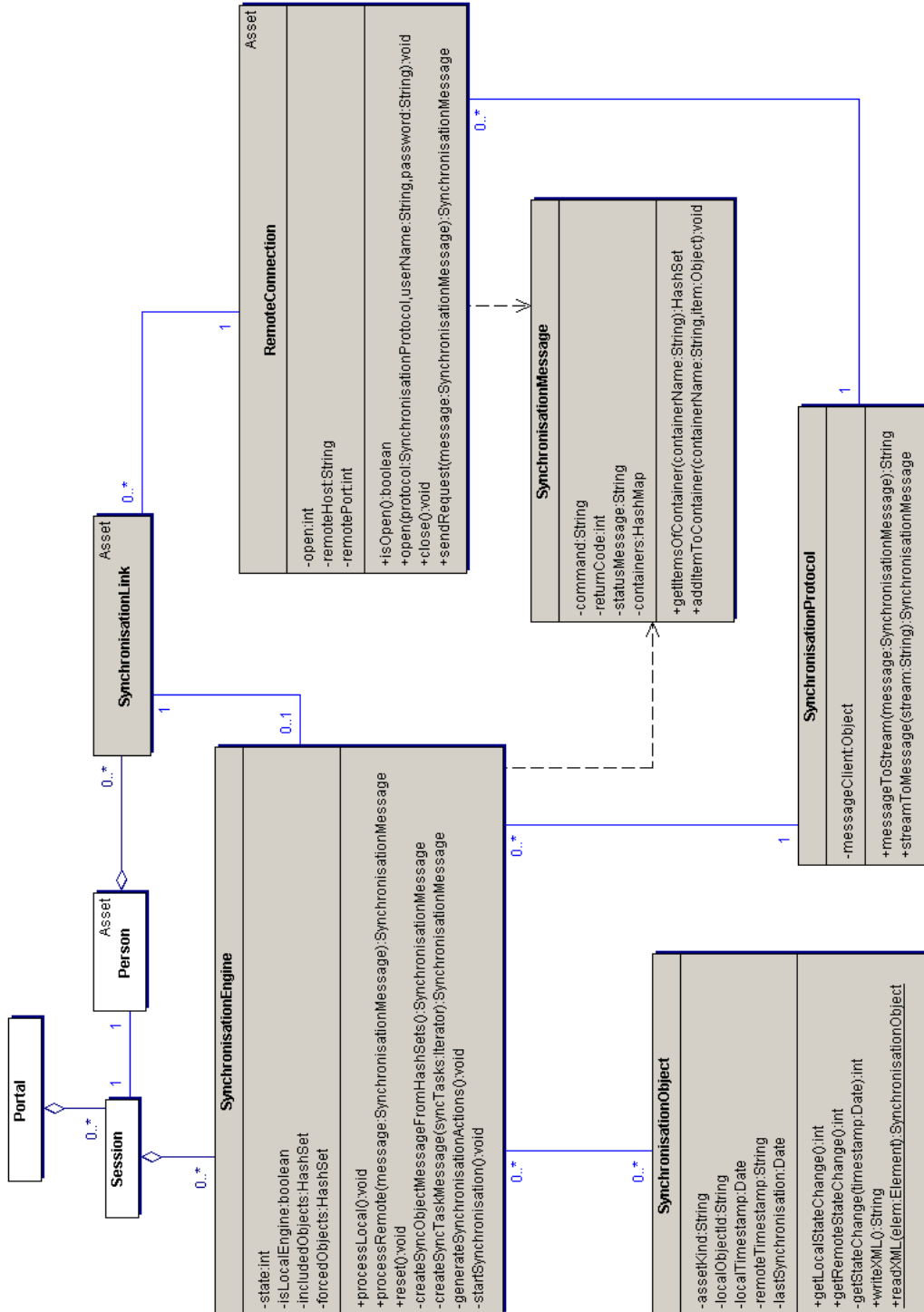


Abbildung 2.8: Konzeptuelles Klassendiagramm zum Synchronisationsvorgang aus [Leh02]

Die lokale SynchronisationEngine des Client-Systems ist zustandsbehaftet und führt die Synchronisation in Abhängigkeit ihres Zustands durch. Zum Informationsaustausch mit dem Remote-Portal kommuniziert die lokale SynchronisationEngine mit der SynchronisationEngine des RemotePortals, welche zustandslos ist. Sie nimmt Anfragen entgegen, bearbeitet sie und sendet eine Antwort zurück. Der Nachrichtenaustausch erfolgt dabei über *SynchronisationMessage*-Objekte. Die lokale SynchronisationEngine greift zur Kommunikation auf die dem jeweiligen Synchronisationslink zugeordnete RemoteConnection zurück. Um eine Verbindung herzustellen, muss der RemoteConnection das Synchronisationsprotokoll bekannt sein, welches durch die Klasse *SynchronisationProtocol* repräsentiert wird. Die Klasse legt sowohl das Transportprotokoll für den Nachrichtenaustausch sowie auch das Repräsentationsprotokoll fest, durch das *SynchronisationMessage*-Objekte zur Übermittlung zwischen den SynchronisationEngines serialisiert werden (Details : siehe [Leh02]).

Zusammenfassend ergibt sich damit die folgende Architektur, die in Abbildung 2.9 dargestellt ist. Dabei werden die Endpunkte auf dem Client-System und dem Remote-System als *SyncComm-Client* bzw. *SyncComm-Server* bezeichnet.

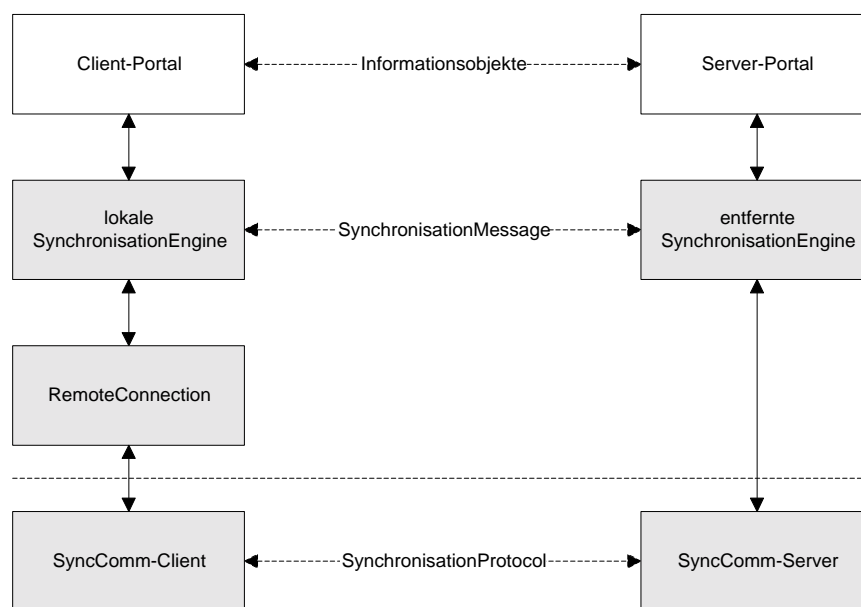


Abbildung 2.9: Informationsaustausch zwischen den Portalen aus [Leh02]

2.3.2 Der Synchronisationsvorgang

Zur Erläuterung des Synchronisationsmodells ist es zunächst notwendig, das Konzept der Mapping-Tabelle einzuführen. Jedes Client-System verwaltet zu jedem *RemoteConnection*-Objekt eine Mapping-Tabelle. Die Einträge der Mapping-Tabelle geben an, welche Informationsobjekte des lokalen Portals mit welchen des Remote-Portals (synonym: Server-Portal) *korrespondieren*, d.h. welche inhaltlich dasselbe Objekt repräsentieren. Eine Synchronisation kann nur zwischen korrespondierenden Objekten stattfinden.

Im Beispiel (vgl. Abbildung 2.10) soll vorausgesetzt werden, dass die Mapping-Tabelle Einträge bereits einmal synchronisierter Informationsobjekte enthält.

Die folgende Abbildung 2.10 zeigt das Abbilden von korrespondierenden Objekten durch eine Mapping-Tabelle beispielhaft. Dabei existiert pro Synchronisationslink („synch.“) eine Mapping-Tabelle. („l-id“ ist die lokale Objekt-Id, während „r-id“ die entfernte Objekt-Id ist)

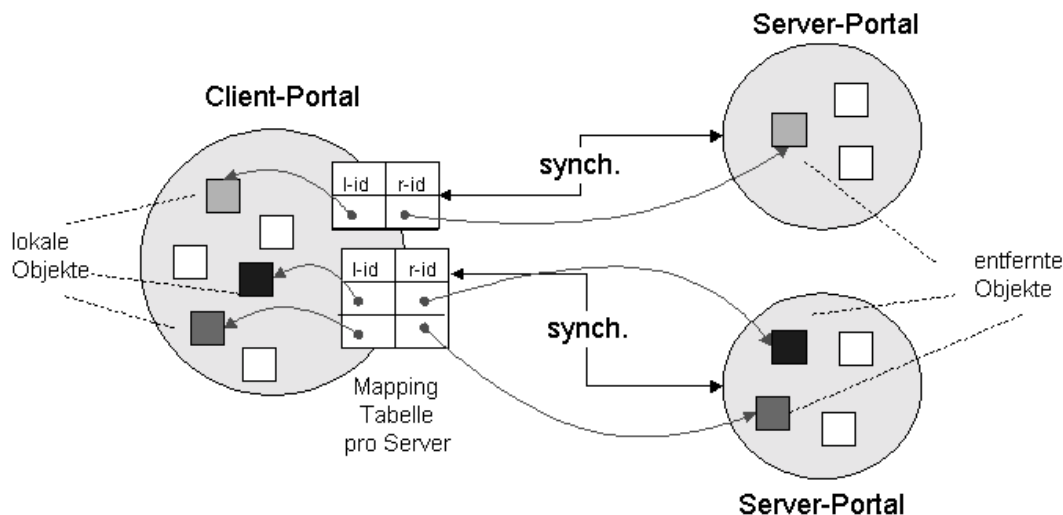


Abbildung 2.10: Abbilden korrespondierender Objekte durch die Mapping-Tabelle [Leh02]

Im Synchronisationsmodell repräsentiert ein `AssetSynchronisationState`-Objekt einen solchen Eintrag in der Mapping-Tabelle. Es besitzt Attribute, die die lokale `id` (`localObjectId`) und die entfernte `id` (`remoteObjectId`) korrespondierender Informationsobjekte speichern. Der Asset-Typ wird in `assetKind` und der letzte Synchronisationszeitpunkt dieser korrespondierenden Assets in `lastSynchronisation` gespeichert.

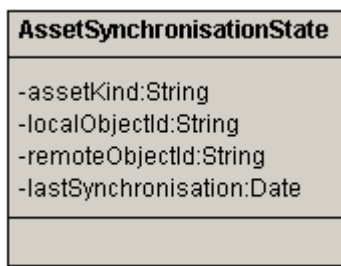


Abbildung 2.11: Attribute der Klasse `AssetSynchronisationState` [Leh02]

Das Synchronisationsmodell ist im `infoAsset Broker` so umgesetzt, dass als Synchronisationsprotokoll SOAP zu Einsatz kommt. Die `SynchronisationMessage`-Objekte werden entsprechend einem vordefiniertem XML-Schema serialisiert und dann innerhalb einer SOAP-Nachricht zwischen den `SynchronisationEngines` ausgetauscht. Die Synchronisation erfolgt dabei in zwei Phasen:

Die 1. Phase (Abbildung 2.12) beginnt damit, dass über die zum Synchronisationslink gehörenden `SynchronisationTask`-Objekte pro Asset-Typ und Richtung die Objektmengen der zu synchronisierenden Informationsobjekte bestimmt werden. Dabei gibt es die zwei

Richtungen „Publizieren“ und „Übernehmen“. Beim „Publizieren“ werden die Zustandsänderungen eines Informationsobjektes vom lokalen auf das entfernte Portal übertragen und das dort korrespondierende Informationsobjekt wird dem lokalen angeglichen. Beim „Übernehmen“ ist es genau umgekehrt. Zustandsänderungen werden vom entfernten auf das zentrale Portal übernommen. Die Zustandsänderungen korrespondierender Informationsobjekte werden von der lokalen SynchronisationEngine durch den Vergleich von Zeitstempeln (*lastModification*, siehe Kapitel 2.2) erkannt. Das lokale Portal speichert den letzten Synchronisationszeitpunkt und vergleicht die Zeitstempel der korrespondierenden Informationsobjekte mit diesem Synchronisationszeitpunkt. Liegt einer der Zeitstempel „zeitlich vor“ dem Synchronisationszeitpunkt, so hat sich das zugehörige Informationsobjekt seit der letzten Synchronisation geändert. Dazu wird von der lokalen SynchronisationEngine für jedes korrespondierende und zu synchronisierende Informationsobjektpaar in Phase 1 ein *SynchronisationObject* (siehe Abbildung 2.8) angelegt, welches die *ids*, sowie die Zeitstempel der korrespondierenden Objekte verwaltet.

Um an die Zeitstempel der entfernten Informationsobjekte zu kommen, werden dazu im Falle der Synchronisationsrichtung „Publizieren“ *SynchronisationObjects* gekapselt in einem *SynchronisationMessage*-Object zwischen den SynchronisationEngines ausgetauscht. Die entfernte SynchronisationEngine liest dazu die *ids* der *SynchronisationObjects* ein, ergänzt die Zeitstempel und sendet sie wieder zurück zum Client-Portal.

Im Falle der Synchronisationsrichtung „Übernehmen“ sendet die lokale SynchronisationEngine die betreffenden *SynchronisationTask*-Objekte wieder gekapselt durch ein *SynchronisationMessage*-Objekt zur entfernten SynchronisationEngine, welche die Menge der zu übernehmenden Assets bestimmt. Sie antwortet mit einem *SynchronisationMessage*-Objekt, welches zu jedem zu übernehmenden Informationsobjekt ein *SynchronisationObject* mit *id* und Zeitstempel des entfernten Assets enthält. Die lokale SynchronisationEngine ergänzt dann *id* und Zeitstempel der korrespondierenden lokalen Assets.

Im Anschluss werden in Phase 1 die Zustandsänderungen ermittelt und die Synchronisationsaktionen bestimmt.

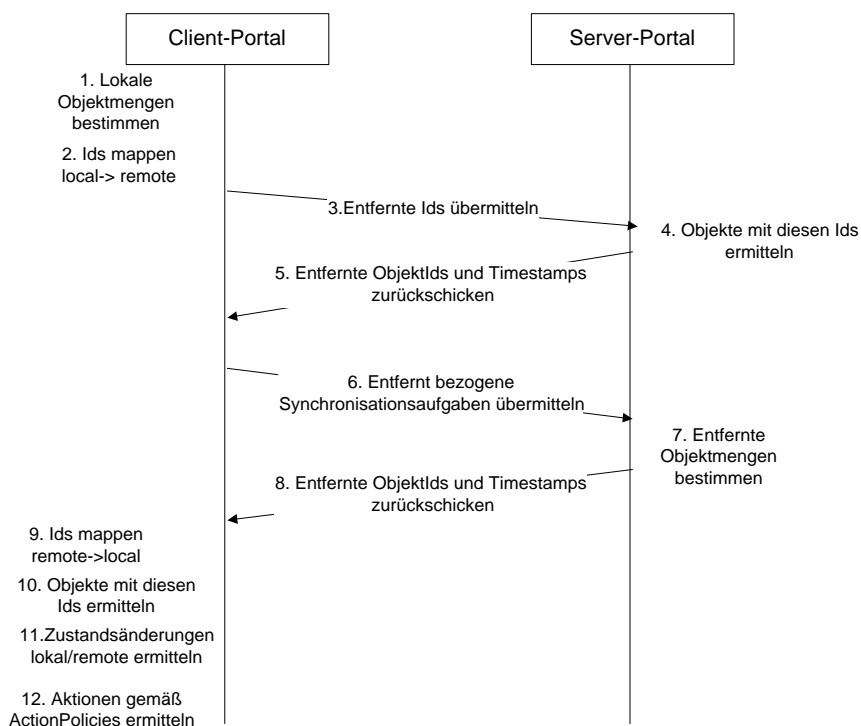


Abbildung 2.12: Synchronisationsphase I zwischen den Portalen [Leh02]

An die Synchronisationsphase I schließt sich Synchronisationsphase II an (Abbildung 2.13). In dieser Phase werden die Synchronisationsaktionen ausgeführt. Dazu werden zunächst die Synchronisationsoperationen „lokal erstellen“ ausgeführt, d.h. es werden lokal neue Informationsobjekte erstellt. Anschließend werden die Synchronisationsaktionen, wiederum gekapselt durch ein *SynchronisationMessage*-Objekt, zur entfernten SynchronisationEngine übertragen. Dabei werden die benötigten Parameter (dies sind die Informationsobjekte, die entfernt erstellt oder überschrieben werden) ebenfalls in der *SynchronisationMessage* gekapselt. Die entfernte SynchronisationEngine antwortet mit den Ergebnissen der ausgeführten Synchronisationsaktionen (gekapselt durch *SynchronisationMessage*) und ergänzt die Parameter (Informationsobjekte), die lokal überschrieben oder erstellt werden müssen. Durch die lokale SynchronisationEngine werden dann die verbleibenden lokalen Operationen ausgeführt und die Mapping-Tabelle aktualisiert.

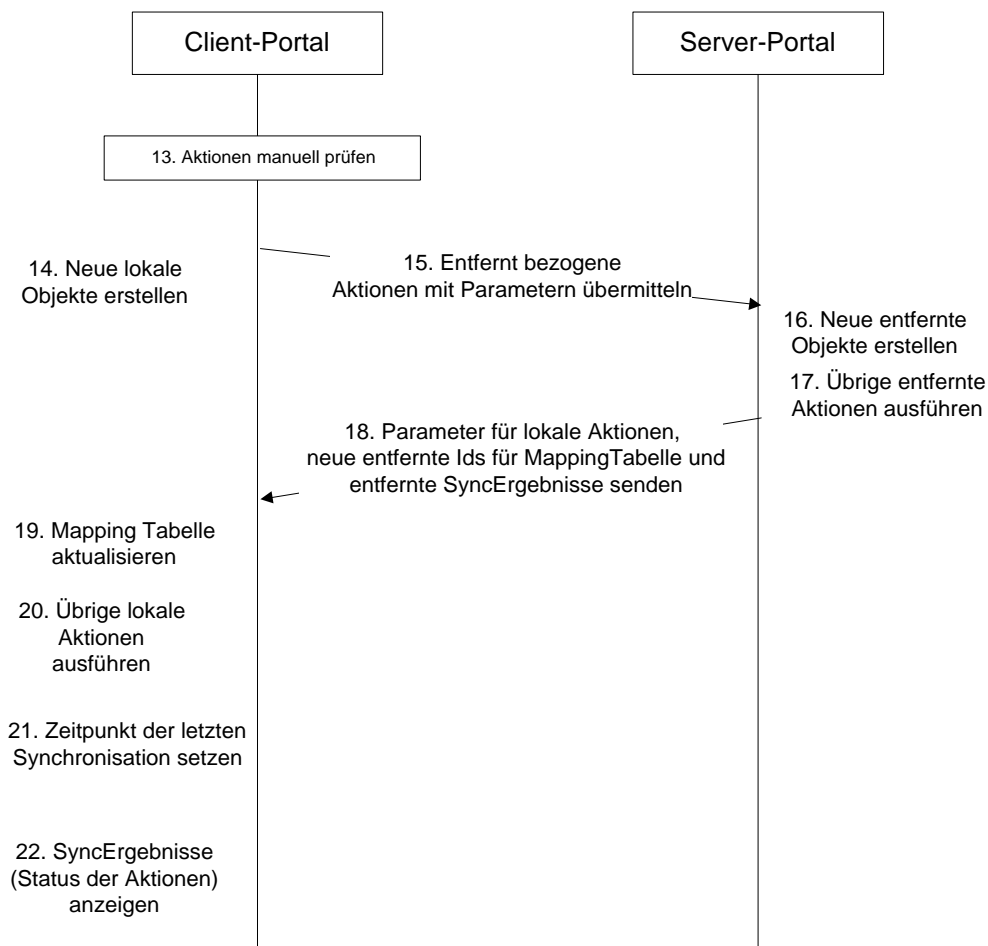


Abbildung 2.13: Synchronisationsphase II zwischen den Portalen [Leh02]

Durch das hier vorgestellte Synchronisationsmodell ist es möglich Assets zwischen Portalen, die auf der Plattform infoAsset Broker basieren, zu synchronisieren.

Kapitel 3

Case Study: Synchronisation von Informationen für die Schiffsbesichtigung

In diesem Kapitel werden die Anforderungen und Anwendungsfälle zusammengetragen, die bei der prototypischen Entwicklung einer Lösung zur Synchronisation von Besichtigungsinformationen zu berücksichtigen sind.

Die globale Tätigkeit eines Schiffsklassifizierungs-Unternehmens, wie es z.B. der Germanische Lloyd (GL, siehe auch [gl]) ist, erfordert es, dass Schiffsbesichtiger des Unternehmens an verschiedenen Standorten weltweit ihre Arbeit verrichten. Sie besichtigen im Auftrag des Eigners ein Schiff und halten ihre Beobachtungen fest. Anhand festgelegter Regeln wird auf der Grundlage der Besichtigungsergebnisse dem Schiff eine bestimmte Klasse durch den Germanischen Lloyd zertifiziert. Ohne Klassenzuteilung kann ein Schiff kaum eingesetzt werden, da z.B. Häfen bestimmte Klassen voraussetzen, damit ein Schiff sie anlaufen darf.

3.1 Ausgangssituation

Wie in vielen Geschäftsfeldern ist auch im Schiffsbesichtigungsprozess die elektronische Datenverarbeitung auf dem Vorweg. In der Zentrale des Germanischen Lloyds in Hamburg werden die Besichtigungsaufträge für die einzelnen Schiffsbesichtiger weltweit vergeben. Schiffsbesichtiger arbeiten an Laptops und können diese mit auf das zu besichtigende Schiff nehmen und vor Ort Informationen abfragen oder die Ergebnisse einer Besichtigung festhalten. Während dieser Arbeitsphase ist es oft nicht möglich, eine dauerhafte Internetverbindung aufrechtzuerhalten, da die Infrastruktur vielen Orts nicht gegeben ist oder die Kosten zu hoch sind. Auf den Servern des Germanischen Lloyds in Hamburg befinden sich alle notwendigen Informationen, die auch der Schiffsbesichtiger vor Ort benötigt, z.B. Angaben über Schiffsausrüstung oder Angaben über die Art der Besichtigung. Um die Informationen vor Ort und in der Zentrale stets auf dem gleichen Stand zu halten, müssen sie synchronisiert werden.

Als Ausgangssituation für diese Studienarbeit wird vorausgesetzt, dass sowohl in der Zentrale des GL ein WIPS IT1.1 Informationsportal im Einsatz ist (siehe Kapitel 2.2) als auch

am jeweiligen Standort des Schiffsbesichtigers. Die nachfolgende Abbildung 3.1 verdeutlicht noch einmal die Ausgangssituation:

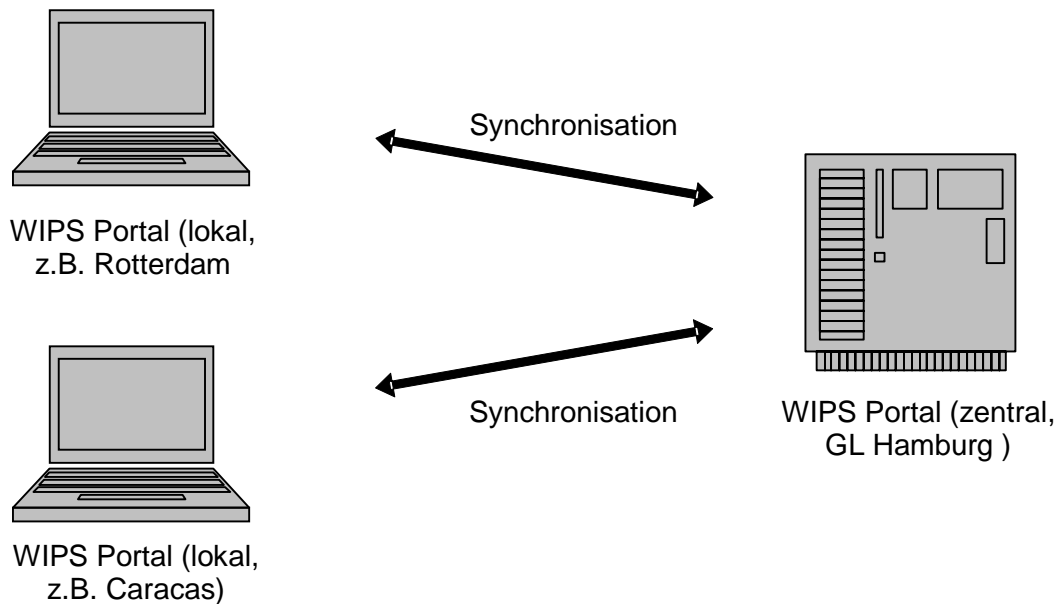


Abbildung 3.1: Ausgangssituation

3.2 Anforderungen der Synchronisation

Die folgenden Anforderungen wurden in Zusammenarbeit mit dem GL festgelegt. Sie orientieren sich an den Bedürfnissen und der Arbeitsweise eines Schiffsbesichtigers beim GL. Die festgestellten Anforderungen bilden die Grundlage für die in den nächsten Kapiteln beschriebene Lösung des Synchronisationsproblems.

Folgende Anforderungen konnten festgestellt werden:

- *Synchronisation weitgehend ohne Benutzerinteraktion:* Einmal gestartet, soll der Synchronisationsprozess ohne weitere Benutzereingaben ablaufen.
- *Offline-Arbeit muss möglich sein:* Die Internetverbindung soll nur zur Synchronisation notwendig sein, während die Arbeit die synchronisierten Informationen jederzeit ohne Internetverbindung abrufbar sein sollen.
- *Automatische/manuelle Synchronisation:* Automatische Synchronisation nach Zeitplan oder Ereignis, manuelle Synchronisation durch Benutzerwunsch.
- *Synchronisation möglichst konfliktfrei:* Diese Anforderung ergibt sich aus dem ersten Punkt. Auftretende Konflikte erfordern häufig eine Benutzerinteraktion. Eine konfliktfreie Synchronisation würde dies nicht erfordern. Es ist aber noch nicht geklärt, ob sich die Synchronisation auch konfliktfrei durchführen lässt. Dies wird im nächsten Kapitel behandelt.

- *Änderung der Schiffsdaten muss möglich sein:* Stellt der Besichtigter während einer Besichtigung fest, dass sich die Daten über ein Schiff geändert haben, so soll ihm eine Änderung dieser Daten möglich sein. Die Änderung muss bei der nächsten Synchronisation auch an die Zentrale des GL gesendet werden.

Im Rahmen der Anforderungsanalyse wurde auch untersucht, welche Informationen der Besichtigter für seine Arbeit benötigt. Diese Informationsmenge bildet auch die Basis, die bei einer Synchronisation mit dem zentralen Portal des GL in Hamburg abgeglichen werden müsste. Aus Gesprächen mit Mitarbeitern des GL wurden folgende Informationen als notwendig für die Arbeit eines Schiffsbesichtigers festgelegt:

- *Vorschriftenkatalog:* Der Vorschriftenkatalog enthält Angaben über die verschiedenen Arten der Besichtigung und genaue Vorgehensweisen bei der Besichtigung. Außerdem gibt er Aufschluss über die Kriterien, die einzelne Schiffsausrüstungsteile erfüllen müssen, um eine Zertifizierung erhalten zu können.
- *Informationen zu einem Besichtigungsauftrag:* Zeitpunkt und Ort der Besichtigung, Liegezeiten des Schiffes im Hafen, Auftraggeber, Eigner, Ansprechpartner vor Ort und zu besichtigende Schiffsausrüstung.
- *Daten über das zu besichtigende Schiff (Schiffsdaten):* Von Bedeutung ist hier die gesamte Schiffsausrüstung von der Maschine über Rettungsboote bis hin zu Navigationsinstrumenten. Aufgelistet werden hier Art, Anzahl und Hersteller der einzelnen Ausrüstungsteile.

Die beiden folgenden Punkte wurden zwar nicht als notwendig, aber als wünschenswert für die Arbeit eines Schiffsbesichtigers eingestuft. Sie können zur Not auch online in Vorbereitung einer Besichtigung abgefragt werden.

- *Bisherige Besichtigungsergebnisse des zu besichtigenden Schiffes:* Alle durch den Schiffs-Klassifizierer, hier der GL, jemals durchgeführten Besichtigungen des Schiffes, sowie deren Ergebnisse (Historie der Besichtigungen).
- *Besichtigungsergebnisse eines baugleichen Schiffes:* Relevant, da durch den Vergleich Schwachstellen oder Ausrüstungsteile mit besonderem Augenmerk erkannt werden können.

Alle oben angegebenen Informationen sind im WIPS Portal der GL-Zentrale in Hamburg elektronisch gespeichert. Bei einer Synchronisation müssten diese Informationen auf das lokale Portal des Schiffsbesichtigers überspielt bzw. Änderungen entsprechend der Synchronisationsrichtung übernommen werden.

3.3 Anwendungsfälle

Aus den oben geschilderten Anforderungen und den Gesprächen mit Mitarbeitern des GL wurden dann die folgenden Anwendungsfälle (siehe Abbildungen 3.2, 3.3, 3.4) entwickelt, die eine prototypische Lösung des Synchronisationsproblems im Rahmen der Studienarbeit abdecken sollen; das bedeutet, dass der Benutzer des zu entwickelnden Prototyps die geschilderten Anwendungsfälle konkret ausführen kann. Die Anwendungsfälle werden als UML-Anwendungsfalldiagramme gezeigt und anschließend näher beschrieben.

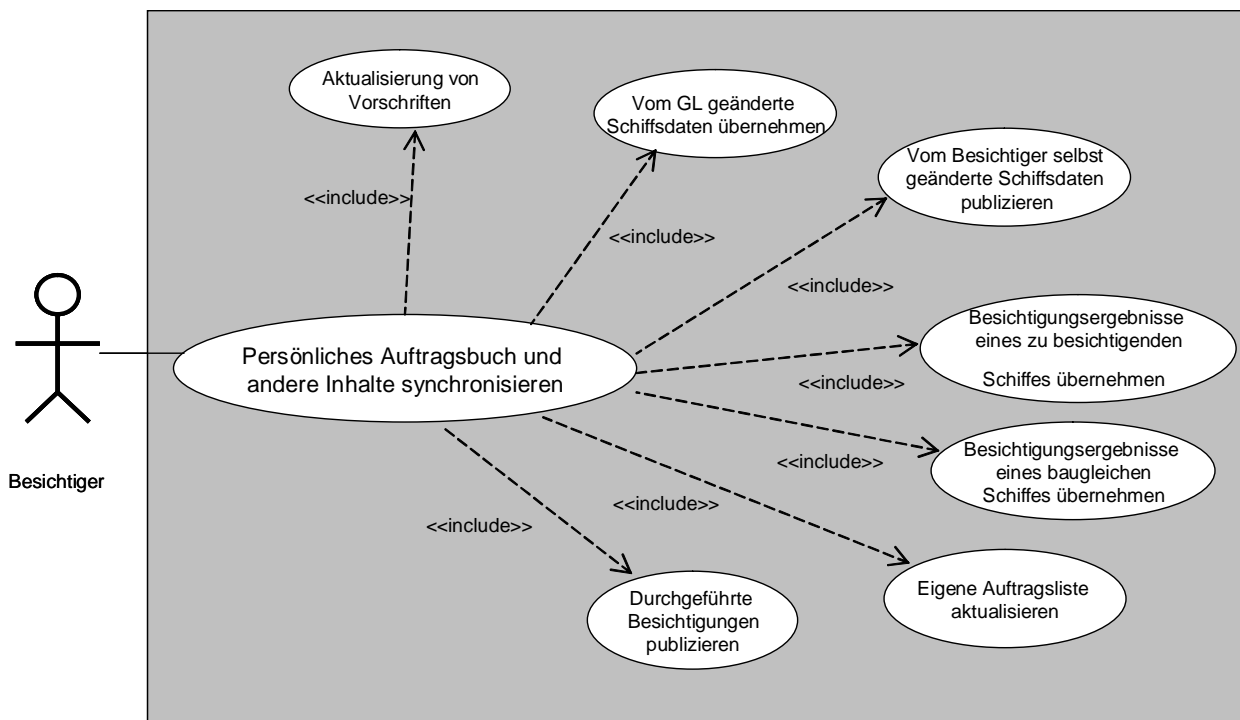


Abbildung 3.2: UML-Anwendungsfalldiagramm – Synchronisation

Persönliches Auftragsbuch und andere Inhalte synchronisieren

Dieser Anwendungsfall umfasst mehrere andere Anwendungsfälle, die unten beschrieben sind. Das persönliche Auftragsbuch enthält alle einem Schiffsbesichtigter vom GL zugeteilten Besichtigungsaufträge. Ein Besichtigungsauftrag besteht aus einer Liste von durchzuführenden Arbeiten und Informationen über das zu besichtigende Schiff (spezifische Schiffsdaten und Ergebnisse früherer Besichtigungen).

- **Aktualisierung von Vorschriften:** Der Besichtigter kann den Vorschriftenkatalog auf dem lokalen Portal aktualisieren wollen. So werden Änderungen an den Vorschriften und neue Vorschriften vom zentralen Portal übernommen.
- **Vom GL geänderte Schiffsdaten übernehmen:** Der GL hat die Schiffsdaten eines Schiffes, welches einem Auftrag des Besichtigters im persönlichen Auftragsbuch

zugeordnet ist, geändert. Der Besichtiger kann diese Änderungen nun auf das lokale Portal des Besichtigers übernehmen.

- **Vom Besichtiger selbst geänderte Schiffdaten publizieren:** Der Besichtiger hat während seiner Arbeit an einem Schiff Änderungen an den Schiffsdaten auf seinem Portal vorgenommen. Welche Daten er ändern darf, wird durch den GL vorgegeben. Durch diesen Anwendungsfall kann er nun die geänderten Daten in das Portal beim GL einspielen bzw. publizieren.
- **Besichtigungsergebnisse eines zu besichtigenden Schiffes übernehmen:** Der Besichtiger kann Ergebnisse bereits durchgeführter Besichtigungen eines Schiffes aus seinem Auftragsbuch vom GL auf sein Portal überspielen bzw. aktualisieren.
- **Besichtigungsergebnisse eines baugleichen Schiffes übernehmen:** Der Besichtiger synchronisiert die Besichtigungsergebnisse eines Schiffes, welches baugleich mit einem von ihm zu besichtigenden Schiff ist, mit seinem Portal.
- **Eigene Auftragsliste aktualisieren:** Der Besichtiger kann Änderungen an der Auftragsliste vom Portal des GL auf das eigene überspielen. Änderungen können durch neue Aufträge oder Änderungen an bestehenden Aufträgen auftreten.
- **Durchgeführte Besichtigungen publizieren:** Der Besichtiger bringt die Ergebnisse seiner Arbeit (also das konkrete Ergebnis einer Schiffsbesichtigung) in sein Portal ein. Durch die Synchronisation mit dem zentralen Portal publiziert er die Ergebnisse.

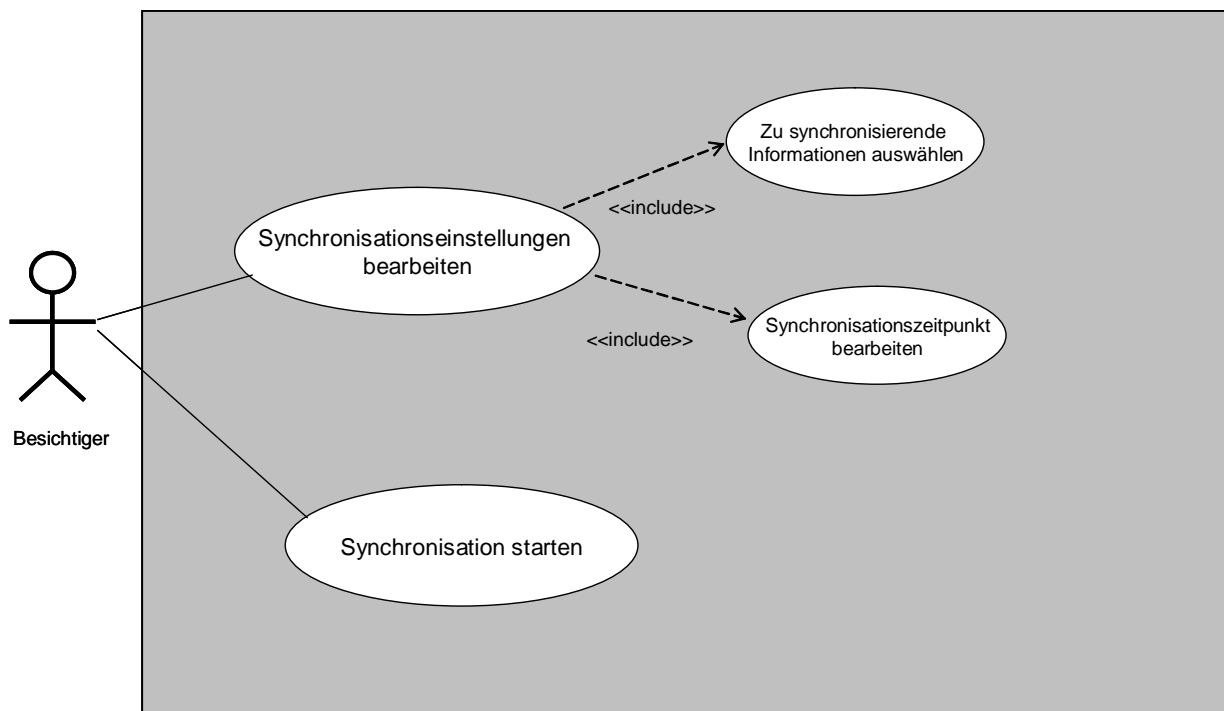


Abbildung 3.3: UML-Anwendungfalldiagramm - Synchronisationseinstellungen

Synchronisationseinstellungen bearbeiten

Der Besichtigter kann die Einstellungen für die Synchronisation vornehmen, die in den beiden enthaltenen Anwendungsfällen beschrieben werden.

- **Zu synchronisierende Informationen auswählen:** Hier kann der Besichtigter die Menge der zu synchronisierenden Informationen eingrenzen. Es können alle oben angegebenen Informationen synchronisiert werden, oder nur eine Teilmenge.
- **Synchronisationszeitpunkt bearbeiten:** Der Besichtigter kann den Zeitpunkt für die automatische Synchronisation eingeben oder ein Ereignis auswählen, bei dessen Eintritt die Synchronisation erfolgen soll.

Synchronisation starten

Erlaubt es dem Schiffsbesichtigter, manuell einen Synchronisationsvorgang zu starten.

Während der Anforderungsanalyse wurde offenbar, dass es wünschenswert ist, wenn die Änderungen an den Schiffsdaten durch einen Schiffsbesichtigter, bevor sie tatsächlich übernommen werden, noch durch einen Mitarbeiter beim GL kontrolliert werden und entweder akzeptiert oder zurückgewiesen werden können. Daher wurden auch die folgenden Anwendungsfälle mit in die Anforderungen aufgenommen.

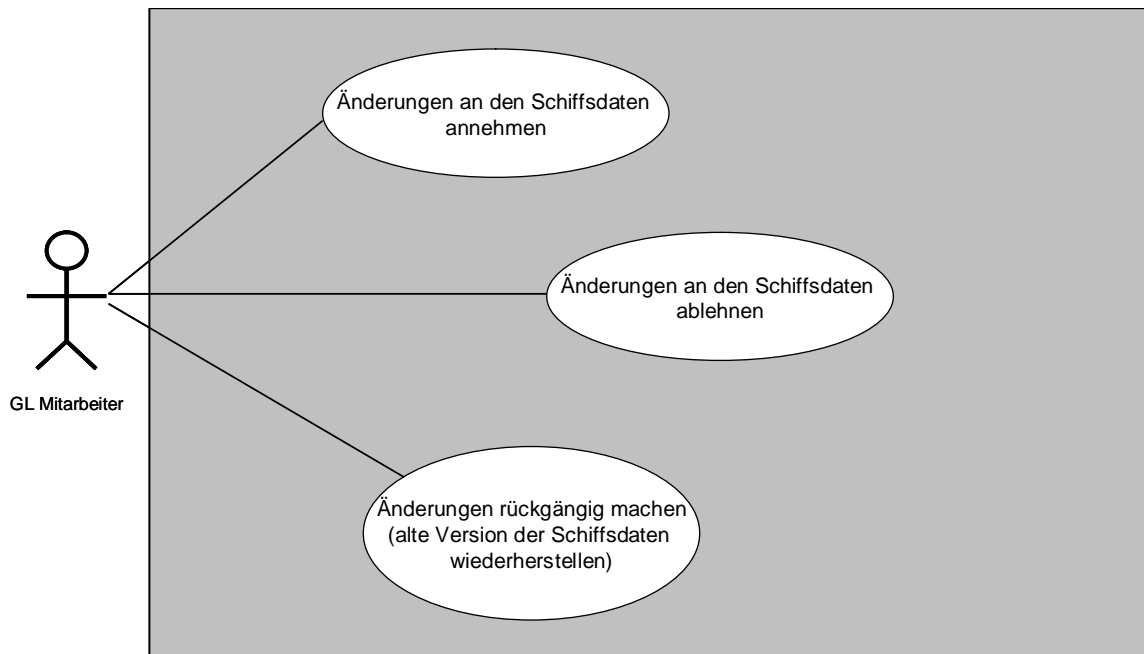


Abbildung 3.4: UML-Anwendungsfalldiagramm – Änderungen an Schiffsdaten bearbeiten

Änderungen an den Schiffsdaten annehmen

Die durch den Schiffsbesichtiger eingereichten Änderungen an Schiffsdaten kann der GL Mitarbeiter akzeptieren. Die Änderungen werden dann auf dem zentralen Portal übernommen.

Änderungen an den Schiffsdaten ablehnen

Die durch den Schiffsbesichtiger eingereichten Änderungen an Schiffsdaten kann der GL Mitarbeiter ablehnen, d.h. die Daten eines Schiffes ändern sich nicht.

Änderungen rückgängig machen

Der GL Mitarbeiter kann bereits von ihm akzeptierte Änderungen an den Schiffsdaten wieder rückgängig machen. Dadurch wird die alte, ursprüngliche Version der Schiffsdaten vor der Änderung wieder hergestellt.

Die oben geschilderten Anwendungsfälle bilden die Grundlage für die prototypische Entwicklung einer Lösung.

Kapitel 4

Entwurf und Erweiterung des Synchronisationsmodells

In diesem Kapitel soll der entwickelte Entwurf einer Lösung des Synchronisationsproblems im Schiffsbesichtigungsprozess vorgestellt werden. Als zentrale Entwurfsentscheidung soll schon an dieser Stelle erwähnt werden, dass die Grundlage einer Lösung die Verwendung des Synchronisationsmodells für die Portalplattform infoAsset Broker aus Kapitel 2.3 bilden soll. Dieses Synchronisationsmodell ist für die Synchronisation von zwei Portalen, die auf der Plattform infoAsset Broker basieren, entworfen worden und kann somit auch im vorliegenden Anwendungsfall verwendet werden, denn die WIPS Portale, die, wie im vorigen Kapitel erwähnt, auf beiden Seiten (lokal und zentral) im Einsatz sind, basieren auf der Plattform infoAsset Broker.

Aus den Anforderungen und den Entwurfsentscheidungen ergibt sich in diesem Kapitel auch, dass das bestehende Synchronisationsmodell nicht ausreicht, um eine funktionsfähige Lösung zu entwickeln. Es wird weiter erläutert, in welchen Teilen dieses Modell erweitert werden muss, um die Anforderungen erfüllen zu können.

Zunächst aber sollen verschiedene Entwurfsalternativen aufgezeigt und erläutert werden, die im Rahmen des Entwurfs entwickelt worden sind, um eine Konfliktfreiheit während des Synchronisationsprozesses zu ermöglichen. Am Ende wird ein konkreter Entwurf für dieses Teilproblem präsentiert.

Im Anschluss daran werden weitere Entwurfsentscheidungen behandelt, die sich aus den Anforderungen ergeben.

4.1 Konfliktfreiheit bei der Synchronisation

Im vorliegenden Szenario versteht man unter einem Synchronisationskonflikt die Tatsache, dass sich die zu synchronisierenden Informationen, in diesem Falle korrespondierende Asset-Objekte der Portale, seit dem letzten Synchronisationszeitpunkt auf beiden Synchronisationsseiten geändert haben. Die beiden Synchronisationsseiten sind zum einen das lokale Portal des Schiffsbesichtigers und das zentrale Portal in der Zentrale des Germanischen Lloyd. Wie in Kapitel 2 erwähnt, werden Änderungen der Asset-Objekte durch den Vergleich von Zeitstempeln mit dem letzten Synchronisationszeitpunkt erkannt. Haben sich nun zwei korrespondierende Asset-Objekte geändert, so muss der Benutzer, der die Synchronisation durchführt, während des Synchronisationsprozesse diesen Konflikt beheben, in dem er angibt, welche der beiden Versionen desselben Assets die aktuelle sein soll, oder indem er einen

Merge zwischen den beiden Versionen durchführt und so eine neue aktuelle Version bestimmt. Natürlich kann man sich eine Strategie überlegen, nachdem fest vorgegeben ist, welche Version die aktuelle ist. (z.B. immer die lokale Version) Dies bedeutet aber, dass der Benutzer die Informationen, die die andere Asset-Version repräsentiert, nie in Betracht ziehen wird, da sie bei der Synchronisation überschrieben werden würden. Da dies ein entscheidender Nachteil ist (die überschriebene Version kann wertvolle Informationen für den Benutzer enthalten haben), wurde im Rahmen der Studienarbeit untersucht, welche Alternativen es gibt, eine Synchronisation zwischen zwei Portalen auch im Konfliktfalle ohne Benutzerinteraktion und ohne Informationsverlust durchzuführen.

Diese Alternativen werden nun kurz vorgestellt und ihre Vor- und Nachteile erläutert. Zunächst aber wird untersucht, welche Informationen einen Konflikt während der Synchronisation verursachen könnten.

4.1.1 Untersuchung des Konfliktpotentials

Aus den Anforderungen im letzten Kapitel ergibt sich, welche Informationen zu synchronisieren sind. Hier wird nun kurz beschrieben, wer Änderungen an diesen Informationen herbeiführen darf und ob dies zu einem Konflikt während der Synchronisation führen kann. Die folgende Tabelle gibt darüber Aufschluss.

Informationen	Änderungen durch	Konflikt möglich
Vorschriftenkatalog	Hauptverwaltung (zentral)	Nein
Besichtigungsauftrag	Schiffsbesichtiger (lokal) Hauptverwaltung (zentral)	Nein (prinzipiell Ja)
Schiffsdaten	Schiffsbesichtiger (lokal) Hauptverwaltung (zentral)	Ja
Andere Besichtigungsergebnisse	Hauptverwaltung (zentral)	Nein

Tabelle 4.1: Konfliktuntersuchung

Ein Konflikt ist immer dann möglich, wenn Änderungen an den Informationen auf beiden Synchronisationsseiten (lokal und zentral) unabhängig voneinander vorgenommen werden können. Dies ist bei den Besichtigungsaufträgen und den Schiffsdaten der Fall (siehe Tabelle 4.1).

Im Falle der Besichtigungsaufträge ist allerdings durch den Geschäftsprozess gewährleistet, dass dort doch keine Konflikte auftreten können, denn ein Besichtigungsauftrag wird, nachdem er einem Besichtiger zugewiesen wurde, nicht mehr von Seiten der Hauptverwaltung geändert. Das bedeutet, dass ein Schiffsbesichtiger diesen Auftrag bearbeiten kann, ohne dass daran von Seiten der Hauptverwaltung noch Änderungen vorgenommen werden. Deshalb kommt es auch zu keinen Synchronisationskonflikten. Einzig die Änderungen der Schiffsdaten können zu Synchronisationskonflikten führen, denn diese können jederzeit zentral und bei einer aktuellen Besichtigung des betreffenden Schiffes auch lokal geändert werden. Alle anderen Informationen werden nur zentral geändert. Nachdem also nun bekannt ist, dass es nur die Schiffsdaten sind, die zu Konflikten führen können, wird im folgenden wieder verallgemeinert und von einem beliebigen Asset-Objekt gesprochen, dass einen Konflikt während der Synchronisation verursacht. Erst am Ende dieses Teilkapitels, bei der Erläuterung des konkreten Entwurfs für die Konfliktfreiheit, werden die Schiffsdaten wieder explizit behandelt. Es folgen nun die verschiedenen möglichen Entwurfsalternativen zur Entwicklung eines konfliktfreien Synchronisationsmodells.

Nachteile:

- Offline-Arbeit ist nicht möglich (Neue Objekte erfordern eine Verbindung zum ID-Server, um eine ID zugewiesen zu bekommen)
- Protokoll zum Kopieren ins Repository muss entwickelt werden
- Implizites Wissen, dass Portal-ID = Replikationsportal-ID ist, muss im Portal verankert werden
- Replikationsportal ist kein eigenständiges Portal, denn es enthält keinen ID-Server
- Wenn es mehr als ein lokales Portal gibt, könnte die letzte Änderung eines Datensatzes im Replikationsportal verloren gehen, wenn ein lokales Portal sie überschreibt, bevor das zentrale Portal und Replikationsportal synchronisiert wurden. Das bedeutet, es müsste zu jedem lokalen Portal ein Replikationsportal geben. Dies aber bedeutet wiederum, dass unverhältnismäßig viele Hardwareressourcen bereitgehalten werden müssten, um einen Betrieb der Portale zu ermöglichen.

Zusammenfassend wird klar, dass dies ein eher schlechter Lösungsansatz ist, da in dem vorliegenden Fall sehr viele lokale Portale betrieben werden müssten. Für jedes lokale Portal ein Replikationsportal zu betreiben, würde unnötige Kosten und Zeit für die Wartung verschlingen.

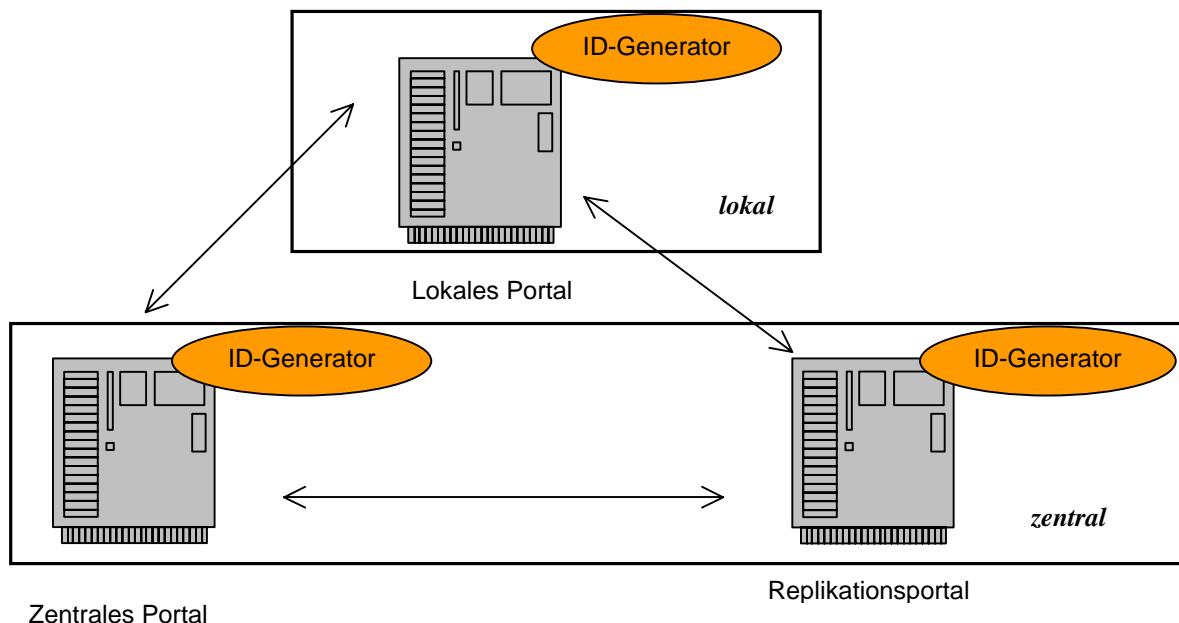
4.1.3 Alternative 2 : Drei Portale und lokale ID-Generatoren

Abbildung 4.2: Drei Portale und lokale ID-Generatoren

Die Funktionsweise gleicht der von Alternative 1. Allerdings werden hier die IDs eines Objektes nicht zentral generiert, sondern lokal von den sog. ID-Generatoren. Die generierten IDs müssen global eindeutig sein, d.h. eine ID darf niemals zweimal generiert werden. Dies muss der Algorithmus sicherstellen. Ist diese Voraussetzung erfüllt, kann auch hier das ID-Mapping verhindert werden, da die IDs ja global eindeutig sind und so auf jedes Portal übernommen werden können.

Zu den Vor- und Nachteilen von Alternative 1 kommen folgende hinzu :

Vorteile:

- Offline arbeiten möglich (ID kann lokal generiert werden)
- Replikationsportal kann als vollwertiges Portal dienen

Nachteile:

- ID-Generatoren müssen entwickelt werden
- Eindeutigkeit der generierten IDs muss gesichert werden
- Die zentralen Interfaces *Asset* und *AssetContainer* der InfoAssetBroker-Plattform sowie die zugehörigen implementierenden Klassen müssen geändert werden, um die globalen IDs verarbeiten zu können.

Auch bei dieser Alternative bleibt das Problem, dass bei mehreren lokalen Portalen und nur einem Replikationsportal möglicherweise Änderungen überschrieben werden. Das Einzige, was diese Lösung wirklich qualitativ besser aussehen lässt als Alternative 1, ist der Punkt, dass Offline-Arbeit hier möglich ist. Dem entgegen steht allerdings die Tatsache, dass die dem Portal zugrunde liegende Plattform infoAsset Broker stark verändert werden muss. Die Interfaces *Asset* und *AssetContainer* und die implementierenden Klassen sind zentrale Bestandteile der Portalplattform. Eine Änderung würde enormen Aufwand bedeuten, da alle davon erbenenden Klassen eventuell geändert werden müssen.

Dies lässt sich aber verhindern, indem jedes Portal eine Mapping-Tabelle zur Synchronisation einführt, ähnlich der im Synchronisationsmodell aus Kapitel 2.3. Allerdings werden dort nicht die IDs korrespondierender Asset-Objekte eingetragen, sondern es wird die in der Plattform infoAsset Broker verwendete Asset-ID auf eine global eindeutige ID eines Generators abgebildet. Folgende Abbildung verdeutlicht diesen Sachverhalt noch einmal.

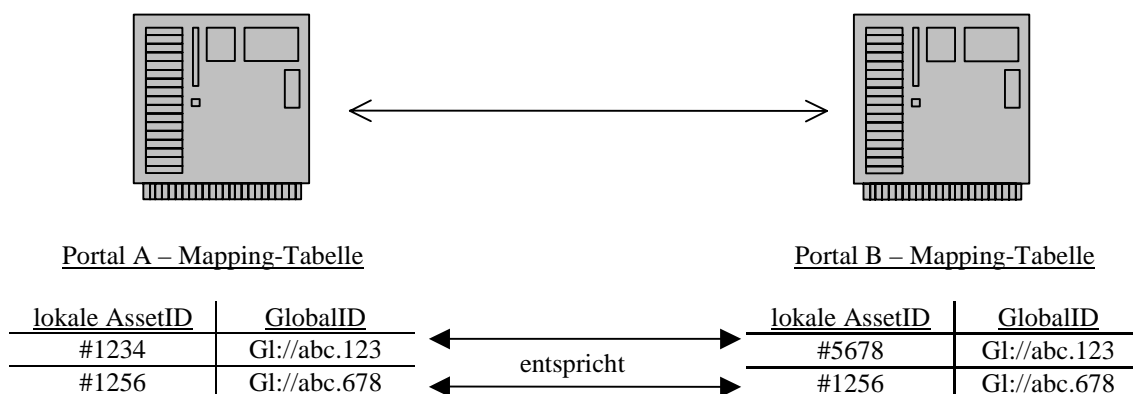


Abbildung 4.3: Mapping-Tabelle globale/lokale Asset-IDs

Es kann also durchaus sein, dass eine globale ID auf einem Portal auf eine lokale Asset-ID abgebildet wird, während die gleiche globale ID auf einem anderen Portal auf eine ganz andere lokale Asset-ID abgebildet wird. Wichtig ist bei diesem Konzept nur, dass die globalen IDs eindeutig sind und eine globale ID nur ein einziges Asset-Objekt repräsentiert, das aber auf verschiedenen Portalen verschiedenen Asset-IDs besitzen darf.

Zusammenfassend lässt sich bei dieser Lösung sagen, dass der Aufwand, der in die Entwicklung eines solchen Systems gesteckt werden müsste, nicht im Rahmen dieser

Studienarbeit zu erledigen ist. Außerdem bleibt auch immer noch dasselbe Problem wie bei Alternative 1, nämlich dass Änderungen verloren gehen könnten, wenn die Synchronisation zwischen Replikationsportal und zentralem Portal nicht geschieht, bevor neue Änderungen in das Replikationsportal übernommen werden.

4.1.4 Alternative 3 : Zwei Portale und Synchronisation durch Hauptverwaltung

Es gibt in diesem Szenario kein Replikationsportal. Die Daten verbleiben auf dem lokalen Portal und werden im Konfliktfall bei der Synchronisation zunächst übergangen. Dann muss sich die Hauptverwaltung des Germanischen Lloyd bzw. ein zentraler Konfliktlöser mit dem lokalen Portal verbinden und die den Konflikt verursachenden Asset-Objekte synchronisieren. Während der Synchronisation behebt er dann den Konflikt. Bei dieser Variante wäre die Synchronisation für den Schiffsbesichtigter ebenfalls ohne Benutzerinteraktion möglich.

Vorteile dieses Ansatzes:

- Einfache Realisierung mit dem existierenden Synchronisationsmodell

Nachteile:

- Lokale Portale arbeiten oft offline und sind nicht zur regelmäßigen Zeiten verfügbar

Dies ist ein schlechter Lösungsansatz, denn er ist in der Praxis schwer zu anzuwenden. Die lokalen Portale sind lange offline und sollen nicht nach einem bestimmten Zeitplan online gehen. Der Synchronisationsprozess zur Konfliktauflösung kann also nicht regelmäßig und automatisch ablaufen. Der zentrale Konfliktlöser kann nicht damit rechnen, dass ein lokales Portal auch wirklich online ist.

4.1.5 Alternative 4: Zwei Portale und Synchronisationsdaten per Email

Im Konfliktfall werden die Daten des den Konflikt verursachenden Asset-Objektes per Email an den zentralen Konfliktlöser beim GL gesendet und dieser entscheidet dann, was damit passieren soll. Er kann sie dann ganz oder nur zum Teil übernehmen oder verwerfen. Auf dem lokalen Portal werden dann bei der nächsten Synchronisation die angepassten Daten vom zentralen Portal übernommen.

Vorteile:

- Einfache Realisierung mit dem existierenden Synchronisationsmodell

Nachteil:

- Skaliert nicht, denn die Attribute eines Objektes könnten auch Bilder, Diagramme und Videos sein. In diesem Fall wäre ein Versand per Email nicht angebracht, wenn es sich um große Datenmengen handelt. Dazu kommt, dass im Falle vieler lokaler Portale, die anfallende Email-Flut nicht zu handhaben ist.
- Unnötige „Handarbeit“, die durch Portal vermieden werden sollte

Dass die Lösung nicht skaliert ist ein entscheidender Nachteil. Bei großen Datenmengen oder einer hohen Anzahl an Konflikten eignet sich die Lösung nicht.

4.1.6 Alternative 5 : Zwei Portale und verschiedene Asset-Versionen

Das WIPS Portal wird um eine Versionsstruktur ergänzt. Es existieren Versionen als unabhängige Asset-Objekte mit verschiedenen IDs. Die Versionsstruktur erklärt den Zusammenhang zwischen Version und Objekt. Im Konfliktfall speichert das lokale Portal ein neues Objekt als neue Version eines bereits vorhandenen Objektes im zentralen Portal und der Konfliktlöser löst den Konflikt zwischen diesen Versionen.

Vorteile dieses Ansatzes:

- Zwei Sichten zur Laufzeit denkbar : Mit und ohne Versionen (nur die aktuellste Version eines Assets ist sichtbar)

Nachteile:

- *Asset* und *AssetContainer* sowie implementierende Klassen müssen geändert werden, um die neuen Versionen verarbeiten zu können

Diese Alternative ist schon ein sehr guter Lösungsansatz, aber die Portal-Software muss stark verändert werden. Die Interfaces *Asset* und *AssetContainer* und die implementierenden Klassen sind zentrale Bestandteile der Portalplattform. Eine Änderung würde enormen Aufwand bedeuten, da alle davon erbenenden Klassen eventuell geändert werden müssen. Es bleibt zu überlegen, ob sich der Aufwand für die Synchronisation von Schiffsdaten lohnt. Wie aber gleich beschrieben wird, bildet diese Alternative die Grundlage für die Entwurfsentscheidung.

4.1.7 Entwurf einer Lösung für die Konfliktfreiheit

Wie im Unterabschnitt 4.1.1 erläutert, können nur die Schiffsdaten im vorliegenden Szenario einen Konflikt während der Synchronisation verursachen. Zur Lösung der Konfliktfreiheit wird die Alternative 5 herangezogen und leicht abgeändert. Die Alternative wird speziell auf die Anforderungen im vorliegenden Fall angepasst und implementiert. Eine Verallgemeinerung zur Behandlung von Konflikten bei beliebigen Asset-Typen kann später erfolgen.

Modellierung von *UpdateRequests* für in Konflikt stehende Objekte

Es wird ein neuer Asset-Typ eingeführt, der den Namen *ShipUpdateRequest* trägt. Er gleicht von der Struktur dem Asset-Typ, der Schiffsdaten repräsentiert, in diesem Fall *Ship*, und wird um Attribute für Meta-Informationen erweitert (siehe konzeptuelles Klassendiagramm in Abbildung 4.4, weiter unten). Ein *ShipUpdateRequest* modelliert einen Änderungsantrag an den Daten eines Schiffes. Wenn ein Schiffsbesichtiger eine Änderung der Schiffsdaten während der Besichtigung feststellt, so legt er einen Änderungsantrag an und trägt die Informationen ein, die sich geändert haben. Auf dem lokalen Portal wird dadurch ein Asset vom Typ *ShipUpdateRequest* generiert, dessen Attribute die zu ändernden Informationen beinhalten. Bei der nächsten Synchronisation gelangt dieses Asset dann auf das zentrale

Portal und wird dort vom Konfliktlöser bearbeitet. Entweder wird die Änderung angenommen (auch nur teilweise) oder abgelehnt.

Durch dieses Verfahren wird erreicht, dass sich die Schiffsdaten nur noch zentral ändern, denn der Schiffsbesichtigter führt keine Änderungen mehr an Schiffsdaten durch. Also können keine Konflikte der Schiffsdaten während der Synchronisation auftreten. Wird ein Änderungsantrag auf dem zentralen Portal angenommen, so werden die Schiffsdaten entsprechend geändert. Die geänderten Schiffsdaten gelangen dann bei der nächsten Synchronisation wieder auf das lokale Portal des Schiffsbesichtigters. Damit ist auch die Anforderung erfüllt, dass die Änderungen an Schiffsdaten zentral angenommen oder abgelehnt werden sollen. (Siehe Kapitel 3.3)

In dem folgenden Klassendiagramm (Abbildung 4.4) werden das entworfene Asset, sowie die hinzugefügten Meta-Informationen gezeigt: Ein *ShipUpdateRequest* kennt den Besichtigter (surveyor), der es angelegt hat, sowie den Mitarbeiter, der es bearbeitet hat (personInCharge) und natürlich das Schiff, dessen Daten geändert werden sollen (ship). Die Datums-Attribute geben Aufschluss darüber, wann ein *ShipUpdateRequest* angelegt, auf dem zentralen Portal empfangen und bearbeitet wurde. Außerdem gibt es noch Attribute, die den Bearbeitungsstatus des *ShipUpdateRequests* angeben (status) und ein Attribut, das Gründe im Falle einer eventuellen Ablehnung speichern kann. (reasons). Da *ShipUpdateRequest*, wie oben beschrieben, ansonsten dem Asset vom Typ *Ship* gleicht, werden diese Attribute und Referenzen zur Übersichtlichkeit weggelassen. Im Anhang B ist ein Klassendiagramm für die Klasse *Ship* angegeben.

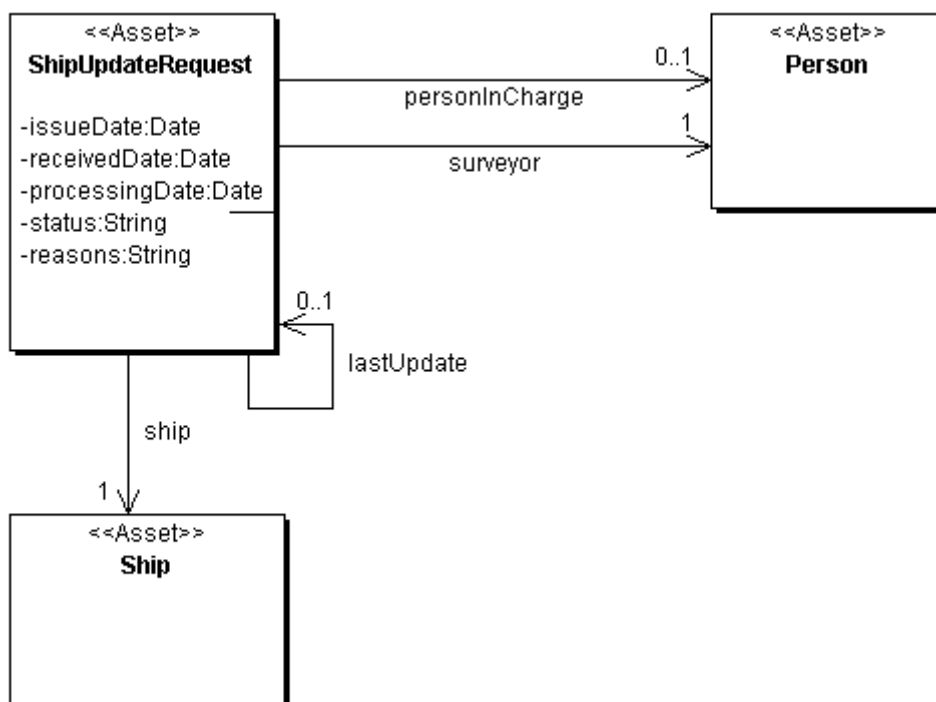


Abbildung 4.4: Konzeptuelles Klassendiagramm - ShipUpdateRequest

Zusätzlich wird eine lineare Verkettung der *ShipUpdateRequests* für ein Schiff eingeführt, wodurch alle *ShipUpdateRequests* eines Schiffes in eine zeitliche Reihenfolge gebracht werden. Ein Attribut speichert die ID des letzten *ShipUpdateRequests*, so dass am Anfang der

Kette immer das neuste *ShipUpdateRequest* für ein Schiff ist. (siehe auch folgendes Diagramm)

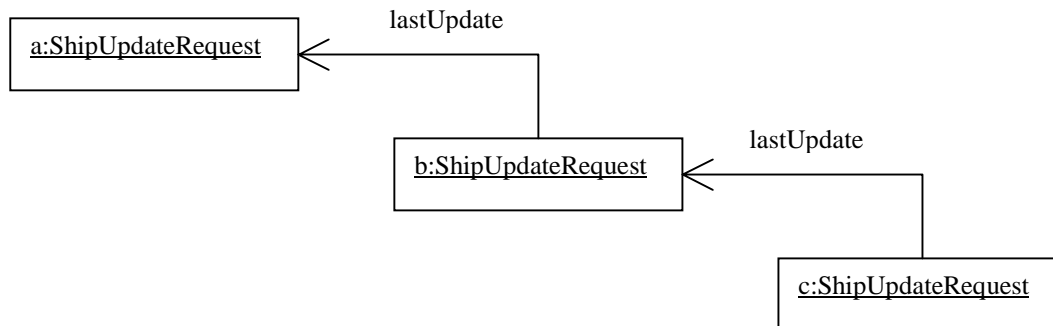


Abbildung 4.5: lineare Verkettung der *ShipUpdateRequests*

In der Abbildung 4.5 wäre das *ShipUpdateRequest* c der zuletzt bearbeitete Änderungsantrag für ein bestimmtes Schiff und damit der Anfang der Kette. Die Verkettung geschieht auf dem zentralen Portal zum Zeitpunkt der Bearbeitung der Änderungsanträge.

Diese Lösung hat der Autor gewählt, da sie die Anforderungen erfüllt und von allen Alternativen deutlich weniger Aufwand bei der Implementierung erwarten lässt. Es muss nur ein neuer Asset-Typ implementiert werden, sowie Methoden zum Anlegen, Bearbeiten und Verketteten von *ShipUpdateRequests*. Alle Assets lassen sich mit dem vorgestellten Synchronisationsmodell synchronisieren und es ist kein Replikationsportal notwendig. Eine Verkettungsstruktur speziell nur für den neuen Asset-Typen einzuführen, bedeutet, dass auch die Interfaces *Asset* und *AssetContainer* unverändert bleiben. Auch führt die Synchronisation des neuen Asset-Typen nicht zu neuen Synchronisationskonflikten, denn Änderungsanträge (repräsentiert durch Assets vom Typ *ShipUpdateRequest*) werden nur einmal lokal durch den Besichtigter generiert und nach ihrer Generierung nur noch zentral bearbeitet.

4.2 Lineare Versionierung von Schiffsdaten

Aus den Anforderungen ergibt sich, dass die Änderungen an den Schiffsdaten jederzeit wieder rückgängig zu machen sind. Da aber an keiner Stelle des WIPS Portals die alten Versionen von Schiffsdaten gespeichert werden, ist das nicht ohne weiteres möglich. Um das aber trotzdem zu ermöglichen, wird eine Versionierung nur für die Assets vom Typ *Ship* eingeführt. Dadurch kann verhindert werden, dass die zentralen Interfaces *Asset* und *AssetContainer* geändert werden müssen. Die Kontrolle der Versionsstruktur wird allein durch die implementierende Klasse und den *AssetContainer* übernommen. Es wird eine lineare Versionierung eingeführt, wobei jede Version eines *Ship*-Assets seine Vorgängerversion kennt. Eine lineare Versionierung ist auch ausreichend, da keine gleichzeitige Änderung von Schiffsdaten durch mehrere Benutzer vorausgesetzt wird. Realisiert wird dies, indem dem Typ *Ship* eine Referenz auf sich selbst hinzugefügt wird. Außerdem wird ein Flag hinzugefügt, welches angibt, ob es sich bei einem Asset vom Typ *Ship* um die aktuelle (neueste) Version handelt, oder um eine ältere Version.

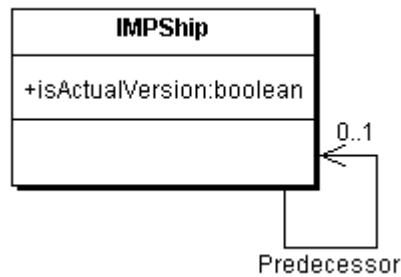


Abbildung 4.6: konzeptuelles Klassendiagramm - Lineare Versionierung von Schiffsdaten

In der späteren Anwendung wird immer dann, wenn die Daten eines Schiffes geändert werden, eine neue aktuelle Version mit den geänderten Daten erzeugt, während die vorige Version ihren Status als neueste Version verliert, aber unverändert weiter existiert. Die neue aktuelle Version zeigt dann über eine Referenz auf die vormals aktuelle Version.

Bei dieser Verkettung handelt es sich um eine tatsächliche Versionierung und nicht nur um eine Verkettung von Asset-Objekten, denn die Assets vom Typ *Ship* repräsentieren tatsächlich verschiedene Informations-Versionen ein und desselben Schiffes. Im vorigen Abschnitt wurden *ShipUpdateRequest*-Assets verkettet, die aber verschiedene Änderungsanträge repräsentierten und nicht Versionen des gleichen Antrags.

4.3 Erweiterungen des Synchronisationsmodells

Das Synchronisationsmodell für den infoAsset Broker (siehe auch [Leh02]) unterstützt in der bisherigen Version nur die Synchronisation von einzelnen Assets. Das bedeutet, wenn ein zu synchronisierendes Asset durch eine Referenz auf ein anderes verweist, so wird diese Referenz bei der Synchronisation unbrauchbar. Denn zum einen wird zwar das zu synchronisierende Asset übermittelt, aber nicht das referenzierte und sollte sich das referenzierte Asset durch Zufall auch auf beiden Portalen, zwischen denen synchronisiert wird, befinden, so ist keinesfalls gewährleistet, dass es dort dieselbe Asset-ID trägt wie auf dem anderen Portal. Die Referenz würde also ins Leere führen (Fachbegriff: dangling reference). In dem vorliegenden Szenario ist es aber von Bedeutung, dass auch referenzierte Assets bei der Synchronisation mit übermittelt werden, da sonst relevante Informationen fehlen und der Schiffsbesichtigter seiner Arbeit nicht nachgehen kann. Ein Besichtigungsauftrag zum Beispiel (*Inspection*) referenziert eine Bestellung (*Order*), die wiederum eine Besichtigungsanfrage referenziert (*Request*), welche ein Schiff referenziert (*Ship*, siehe dazu auch Anhang B, Klassendiagramm Besichtigungsauftrag). Würde jetzt nur ein Asset vom Typ (*Inspection*) bei der Synchronisation übermittelt, so fehlt dem Schiffsbesichtigter die Information darüber, um welches Schiff es sich handelt, wann die Besichtigung bestellt ist, in welchem Hafen, was besichtigt werden soll und noch mehr. Deshalb muss das Synchronisationsmodell so erweitert werden, dass es auch referenzierte Assets synchronisieren kann. Zu diesem Zweck wird im Abschnitt 4.3.1 die Definition von Objektgraphen beschrieben, die die Menge zu synchronisierender Assets beschreiben sollen. Ein weiteres Problem ergibt sich bei der Bestimmung von Zustandsänderungen durch den Vergleich von Zeitstempeln. Wenn die Systemuhren der beiden Portale unterschiedlich laufen, so werden die Änderungen möglicherweise nicht richtig erkannt. Auch in diesem Punkt muss das Modell erweitert werden, wie im Abschnitt 4.3.2 beschrieben.

4.3.1 Definieren der Objekthülle

Damit auch referenzierte Assets synchronisiert werden, wird pro Anwendungsfall eine *Objekthülle* definiert. Alle Objekte, die innerhalb dieser Hülle liegen, werden dann bei einem Synchronisationsvorgang berücksichtigt. Diese Lösung ist für jeden modellierten Anwendungsfall angepasst und umfasst nur die beim jeweiligen Anwendungsfall zu synchronisierenden Assets. Zur Bestimmung der Objekthülle wird als erstes der Asset-Typ gewählt, der in dem betreffenden Anwendungsfall synchronisiert werden soll. Im folgenden wird dieser als Wurzel-Knoten bezeichnet. Dann wird ein Referenzbaum aufgestellt, der so viele Äste von der Wurzel ausgehend besitzt, wie der Wurzel-Knoten Referenzen auf andere Assets, die synchronisiert werden sollen, besitzt. Die Blätter dieser Äste sind dann die referenzierten Assets. Von diesen wiederum ausgehend werden auch alle Referenzen betrachtet. Der Referenzbaum wird solange aufgebaut, wie die referenzierten Assets relevant für den Anwendungsfall sind. Wird ein Asset erreicht, das dieses Kriterium nicht erfüllt, so wird die Referenz „abgeschnitten“, d.h. sie wird bei der Synchronisation auf null gesetzt. Alle Assets, die von der Wurzel aus erreichbar sind, befinden sich in der Objekthülle, die für diesen Anwendungsfall gilt. Dazu zählen keine Assets, die über abgeschnittene Referenzen erreicht werden.

Beispiel:

Wir betrachten den *Anwendungsfall Aktualisierung von Vorschriften*. Bei diesem Anwendungsfall sollen alle Vorschriften auf dem lokalen Portal aktualisiert werden. Dazu werden die Vorschriften vom zentralen Portal auf das lokale übernommen, die neu hinzugekommen sind oder sich geändert haben. Als Wurzel-Knoten wird daher der Asset-Typ *Form* gewählt, denn ein Asset vom Typ *Form* repräsentiert eine Vorschrift. Dann wird der Referenzbaum aufgestellt. (siehe Abbildung 4.7)

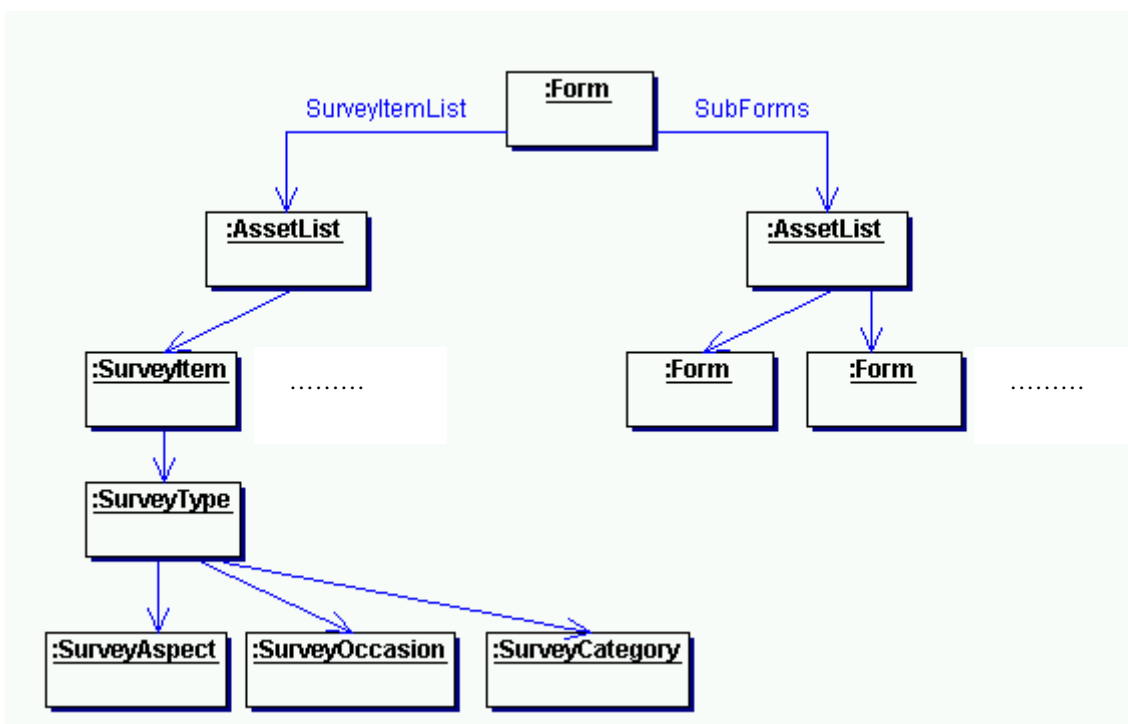


Abbildung 4.7: Beispiel : Referenzbaum – Vorschriften

Wie dem Referenzbaum entnommen werden kann, besitzt ein *Form* möglicher Weise mehrere SubForms, die in einer *AssetList* gespeichert sind. Diese sind wiederum vom Typ *Form*. Außerdem verweist ein *Form* auf eine *AssetList* (*SurveyItemList*), die mehrere Assets vom Typ *SurveyItem* enthalten kann, die wiederum auf andere Assets verweisen. Alle diese Assets befinden sich in der Objekthülle und werden, falls der Wurzel-Knoten *Form* synchronisiert wird, ebenfalls synchronisiert. In diesem Beispiel treten keine „abgeschnittenen“ Referenzen auf.

Nochmals sei erwähnt, dass einmal pro Anwendungsfall eine Hülle definiert wird, der zugehörige Referenzbaum mit konkreten Objekten aber pro Synchronisationsvorgang erstellt wird. Die Referenzbäume für alle Anwendungsfälle befinden sich im Anhang A.

Beim Aufstellen der Referenzbäume werden einige Referenzen abgeschnitten. Im Folgenden soll kurz erläutert werden, welche das sind und warum diese Referenzen abgeschnitten werden.

Asset Person:

Hier werden die Referenzen zu den Assets vom Typ *Membership*, *PortfolioItems*, *Relationship* abgeschnitten. Diese Assets weisen einer Person bestimmte Rechte zu, verwalten die Gruppenzugehörigkeiten, etc. Da sich die Personen, die im vorliegenden Szenario synchronisiert werden, nie am lokalen Portal des Besichtigers anmelden werden, sind diese Informationen überflüssig. Es handelt sich dabei um Personen wie Schiffseigner, Reeder, Auftraggeber.

Asset Inspection:

Hier wird die Referenz (*Report*) zum Asset vom Typ *Document* abgeschnitten. Diese Referenz wird im Portal zum Zeitpunkt der Studienarbeit nicht benutzt und kann daher abgeschnitten werden.

Auch die Referenzbäume, in denen die eben genannten Asset-Typen mit ihren „abgeschnittenen“ Referenzen vorkommen, sind in Anhang A zu finden.

Nach einem Synchronisationsvorgang müssen die Referenzen so angepasst werden, dass sie wieder die Objekthülle ergeben. Dies ist notwendig, da die Referenzen durch Speichern der ID des referenzierten Assets in einem Attribut modelliert werden, aber die Asset-IDs korrespondierender Assets auf lokalem und zentralem Portal unterschiedlich sein können.

4.3.2 Erkennen von Zustandsänderungen korrespondierender Assets

In der bisherigen Version des Synchronisationsmodells aus [Leh02] kann es zu Problemen beim Erkennen von Zustandsänderungen kommen. Wie in Kapitel 2.3 erläutert, wird die Zustandsänderung eines Assets durch den Vergleich des letzten Änderungszeitpunktes mit dem letzten Synchronisationszeitpunkt erkannt. Der Synchronisationszeitpunkt wird immer durch das lokale Portal gespeichert, da dieses den Synchronisationsablauf kontrolliert. Laufen die Systemuhren auf beiden Portalen unterschiedlich, was immer der Fall ist (speziell im weltweiten Szenario), so werden Änderungen möglicherweise nicht erkannt. Dazu wird folgendes Beispiel betrachtet:

Kapitel 5

Realisierung und Evaluation

Dieses Kapitel teilt sich in zwei Abschnitte auf. Im Abschnitt 5.1 wird die Realisierung des im vorigen Kapitel besprochenen Entwurfs behandelt. Dabei werden auftretenden Probleme beschrieben, sowie die Lösung dieser Probleme. Dieses Kapitel behandelt die programmiertechnischen Aspekte der Studienarbeit und soll klären, wie der Entwurf implementiert wurde, welche Klassen erstellt, oder ob existierende angepasst wurden.

Im Abschnitt 5.2 wird das Ergebnis dieser Studienarbeit ausgewertet. Es wird erläutert, wie gut die Anforderungen erfüllt worden sind und ob im nachhinein andere Entwurfsentscheidungen getroffen werden würden.

5.1 Realisierung

Auch dieser Abschnitt teilt sich in mehrere Unterabschnitte auf, die zum Teil einzelne Aspekte aus Kapitel 4 behandeln, zum anderen aber auch generelle Probleme, die während der Implementierung aufgetreten sind.

Da sowohl das WIPS Portal in Java programmiert wurde, wie auch das Synchronisationsmodell für die Plattform infoAsset Broker, wurde zur Realisierung ebenfalls Java (JDK 1.4) als Programmiersprache verwendet.

5.1.1 Die Klasse *ReferenceMapping*

Wie in Kapitel 2.2 erwähnt, werden Referenzen zwischen Assets in der zugrunde liegenden Plattform infoAsset Broker durch das Speichern der AssetID in einem Asset-Attribut realisiert. Da der infoAsset Broker in der bei der Studienarbeit verwendeten Version keine Möglichkeiten bereitstellt, Attribute, die Referenzen darstellen, als solche zu identifizieren, wurde diese Funktionalität hinzugefügt, soweit es für die Synchronisation notwendig war. Dazu wurde die Klasse *ReferenceMapping* implementiert.

Ein Objekt der Klasse *ReferenceMapping* kennt die während des Entwurfs definierten Objekthüllen. Die Klasse besitzt Methoden zum Abfragen der Referenzen eines Assets, solange diese nicht „abgeschnitten“ sind. Dazu verweist ein Objekt vom *ReferenceMapping* auf ein *HashMap*-Objekt (`references`, wie in Abbildung 5.1 unten zu sehen ist). Eine *HashMap* ist eine Java-Basisklasse, die Schlüssel-Werte-Paare (engl: Key-Value-Pair)

speichert, wobei Schlüssel und Wert jeweils vom Typ *Object* sind. Jedes Schlüssel-Objekt identifiziert dabei eindeutig ein Wert-Objekt.

Abbildung 5.1 zeigt die realisierte Struktur in einem Objektdiagramm:

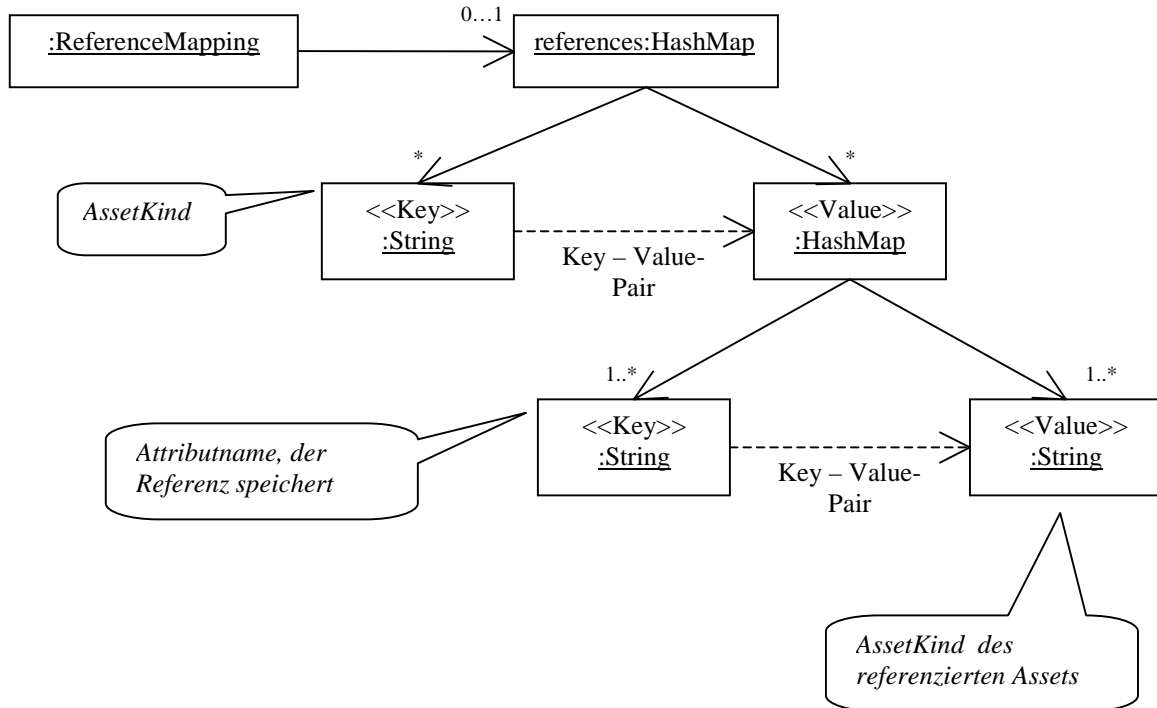


Abbildung 5.1: Objektdiagramm ReferenceMapping und HashMaps

Wie in Abbildung 5.1 zu sehen ist werden, speichert das *HashMap*-Objekt *references* als Wert-Objekte weitere *HashMap*-Objekte, wobei zu jedem *HashMap*-Objekt ein *String*-Objekt als Schlüssel dient. Es existiert zu jedem Asset-Typ, der im vorliegenden Anwendungskontext synchronisiert werden soll, ein Schlüssel-Wert-Paar, wobei der Schlüssel den Namen des Asset-Typs darstellt. Dabei werden aber nur solche Asset-Typen berücksichtigt, die nach Definition der Objekthülle und Aufstellen des Referenzbaumes (siehe Kapitel 4) Referenzen auf andere Assets besitzen, die ebenfalls in der Objekthülle liegen. Das zu jedem Schlüssel gehörende *HashMap*-Objekt speichert seinerseits wieder Schlüssel-Wert-Paare, bei denen Schlüssel und Wert ein *String*-Objekt sind. Dabei gibt der Schlüssel den Attributnamen eines Asset-Typs an, der eine Referenz im aufgestellten Referenzbaum darstellt, und der Wert gibt an, welchen Typ (*AssetKind*) das referenzierte Asset hat. Auf diese Art werden zu jedem Asset-Typen, der Referenzen besitzt, die während der Synchronisation verfolgt werden sollen, die Attributnamen der Attribute gespeichert, die Referenzen darstellen.

Die Klasse *ReferenceMapping* besitzt die Methode `getReferenceMappings(String assetKind)`, die als einen Aufrufparameter einen den Namen eines Asset-Typs übergeben bekommt. Bei Aufruf liefert sie ein *HashMap*-Objekt, das als Schlüssel-Wert-Paare die Attributnamen der Referenzen des im Parameter übergebenen Asset-Typs und die zugehörigen Asset-Typen der referenzierten Assets speichert oder `null`, falls keine Referenzen für diesen Asset-Typ gespeichert sind.

5.1.2 Realisierung des Synchronisationsprozesses

Dieser Abschnitt soll die Realisierung des Synchronisations-Prozesses beschreiben. Die zentralen Klassen des Synchronisationsmodells sind schon in Kapitel 2.3 besprochen worden. Hier soll nun beschrieben werden, wie diese Klassen zur Realisierung der Synchronisation verwendet werden, um die Anforderungen im vorliegenden Fall zu erfüllen. Zunächst wird beschrieben, welche Objekte zur Laufzeit des Systems erzeugt werden, um eine Synchronisation zu ermöglichen. Das Objektdiagramm (Abbildung 5.2, siehe S. 44) zeigt die wichtigsten Objekte.

Wie schon in Kapitel 2.3 erwähnt, repräsentiert ein Asset der Klasse *SynchronisationLink* eine Synchronisationsverbindung mit einem anderen Portal. Da im vorliegenden Szenario immer das lokale Portal die Synchronisation beginnt, werden die Objekte des Synchronisations-Clients dort erstellt. Vom Schiffsbesichtiger wird beim Anlegen einer Synchronisationsverbindung ein solches Asset erstellt. Es besitzt eine Referenz auf ein Objekt vom Typ *RemoteConnection*, welches die IP-Adresse und den Port des entfernten Portals kennt und die gesamte Kommunikationssteuerung übernimmt.

Die Klasse *SynchronisationTask* modelliert eine konkrete Synchronisationsaufgabe für eine Menge von Assets eines Asset-Typs. Die Benutzerschnittstelle wurde so implementiert, dass dem erstellten *SynchronisationLink*-Asset noch maximal neun *SynchronisationTask*-Assets (mindestens aber die vier, siehe unten) zugeordnet werden, je nachdem, ob der Schiffsbesichtiger eine Synchronisation der zugehörigen Asset-Typen wünscht. Einigen dieser Objekte sind *Searches* zugeordnet (siehe Kapitel 2.2 und 2.3), um die zu synchronisierenden Objektmengen weiter einzuschränken.

Durch die Analyse der Referenzbäume wurde festgestellt, dass gewisse Asset-Typen sehr häufig referenziert werden. Diese vier Asset-Typen werden bei jedem Synchronisationsvorgang synchronisiert. Es handelt sich dabei um die Typen: *Domain*, *DomainValue*, *MeasuredValue* und *ValueObject*. Diese Typen finden sich auch in der Abbildung 5.2 wieder. Je ein *SynchronisationTask* repräsentiert eine Synchronisationsaufgabe für einen dieser Typen und wird beim Anlegen einer Synchronisationsverbindung automatisch erstellt. Da diesen Tasks keine *Searches* zugeordnet sind, werden bei einer Synchronisation immer alle Assets dieser Typen vom zentralen auf das lokale Portal übernommen. Die anderen fünf Tasks sind dem *SynchronisationLink* nur zugeordnet, falls der Schiffsbesichtiger eine Synchronisation der zugehörigen Asset-Typen über die Benutzerschnittstelle eingestellt hat. In der Abbildung 5.2 ist dies der Fall. Dabei stellt ein *SynchronisationTask* jeweils eine konkrete Synchronisationsaufgabe eines Synchronisations-Anwendungsfalles dar (siehe Kapitel 3.3).

Es folgt nun eine Übersicht über die fünf dem Schiffsbesichtiger zur Auswahl stehenden *SynchronisationTasks*. Diese werden in der Abbildung 5.2 anhand des Asset-Typs, den sie synchronisieren, sowie der Synchronisationsrichtung (vgl. Kapitel 2.3) identifiziert.

AssetKind Form, Richtung Übernehmen: Es werden alle Vorschriften vom zentralen auf das lokale Portal übernommen. Anwendungsfall: Aktualisierung von Vorschriften.

AssetKind Inspection , Richtung Übernehmen: Es werden Besichtigungsaufträge vom zentralen Portal auf das lokale übernommen. Der zugeordnete Search grenzt die Menge der zu synchronisierenden Besichtigungsaufträge so ein, dass nur die dem Schiffsbesichtiger, der die Synchronisation gestartet hat, zugewiesenen Besichtigungsaufträge übernommen werden. Anwendungsfall: Eigene Auftragsliste aktualisieren.

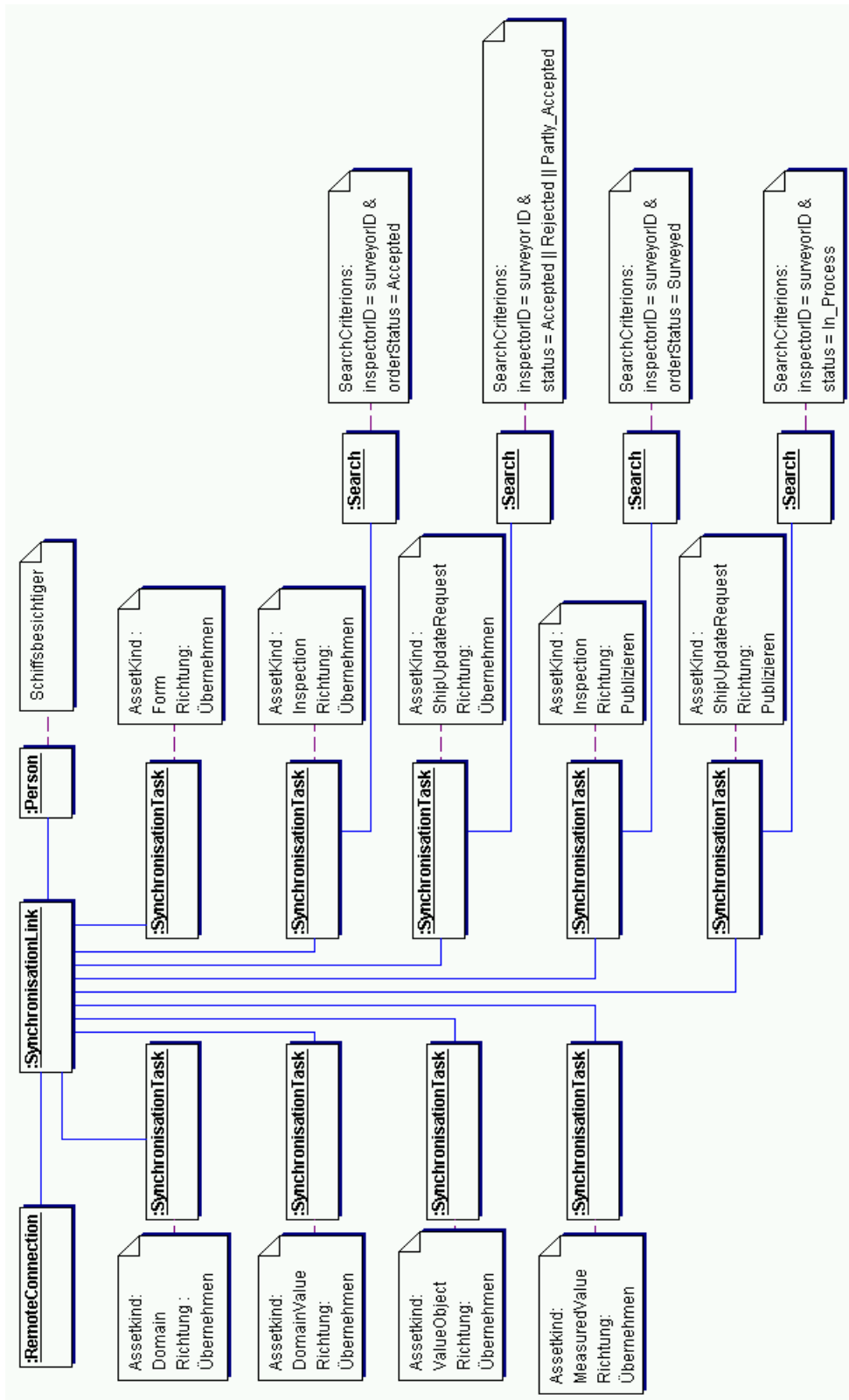


Abbildung 5.2: UML-Objektdiagramm - Synchronisationslink

AssetKind ShipUpdateRequest, Richtung Übernehmen: Es werden alle Änderungsanträge vom zentralen Portal auf das lokale übernommen. Der zugeordnete Search grenzt die Menge der zu synchronisierenden Änderungsanträge so ein, dass nur die vom Schiffsbesichtiger erstellten und in der Zentrale bearbeiteten Änderungsanträge übernommen werden. Anwendungsfall: Vom GL geänderte Schiffsdaten übernehmen.

AssetKind Inspection, Richtung Publizieren: Es werden alle Besichtigungsaufträge vom lokalen auf das zentrale Portal publiziert. Der zugeordnete Search grenzt die Menge der zu synchronisierenden Besichtigungsaufträge so ein, dass nur die vom Schiffsbesichtiger bearbeiteten (also durchgeführten) Besichtigungen publiziert werden. Anwendungsfall: Durchgeführte Besichtigungen publizieren.

AssetKind ShipUpdateRequest, Richtung Publizieren: Es werden alle Änderungsanträge vom lokalen auf das zentrale Portal publiziert. Der zugeordnete Search grenzt die Menge der zu synchronisierenden Änderungsanträge so ein, dass nur die vom Schiffsbesichtiger erstellten und zur Bearbeitung vorgesehenen Änderungsanträge publiziert werden. Anwendungsfall: Vom Besichtiger selbst geänderte Schiffsdaten publizieren.

Die in der Abbildung 5.2 gezeigten Assets werden erstellt, wenn der Schiffsbesichtiger die Einstellungen für die Synchronisation bearbeitet. Einen Sonderfall stellt hier die erstmalige Synchronisation mit einem Portal dar. Gibt der Schiffsbesichtiger eine IP-Adresse und einen Port bei der Bearbeitung der Synchronisationseinstellungen ein, die noch nie Ziel einer Synchronisation waren, so wird, bevor die eigentliche Synchronisation starten kann, eine Vorsynchronisation durchgeführt, um zu prüfen, ob die IP-Adresse gültig ist und ob der Schiffsbesichtiger sich auf dem entfernten Portal anmelden darf und welche AssetID das korrespondierende Personen-Asset dort hat (Bei dem Personen-Asset handelt es sich um das Informationsobjekt, welches Informationen über den Schiffsbesichtiger selbst speichert). Dann werden die vier Basis-Asset-Typen *Domain*, *DomainValue*, *ValueObject* und *MeasuredValue* synchronisiert. Erst danach werden die in der Abbildung 5.2 gezeigten Objekte erstellt und eine Synchronisation ist dann möglich.

5.1.3 Erweiterungen

Damit bei der Synchronisation auch die referenzierten Assets mitsynchronisiert werden, muss die Klasse *IMPSynchronisationEngine* geändert werden. Bei der Bearbeitung der *SynchronisationTasks* wird die Menge zu synchronisierenden Assets bestimmt. Dabei müssen auch die referenzierten Assets berücksichtigt werden. Dazu wurde im vorigen Unterabschnitt die Klasse *ReferenceMapping* eingeführt. Die Klasse *IMPSynchronisationEngine* wird nun so geändert, dass bei der Bestimmung der Objektmenge einmal für jedes zu synchronisierende Asset über die Methode `getReferenceMappings(String assetKind)` die referenzierten Assets sowie deren Referenzen bestimmt werden und mit in die Menge der zu synchronisierenden Assets aufgenommen werden.

Da die AssetIDs auf dem lokalen Portal nicht mit denen des zentralen Portals übereinstimmen, müssen die Referenzen der während der Synchronisation übertragenen Assets angepasst werden. Zu diesem Zweck wurde die Klasse *IMPSynchronisationAction* geändert. (siehe auch Kapitel 2) Die Methode `execute()` wurde erweitert, so dass sie die Referenzen anpasst. Es ist fest codiert, welche Referenzen angepasst werden. Mit der alten AssetID wird versucht, in der MappingTabelle der Synchronisation einen Eintrag zu identifizieren und so die AssetID auf dem lokalen Portal zu bestimmen. Wird eine AssetID

gefunden, so wird die Referenz angepasst. Andernfalls wird sie auf den Wert `null` gesetzt. Eine Nachübertragung und spätere Anpassung von Referenzen ist nicht vorgesehen.

5.1.4 Speicherung von zwei Zeitstempeln

Wie aus Kapitel 2 bekannt, repräsentiert ein Asset des Typs *AssetSynchronisationState* einen Eintrag in der Mapping-Tabelle. Diese Asset-Klasse wurde um zwei Attribute (`remoteObjectTimeStamps` und `localObjectTimeStamps`) erweitert, die die beiden Zeitstempel speichern, wie im Abschnitt 4.3.2 erläutert. Außerdem wurden Methoden zum Abfragen und Setzen der Attribute hinzugefügt. Die Klasse *IMPSynchronisationEngine* wurde so angepasst, dass sie die neuen Attribute verwendet.

5.1.5 Anpassung des AssetSerializers

Die Klasse *XMLAssetSerializer* ist eine Komponente der Plattform *infoAsset Broker*. Sie besitzt Methoden, um Assets in einen XML-Stream zu schreiben und aus einem XML-Stream zu lesen. Mithilfe dieser Methoden werden die Assets während der Synchronisation in XML-Streams serialisiert und ausgetauscht. Allerdings gibt es ein Problem. Wie die Referenzen angepasst werden, ist im Abschnitt 5.1.3 weiter oben beschrieben. Aber dies gilt nur für die Richtung vom zentralen Portal zum lokalen, also für das Übernehmen. In der anderen Richtung (Publizieren) ist es ein wenig schwieriger. Die Mapping-Tabelle der Synchronisation befindet sich auf dem lokalen Portal. Nur hier können die Referenzen angepasst, bzw. die AssetIDs in die korrespondierenden AssetIDs des zentralen Portals umgewandelt werden. Das Anpassen der Referenzen ist deshalb wichtig, weil ein publiziertes Asset sonst auf dem zentralen Portal sonst Referenzen gespeichert hätte, die dort zu keinem Asset passen (*dangling reference*).

Deshalb wurde die Klasse *XMLAssetSerializer* angepasst, so dass die Referenzen während des Serialisierens angepasst werden. Die Attribute eines Assets werden nacheinander serialisiert. Mithilfe der Klasse *ReferenceMapping* werden nun die Attribute identifiziert, die AssetIDs eines referenzierten Assets speichern. Bei diesen Attributen wird über die Mapping-Tabelle die korrespondierende AssetID des referenzierten Assets auf dem zentralen Portal bestimmt und dann anstelle des tatsächlichen Attributwerts serialisiert.

Einschränkung: Dieses Verfahren funktioniert nur, wenn tatsächlich ein Eintrag für die AssetID des referenzierten Assets auf dem zentralen Portal in der Mapping-Tabelle steht. Sollte eine Referenz auf ein Asset zeigen, welches auf dem zentralen Portal nicht existiert, so kann während des Serialisierens diese Referenz nicht angepasst werden. Da dies im vorliegenden Szenario aber nicht passiert (es werden nur *ShipUpdateRequests* neu erstellt, die aber nie von anderen Assets referenziert werden und selbst nur Referenzen auf Assets speichern, die durch eine Synchronisation vom zentralen auf das lokale Portal gelangt sind), muss dieses Problem nicht im Rahmen der Studienarbeit gelöst werden.

5.1.6 Sonderbehandlung für AssetListen

Ein Asset-Typ erfordert eine Sonderbehandlung während der Synchronisation. Es ist der Typ *AssetList*. Ein Asset vom Typ *AssetList* speichert mehrere andere Assets. Dies geschieht durch Speichern der AssetID in einem *String*-Attribut. Dabei werden die AssetIDs der zu speichernden Assets hintereinander in einem *String* gespeichert. Da der *String* nur eine bestimmte Anzahl von Zeichen speichern kann, wird eine

zweites *AssetList*-Objekt verwendet, sobald mehr Assets gespeichert werden müssen, als das String-Attribut IDs aufnehmen kann. Die ID dieses zweiten *AssetList*-Objekts wird in dem ersten ebenfalls in einem Attribut gespeichert. So entsteht eine Kette von *AssetList*-Objekten. Das folgende Klassendiagramm (Abbildung 5.3) zeigt das Konzept der AssetListen noch einmal.

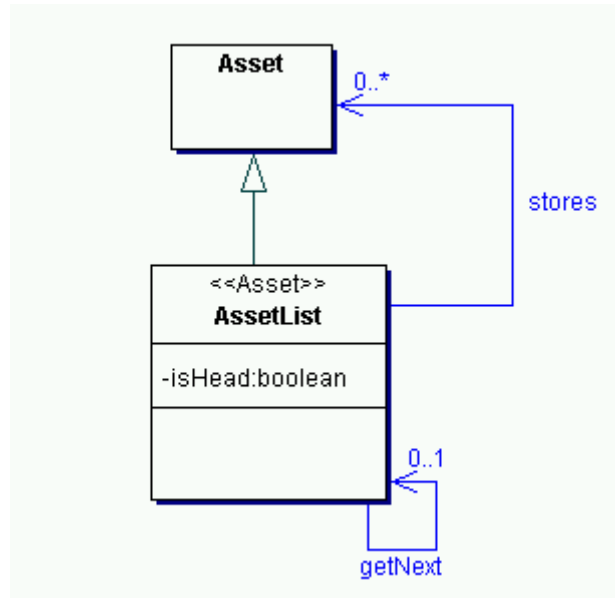


Abbildung 5.3: Konzeptuelles Klassendiagramm – AssetListen

Die AssetListen benötigen eine Sonderbehandlung, denn während der 1.Phase der Synchronisation müssen auch *SynchronisationObjects*(siehe Kapitel 2.3) für die enthaltenen Assets generiert werden. Zum Zweiten müssen natürlich auch die gespeicherten AssetIDs nach einer Synchronisation angepasst werden.

Die Klasse *IMPSynchronisationEngine* wurde so geändert, dass sie für jede AssetListe auch *SynchronisationObjects* für die enthaltenen Assets generiert. Sollte ein *AssetList*-Objekt noch eine Nachfolger-Objekt besitzen (*getNext* im Diagramm 5.3), so wird auch für dieses, sowie alle weiteren Nachfolger ein *SynchronisationObject* generiert.

Zur Anpassung der IDs wurde die Asset-Klasse *AssetList* um eine Methode *replaceAssetURIs(AssetURIReplacer replacer)* erweitert. Diese Methode bewirkt das Ersetzen der gespeicherten IDs. Sie verwendet das folgende Interface:

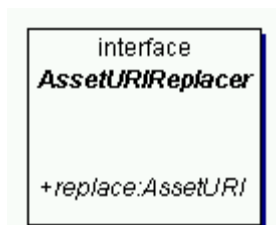


Abbildung 5.4: Interface AssetURIReplacer

Das Interface stellt eine Methode *replace(AssetURI uri)* bereit, die als Parameter eine AssetURI, das ist ein andere Form der AssetID, übergeben bekommt, und als Ergebnis ebenfalls eine AssetURI zurückliefert. Innerhalb der Synchronisationskomponenten

implementier die Klasse *IMPRemoteConnection* dieses Interface, denn sie verwaltet auch die Mapping-Tabelle.

Einschränkung: Dieses Verfahren funktioniert nur für die Richtung Übernehmen. In der anderen Richtung müssten die IDs der *AssetList*-Objekte angepasst werden, wenn sich die betreffende AssetListe lokal geändert hat. Da dies aber in der implementierten Lösung nicht der Fall ist, kann diese Funktionalität später implementiert werden.

5.1.7 Implementierung der Asset-Klasse ShipUpdateRequest

Die Klasse wurde so, wie im Entwurf besprochen, implementiert. Allerdings wurden zunächst nur ein paar ausgewählte Attribute eines Schiffes implementiert, die geändert werden können. Dies reicht aus, um die prototypische Implementierung und Funktion des Systems zeigen zu können. Weitere Attribute können noch ergänzt werden.

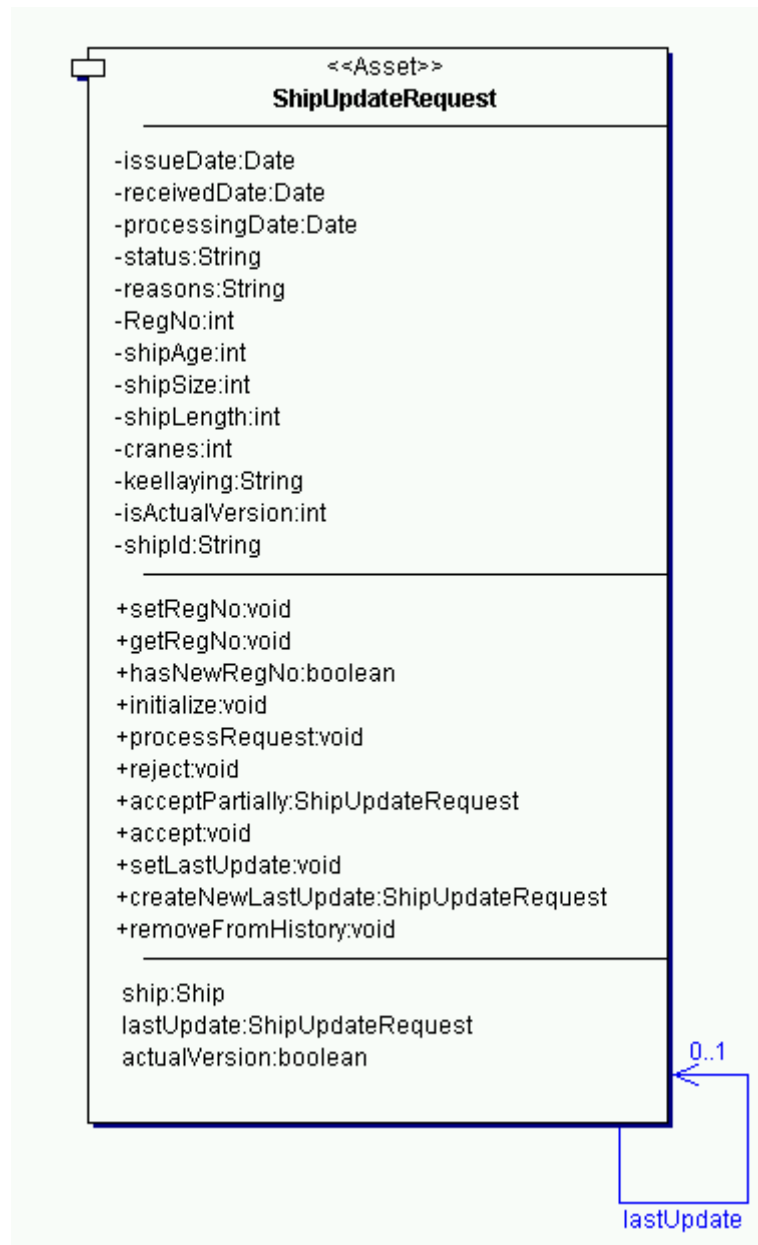


Abbildung 5.5: Klassendiagramm ShipUpdateRequest

Das letzte Diagramm (Abbildung 5.5) zeigt die implementierte Klasse. Einige Methoden wurden zur besseren Übersicht weggelassen. Dabei handelt es sich um die Methoden für den Zugriff auf die Meta-Informationen, die schon im Kapitel 4.1.7 angesprochen wurden. Zu den Attributen die aus den Schiffsdaten übernommen worden sind, zählen `regNo`, `shipAge`, `shipSize`, `shipLength`, `cranes`, `keelaying`. Für jedes dieser Attribute, die die Änderungen an Schiffsdaten repräsentieren, wurden drei Methoden implementiert, wie dies Abbildung 5.5 beispielhaft für das Attribut `regNo` gezeigt ist. Die Methode `setRegNo(int newRegNo)` dient zum Setzen, die Methode `getRegNo()` zum Abfragen des Attributs. Die Methode `hasNewRegNo()` dient dazu, abzufragen, ob eine Änderung dieses Attributs gewollt ist. Liefert die Methode `true` zurück, so soll das Attribut des referenzierten Schiffes in den in diesem Attribut gespeicherten Wert geändert werden. Bei `false` ist keine Änderung beantragt.

Die Methoden `setLastUpdate(ShipUpdateRequest request)`, `createNewLastUpdate()`, `getLastUpdate()`, `removeFromHistory()` und `isActualVersion()` ermöglichen die Verkettung von `UpdateRequest` und sollen nun kurz erläutert werden:

- `setLastUpdate(ShipUpdateRequest request)`: Die Methode setzt das im Parameter übergebene `UpdateRequest`-Objekt `request` als letztes `Update` („lastUpdate“ in Abb. 5.5) des `UpdateRequest`-Objektes, an welchem diese Methode aufgerufen wird.
- `getLastUpdate()`: Die Methode liefert das letzte `Update` des `ShipUpdateRequests` an welchem sie aufgerufen wird, fall dieses existiert, anderenfalls `null`.
- Die Methode `createNewLastUpdate()` erstellt einen neuen Änderungsantrag und setzt diesen als letztes `Update` des `UpdateRequest`-Objektes, an welchem diese Methode aufgerufen wird.
- Um ein `ShipUpdateRequest` aus der Verkettung zu löschen, kann die Methode `removeFromHistory()` verwendet werden. Sie löscht das `ShipUpdateRequest`-Objekt aus der Verkettung an dem die Methode aufgerufen wird.
- Die Methode `isActualVersion()` prüft, ob das `UpdateRequest`-Objekt, an dem sie aufgerufen wird, Kopf einer Verkettung ist und gibt `true` zurück, wenn das der Fall ist, ansonsten `null`.

Die Methoden `initialize(Ship ship, Person surveyor)`, `processRequest()`, `reject(Person person)`, `accept(Person person)` und `acceptPartially(Person person)` sind implementiert worden, um den Bearbeitungsvorgang zu erleichtern:

- Zum Initialisieren wird die Methode `initialize(...)` verwendet. Sie setzt die Referenzen für `Ship` und `Person` (Besichtiger). Damit ist ein Änderungsantrag eindeutig einem Schiff und einer Person, die ihn erstellt hat, zugewiesen.
- Durch die Methode `processRequest()` wird ein Änderungsantrag zur Bearbeitung durch die Zentrale markiert. Dies bedeutet, dass er bei der nächsten Synchronisation vom lokalen auf das zentrale Portal publiziert wird.

- Die Methode `reject(Person person)` weist einen Änderungsantrag ab. Als Parameter wird die Person übergeben, die den Änderungsantrag bearbeitet hat.
- Die Methode `accept(Person person)` nimmt einen Änderungsantrag an und ändert die Daten des zugehörigen Schiffes entsprechend dem Antrag. Als Parameter wird die Person übergeben, die den Änderungsantrag bearbeitet hat.
- Die Methode `acceptPartially(Person person)` markiert einen Änderungsantrag als „teilweise angenommen“ und generiert einen neuen Änderungsantrag, der dann den Teil der Änderungen enthalten soll, die angenommen werden. Die muss aber explizit zu Laufzeit zugewiesen werden. Als Parameter wird die Person übergeben, die den Änderungsantrag bearbeitet hat.

5.1.8 Lineare Versionierung der Schiffsdaten

Die lineare Versionierung von Schiffsdaten wurde so, wie im Entwurf beschrieben, implementiert. Dazu wurde die Asset-Klasse *IMPShip* um zwei Attribute und mehrere Methoden zum Verwalten der Versionierung erweitert. Das Attribut `predecessorId` speichert die AssetID einer Vorgänger-Version der Schiffsdaten. Das Attribut `isActualVersion` gibt an, ob es sich bei einem Schiffsdaten-Asset um die aktuelle Version oder um eine Vorgänger-Version handelt. Das bedeutet, dass verschiedene Versionen eines Schiffes auf dem gleichen Portal existieren können, repräsentiert durch verkettete Assets des Typs *Ship*.

Die hinzugefügten Methoden sollen hier nicht besprochen werden, da ihre Funktion ähnlich der ist, die die Methoden zum Verketteten von *ShipUpdateRequests* im vorigen Abschnitt bereitstellen.

5.1.9 Die Benutzerschnittstelle

Zur Entwicklung einer prototypischen Lösung gehört auch die Realisierung einer Benutzerschnittstelle zur Benutzung des Systems, in diesem Fall zur Synchronisation. Die Benutzerschnittstellen werden im WIPS Portal durch Templates und Handler bereitgestellt werden (siehe auch Kapitel 2.2).

Es wurden im Rahmen der Studienarbeit Handler und Templates erstellt für:

- die Bearbeitung der Synchronisationseinstellungen durch den Schiffsbesichtiger
- die Durchführung eines Synchronisationsvorganges durch einen Besichtiger
- die Bearbeitung der eigenen Besichtigungsaufträge durch einen Besichtiger
- das Erstellen eines Änderungsantrages an Schiffsdaten durch einen Besichtiger
- das Bearbeiten eines Änderungsantrages durch einen GL Mitarbeiter
- die Ansicht von Vorschriften durch einen Schiffsbesichtiger

Es folgen nun fünf Screenshots, die ausgewählte Teile der implementierten Benutzerschnittstelle zeigen.

Synchronisationseinstellungen

Es existiert noch keine Synchronisationsverbindung !

Name : SynchronisationLink
 Beschreibung : Informationsaustausch mit dem zentralen Portal beim GL.
 RemotePortal-IP :
 Port :

Einstellungen für die automatische Synchronisation :
 - noch nicht unterstützt -

Auswahl der zu synchronisierenden Daten :

Aktualisierung der Vorschriften	<input checked="" type="checkbox"/>	(Übernehmen)
Auftragsbuch aktualisieren	<input checked="" type="checkbox"/>	(Übernehmen)
Bearbeitete Änderungsanträge abfragen	<input checked="" type="checkbox"/>	(Übernehmen)
Besichtigungsergebnisse publizieren	<input checked="" type="checkbox"/>	(Publizieren)
Schiffsdaten-Änderungsanträge senden	<input checked="" type="checkbox"/>	(Publizieren)

Abbildung 5.6: Screenshot – Synchronisationseinstellungen

Die Abbildung 5.6 zeigt den Dialog, den der Schiffsbesichtigter bei der Bearbeitung der Synchronisationseinstellungen sieht. Er kann hier IP-Adresse und Port des zentralen Portals eingeben, sowie die Informationen auswählen, die bei einer Synchronisation aktualisiert werden sollen.

Die folgende Abbildung 5.7 zeigt einen Besichtigungsauftrag vor der der Besichtigung. Oben stehen Informationen über Schiff und Ankunftszeit, weiter unten Auftraggeber, sowie durchzuführende Besichtigungspunkte (SurveyItems).

Die Abbildung 5.8 zeigt denselben Besichtigungsauftrag nach der Besichtigung.

Besichtigungsauftrag

Fällig am :14. August 2003

Hafen : Fort Lauderdale, United States of America

Status : Accepted

Schiff: [zu den Schiffsdaten]		Besitzer:	
Name :	Titanic	Name :	Reeder, Uwe
Registernummer:	101386	Telefon :	
Schiffstyp:	UNSINKABLE SHIP	Fax :	
ETA :	Thu Aug 14 00:00:00 CEST 2003	Mobil :	
ETD :	Thu Aug 14 00:00:00 CEST 2003	Email :	reeder@reederel.com

Client :		Agent :	
Name :	Reeder, Uwe	Name :	Y, Agent
Telefon :		Telefon :	
Fax :		Fax :	
Mobil :		Mobil :	
Email :	reeder@reederel.com	Email :	agenty@gl.org

Surveyer

Hupe, Patrick

Besichtigung :

Kurze Beschreibung :

ClientReference :

ValidityStart : Mon Jul 14 00:00:00 CEST 2003

ValidityEnd : Wed Jul 14 00:00:00 CEST 2004

Auszuführende Arbeiten :

SurveyType	LastSurveyDate	RangeStart	RangeEnd
Passenger Lift/Initial/Statutory, last survey: 2.5.2002, due: 2.5.2003 [1.4.2003-1.6.2003]	2. Mai 2002	1. April 2003	1. Juni 2003
Safety Equipment/Annual/Statutory, due: 12.5.2003 [1.4.2003-1.6.2003]	[n.a.]	1. April 2003	1. Juni 2003
Certificate of Fitness (IGC) (HSSC)/Intermediate/Statutory, due: 22.5.2003 [1.4.2003-1.6.2003]	[n.a.]	1. April 2003	1. Juni 2003

Liste der einzelnen Besichtigungspunkte :

SurveyItem - Key	SurveyItem-Description	InspectionValue
SurveyItem/2256	Source of emergency power for electrical pumps	[not surveyed]
SurveyItem/26	Small hatches, weathertight doors, skylights	[not surveyed]
SurveyItem/2239	Automatic stop of ventilation fans serving the protected space	[not surveyed]
SurveyItem/585	The cargo heating / cooling*) system is in satisfactory condition.	[not surveyed]
SurveyItem/2295	Fire detection and fire alarm system provided	[not surveyed]
SurveyItem/914	Main engine(s) slow-down	[not surveyed]
SurveyItem/2388	Piping systems and valves for cargo tanks, pump room(s) and the fixed fire fighting system externally examined and found in order.	[not surveyed]
SurveyItem/1787	Protection devices and switching equipment are located in a non-hazardous area	[not surveyed]

Abbildung 5.7: Screenshot – Besichtigungsauftrag vor der Besichtigung

Besichtigungsauftrag

Fällig am :14. August 2003

Hafen : Fort Lauderdale, United States of America

Status : Surveyed

Schiff:	Besitzer:
Name : Titanic [zu den Schiffsdaten]	Name : Reeder, Uwe
Registernummer: 101386	Telefon:
Schiffstyp: UNSINKABLE BHP	Fax:
ETA : Thu Aug 14 00:00:00 CEST 2003	Mobil:
ETD : Thu Aug 14 00:00:00 CEST 2003	Email : reeder@reederel.com

Client:	Agent:
Name : Reeder, Uwe	Name : Y,Agent
Telefon:	Telefon:
Fax:	Fax:
Mobil:	Mobil:
Email : reeder@reederel.com	Email : agenty@gl.org

Surveyer

Hupe, Patrick

Besichtigung :

Kurze Beschreibung :

ClientReference :

ValidityStart : Mon Jul 14 00:00:00 CEST 2003

ValidityEnd : Wed Jul 14 00:00:00 CEST 2004

Auszuführende Arbeiten :

SurveyType	LastSurveyDate	RangeStart	RangeEnd
Passenger Lift/Initial/Statutory, last survey: 14.8.2003, due: 2.5.2003 [1.4.2003-1.6.2003]	14. August 2003	1. April 2003	1. Juni 2003
Safety Equipment/Annual/Statutory, last survey: 14.8.2003, due: 12.5.2003 [1.4.2003-1.6.2003]	14. August 2003	1. April 2003	1. Juni 2003
Certificate of Fitness (IGC) (HSSC)/Intermediate/Statutory, last survey: 14.8.2003, due: 22.5.2003 [1.4.2003-1.6.2003]	14. August 2003	1. April 2003	1. Juni 2003

Liste der einzelnen Besichtigungspunkte :

SurveyItem - Key	SurveyItem-Description	InspectionValue
SurveyItem/2256	Source of emergency power for electrical pumps	Battery
SurveyItem/26	Small hatches, weathertight doors, skylights	OK
SurveyItem/2239	Automatic stop of ventilation fans serving the protected space	NA
SurveyItem/585	The cargo heating / cooling*) system is in satisfactory condition.	OK
SurveyItem/2295	Fire detection and fire alarm system provided	OK
SurveyItem/914	Main engine(s) slow-down	14 s
SurveyItem/2388	Piping systems and valves for cargo tanks, pump room(s) and the fixed fire fighting system externally examined and found in order.	OK
SurveyItem/1787	Protection devices and switching equipment are located in a non-hazardous area	NOK

Abbildung 5.8: Screenshot – Besichtigungsauftrag nach durchgeführter Besichtigung

Auf der nächsten Abbildung 5.9 wird ein Dialog gezeigt, den der Schiffsbesichtigter sieht, wenn er einen Änderungsantrag erstellt. Auf der linken Seite sind die aktuellen Daten eines Schiffes zu sehen. Auf der rechten Seite trägt der Schiffsbesichtigter die Daten ein, die geändert werden soll.

Ship information

If you want to Create an UpdateRequest, please fill out the fields on the right, according to the attributes you want to change

Name	Titanic	UpdateRequest
Registration Number	101386	-
Owner	Uwe Roeder	
Phone	00441711234	
Flag	Great Britain	
Ship Type	UNSINKABLE SHIP	
Ship Length	100	110
Ship Age	90	19
Ship Size	20	13

Abbildung 5.9: Screenshot – Änderungsantrag

Zum Schluss wird noch ein Dialog (Abbildung 5.10) gezeigt, den ein Mitarbeiter der Hauptverwaltung des GL sieht, wenn er einen Änderungsantrag bearbeitet. Er kann dort auswählen, welche Änderungen akzeptiert werden, er kann Gründe für die Ablehnung bestimmen, und er kann den letzten Änderungsantrag festlegen (Verkettung).

Zu ändernde Daten :

(Markieren Sie die Änderungen, die teilweise übernommen werden sollen)

RegNo soll geändert werden in :	Keine Änderung beantragt	<input type="checkbox"/>
length soll geändert werden in :	110	<input type="checkbox"/>
size soll geändert werden in :	19	<input type="checkbox"/>
age soll geändert werden in :	13	<input type="checkbox"/>
cranes soll geändert werden in :	Keine Änderung beantragt	<input type="checkbox"/>

Gründe für eine Ablehnung/teilweise Annahme :

Bitte wählen Sie den letzten bearbeiteten Änderungsantrag für dieses Schiff aus :

- Keine vorangegangenen Änderungen - ▾

Abbildung 5.10 Screenshot – Bearbeitung eines Änderungsantrages

5.2 Evaluation

In diesem Abschnitt soll geklärt werden, ob alle Anforderungen erfüllt worden und wie gut sie erfüllt wurden und ob der Autor nach Entwurf und Implementierung jetzt andere Entwurfsentscheidungen treffen würde.

Es werden noch mal die Anforderungen aus Kapitel 3.2 betrachtet und dabei erläutert, wie gut jede einzelne Anforderung realisiert ist:

- *Synchronisation weitgehend ohne Benutzerinteraktion:* Diese Anforderung ist vollständig erfüllt. Der Schiffsbesichtiger muss nach dem Starten der Synchronisation nicht mehr in den Synchronisationsvorgang eingreifen. Was einzig in diesem Punkt zu bemängeln ist, ist eine fehlende Benutzerinformation, über die Zeit bis zum Ende der Synchronisation. Diese Funktion wurde im Rahmen der Studienarbeit nicht berücksichtigt.
- *Offline-Arbeit muss möglich sein:* Auch dieser Punkt ist vollständig erfüllt. Für die Arbeit mit dem lokalen WIPS Portal und seinen gespeicherten Informationen ist keine Internetverbindung notwendig. Diese muss nur zur Synchronisation mit dem zentralen WIPS Portal hergestellt werden.
- *Automatische/manuelle Synchronisation:* Eine manuelle Synchronisation ist möglich. Eine automatische Synchronisation ist zwar vorgesehen, konnte aber im Rahmen der Studienarbeit aus Zeitgründen nicht verwirklicht werden.
- *Synchronisation möglichst konfliktfrei:* Diese Anforderung ist erfüllt. Zwar lassen sich Konflikte bei der Synchronisation im generellen nicht vermeiden, solange auf beiden Synchronisationsseiten die Informationen unabhängig voneinander geändert werden können, doch speziell in diesem Anwendungskontext liegt, durch die Einführung eines neuen Asset-Typs (*ShipUpdateRequest*) und durch die beim GL gegebenen Arbeitsabläufe der Fall vor, dass Informationen jeweils nur auf einer Seite zur gleichen Zeit geändert werden. Dadurch lassen sich auch Synchronisationskonflikte vermeiden.
- *Änderung der Schiffsdaten muss möglich sein:* Auch diese Anforderung ist erfüllt. Zwar lassen sich die Schiffsdaten durch den Schiffsbesichtiger nicht direkt ändern, sondern nur durch die Erstellung eines Änderungsantrages, der dann von der Zentrale genehmigt werden muss, aber da durch diese Maßnahme Synchronisationskonflikte verhindert werden, kann dieser Nachteil vernachlässigt werden.

Als nächstes sollen die Anwendungsfälle betrachtet werden. Zwei Anwendungsfälle wurden in der Entwurfs- und Realisierungsphase nicht berücksichtigt. Es handelt sich dabei um „Besichtigungsergebnisse eines zu besichtigenden Schiffes übernehmen“ und „Besichtigungsergebnisse eines baugleichen Schiffes übernehmen“. Diese beiden Anwendungsfälle konnten aus Zeitgründen leider während der Studienarbeit nicht implementiert werden. Da sie aber nicht als notwendig eingestuft worden sind (siehe Kapitel 3.2), ist diese Entscheidung vertretbar.

Zusammenfassend lässt sich sagen, dass das entwickelte System den Zweck erfüllt, zu dem es entworfen wurde. Es synchronisiert Besichtigungsinformationen und erlaubt so die elektronische Bearbeitung der Besichtigungsaufträge durch den Schiffsbesichtigter.

Zur Architektur des Systems lässt sich sagen, dass diese Lösung speziell auf die Anforderungen und Bedürfnisse im vorliegenden Anwendungskontext angepasst ist und keinesfalls als allgemeine Lösung zur Synchronisation zwischen Portalen verwendet werden kann. Das zugrunde liegende Synchronisationsmodell baut auf vielen Konzepten der Plattform infoAsset Broker auf und kann nicht für beliebige Portale verallgemeinert werden. Dazu kommt, dass die entwickelte Lösung, um referenzierte Assets bei der Synchronisation zu berücksichtigen, nicht allgemein einsetzbar ist. Es handelt sich vielmehr um eine spezielle Lösung, die nur in diesem Anwendungskontext eingesetzt werden kann. Dies ist ein Nachteil dieser Lösung, denn die Wiederverwendbarkeit von Software-Lösungen spielt beim Entwickeln von Software eine wichtige Rolle.

Bewertung:

Die Entwurfsentscheidung, das bestehende Synchronisationsmodell zur Synchronisation zwischen den Portalen einzusetzen, war richtig. Man muss jedoch überlegen, ob man gerade bei dieser Situation (ein zentrales Portal, viele lokale Portale) nicht eine Lösung entworfen hätte, die mit einer zentralen AssetID-Vergabe arbeitet. Für die Offline-Arbeit hätte man noch eine vorläufige AssetID-Vergabe durch die lokalen Portale während der Offline-Phasen, sowie ein Protokoll zum ID-Austausch mit dem zentralen Portal entwickeln müssen. Natürlich hätte eine solche Lösung mehr Aufwand bedeutet und wäre vermutlich auch nicht im Rahmen der Studienarbeit zu implementieren gewesen, aber sie hätte entscheidende Vorteile gehabt: Es wären keine Mapping-Tabellen mehr zur Synchronisation notwendig, ein Asset hätte auf allen Portalen dieselbe ID und die Synchronisation wäre schneller, da das Abgleichen der AssetIDs entfiel. Gerade das Anpassen der AssetIDs hat während der Implementationsphase für viel Arbeit gesorgt, da bereits eine nicht angepasste AssetID zu Fehlfunktionen des Portal-Systems führen kann.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel soll zunächst eine Zusammenfassung dieser Studienarbeit geben. (Abschnitt 6.1) Im Abschnitt 6.2 folgt dann abschließend ein Ausblick über die Erweiterungs- und Verbesserungsmöglichkeiten der realisierten Synchronisationslösung.

6.1 Zusammenfassung

Im Rahmen dieser Studienarbeit ist ein Synchronisationsmechanismus zur Synchronisation von Besichtigungsinformationen zwischen zwei WIPS IT1.1 Informationsportalen (siehe auch [wips]) entworfen und prototypisch realisiert worden.

Dazu wurde zunächst eine Übersicht über verschiedene Portalformen gegeben und der Begriff des Informationsportals erläutert. Anschließend wurde die Portalplattform infoAsset Broker (vgl. [iA]), auf der das WIPS IT1.1 Informationsportal basiert, sowie ein existierendes Synchronisationsmodell (siehe auch [Leh02]) vorgestellt.

Danach sind die Anforderungen an eine zu entwickelnde Lösung untersucht und die Anwendungsfälle der Synchronisation zusammengestellt worden. Die Anforderungsanalyse ergab, dass folgende Anforderungen besonders wichtig sind (siehe auch Kapitel 3.2):

- das Offline-Arbeiten am lokalen Portal muss möglich sein,
- der Synchronisationsvorgang muss ohne Benutzerinteraktion stattfinden können,
- der Synchronisationsvorgang soll möglichst konfliktfrei sein,
- Automatische oder manuelle Synchronisation und
- Änderungen an den Schiffsdaten durch einen Schiffsbesichtiger sollen möglich sein

An die Anforderungsanalyse schloss sich der Entwurf einer Lösung an. Dazu wurden zunächst verschiedene Entwurfsalternativen untersucht und ihre Vor- und Nachteile aufgezählt. Im Anschluss wurde dann ein konkreter Entwurf präsentiert, der sich durch folgende Eckpunkte beschreiben lässt:

- Modellierung von ShipUpdateRequests für in Konflikt stehende Objekte,
- Lineare Versionierung von Schiffsdaten,
- Verwendung und Erweiterung des Synchronisationsmodells aus [Leh02],
- Definition einer Objekthülle pro Anwendungsfall, um referenzierte Assets mit zu synchronisieren und
- Speicherung der Änderungszeitpunkte korrespondierender Informationsobjekte

Die entworfene Lösung wurde dann prototypisch realisiert, wobei auch Änderungen an den Basisklassen der Portalplattform vorgenommen werden mussten. Ferner wurde zur Nutzung der Synchronisationskomponenten eine Benutzerschnittstelle realisiert. Die Benutzerschnittstelle ist speziell zur Nutzung durch Schiffsbesichtiger vorgesehen.

6.2 Ausblick

Dieser Abschnitt gibt einen Ausblick über die Verbesserungs- und Erweiterungsmöglichkeiten der realisierten Lösung. Die realisierte Lösung hat sich an den Anforderungen orientiert, so dass einige der hier beschriebenen Erweiterungen nicht berücksichtigt wurden. Andere Verbesserungsmöglichkeiten haben sich während der Realisierungsphase ergeben.

Eine wünschenswerte Erweiterung wäre, dass auch persönliche Informationen des Schiffsbesichtigers synchronisiert werden. Die realisierte Lösung hat die Einschränkung, dass sie nur für Besichtigungsinformationen entwickelt wurde. Wird die Lösung jedoch so angepasst, dass auch persönliche Informationen des Schiffbesichtigers (Telefonnummer, Anschrift, etc) synchronisiert werden können, so ist eine Änderung der Daten ebenfalls lokal und offline möglich. Die Änderungen würden dann bei der nächsten Synchronisation auf dem zentralen Portal publiziert.

Eine weitere Verbesserungsmöglichkeit würde sich ergeben, wenn man das System um einen zentralen ID-Server erweiterte, der in diesem Szenario dem zentralen Portal anzugliedern wäre. Die Vorteile dieses Ansatzes sind schon bei der Evaluation in Kapitel 5.2 beschrieben worden. Ziel dieses Ansatzes soll es sein, eine Mapping-Tabelle bei der Synchronisation (siehe auch Kapitel 2.3) überflüssig zu machen dadurch, dass die Asset-IDs von korrespondierenden Assets auf allen beteiligten Portalen gleich sind. Das spart Zeit während der Synchronisation, denn die IDs müssen nicht mehr nach jeder Synchronisation angepasst werden. Zum anderen lassen sich auch referenzierte Assets leichter mitsynchronisieren, denn auch hier entfällt eine ID-Anpassung. Natürlich wäre in einem solchen Fall ein Protokoll zur Zuteilung einer ID durch den ID-Server zu entwickeln und es müssten vorübergehende IDs durch die lokalen Portale erteilt werden können, die während der Offline-Arbeitsphasen vergeben werden. Die vorübergehenden IDs würden dann bei der nächsten Synchronisation durch endgültige, vom ID-Server zugeteilte IDs ersetzt.

Mit der neuen infoAsset Broker-Version 2.0 (siehe auch [Ger02]) wird sich auch die Identifizierung von Attributen, die Asset-IDs speichern, verbessern. Die Version stellt

Methoden bereit, die diejenigen Attribute eines Assets liefern, die Referenzen zu anderen Assets darstellen. Bei einer Migration des WIPS IT1.1 Informationsportals auf diese Broker-Version könnten diese neuen Dienste genutzt werden. Das Identifizieren der Attribute durch die dafür eigens eingeführte Klasse *ReferenceMapping* könnte dann entfallen. Dies brächte den Vorteil, dass bei einer Änderung der Struktur eines Assets (Hinzufügen von Referenzen, Entfernen von Referenzen) die Klasse nicht jedes Mal angepasst werden müsste. Im Augenblick ist fest codiert, welche Attribute Referenzen darstellen. Eine Anpassung bedeutet hierbei, dass die Klasse neu kompiliert und das Portal neu gestartet werden muss. Die geänderten Klassen müssen auch in alle lokalen Portale publiziert werden, was zu einem hohen Aufwand bei der Systempflege führt. Kurzzeitig könnte man darüber nachdenken, die Attribute, die Referenzen darstellen, durch die Klasse *ReferenceMapping* aus einer Konfigurationsdatei einlesen zu lassen. Dieses Verfahren würde die Systempflege erleichtern, aber langfristig sollte die Migration auf die Broker-Version 2.0 geplant werden.

Als letzter Verbesserungspunkt wird ein eigenes Serialisierungsformat für die Asset-Listen präsentiert. Die Asset-Listen sind zwar ebenfalls Assets, müssen aber aufgrund ihrer Struktur eine Sonderbehandlung erfahren. In Kapitel 5.1.6 wird diese Sonderbehandlung detailliert beschrieben. Da Asset-Listen im Grunde genommen nur Asset-IDs speichern, sind bei der Synchronisation auch nicht die Asset-Listen an sich von Interesse, sondern nur die gespeicherten IDs. Da aber Asset-Listen in dem realisierten Prototyp wie gewöhnliche Assets serialisiert und in einen Stream geschrieben werden, werden unnötige Informationen mittransportiert, da sämtliche Attribute serialisiert werden. Unter Umständen müssen sogar mehrere Asset-Listen serialisiert werden, wenn diese verkettet sind (siehe Kapitel 5.1.6). Dies lässt sich verhindern, wenn ein eigenes Serialisierungsformat für Asset-Listen eingeführt wird, so dass nur die gespeicherten Asset-IDs serialisiert werden. Bei der Deserialisierung wird dann aus diesen Asset-IDs wieder eine verkettete Struktur mit Asset-Listen erstellt.

Die vorliegende Arbeit beschäftigte sich mit der Synchronisation zwischen Portalen, die dieselbe Struktur aufweisen. In Zukunft wird es aber vermehrt auch zur Synchronisation zwischen Portalen kommen, auf denen die Informationsobjekte unterschiedliche Strukturen aufweisen. Dies ist besonders der Fall zwischen leichtgewichtigen persönlichen Portalen, die zur Speicherung der persönlichen Informationen dienen, und großen Informationsportalen die eine Fülle von Informationen und Diensten anbieten. In diesem Bereich besteht noch weiterer Forschungsbedarf.

Anhang A

Referenzbäume für alle Anwendungsfälle

Für alle Abbildungen gilt: Mit einem * gekennzeichnete Referenzen sind so zu verstehen, dass beliebig viele (auch keine) Assets der Asset-Typen, auf die der Pfeil zeigt, referenziert werden.

Anwendungsfälle: „Eigene Auftragsliste aktualisieren“ und „Durchgeführte Besichtigungen publizieren“.

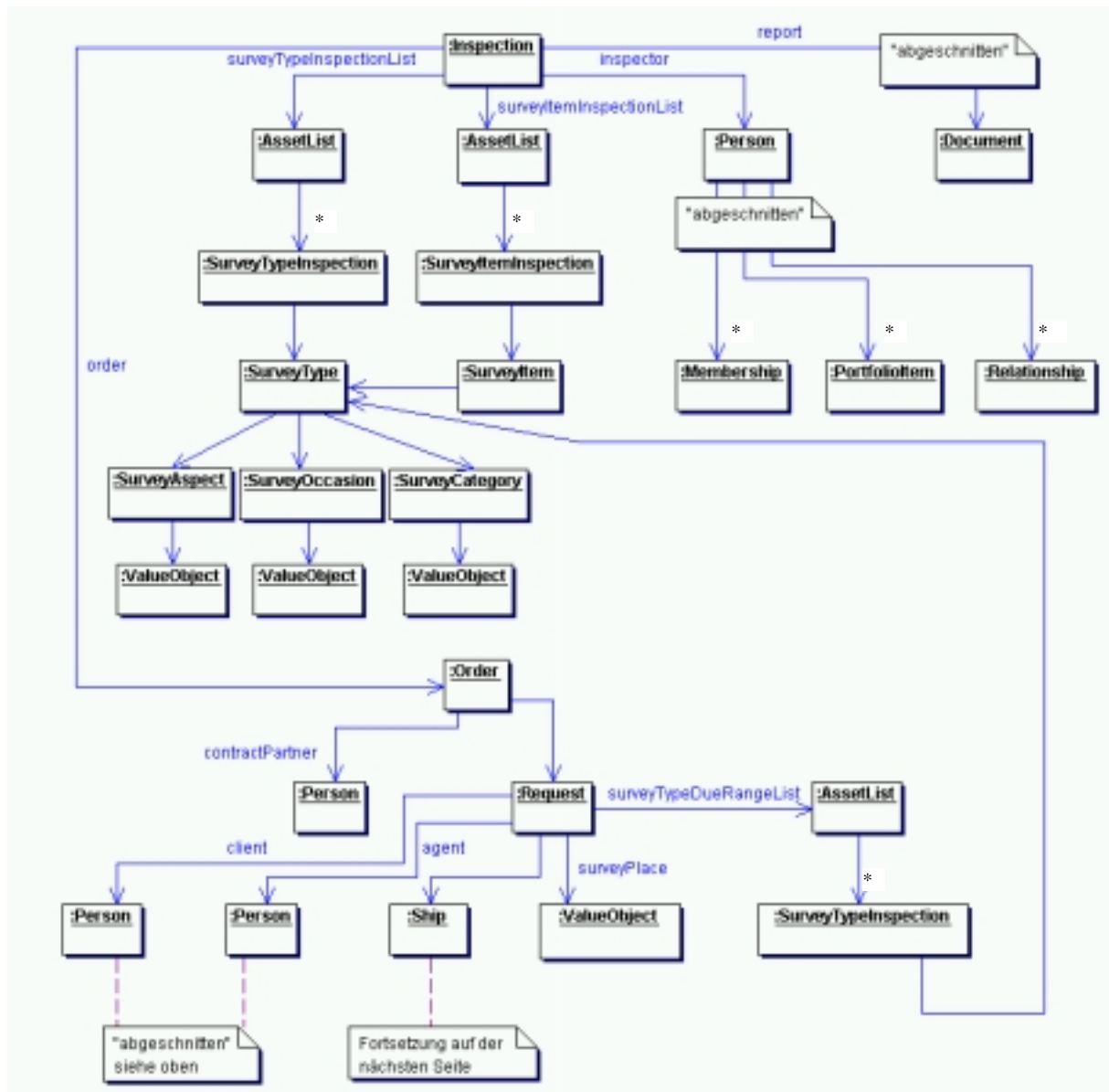


Abbildung A.1: Referenzbaum - Besichtigungen

Fortsetzung von letzter Seite :

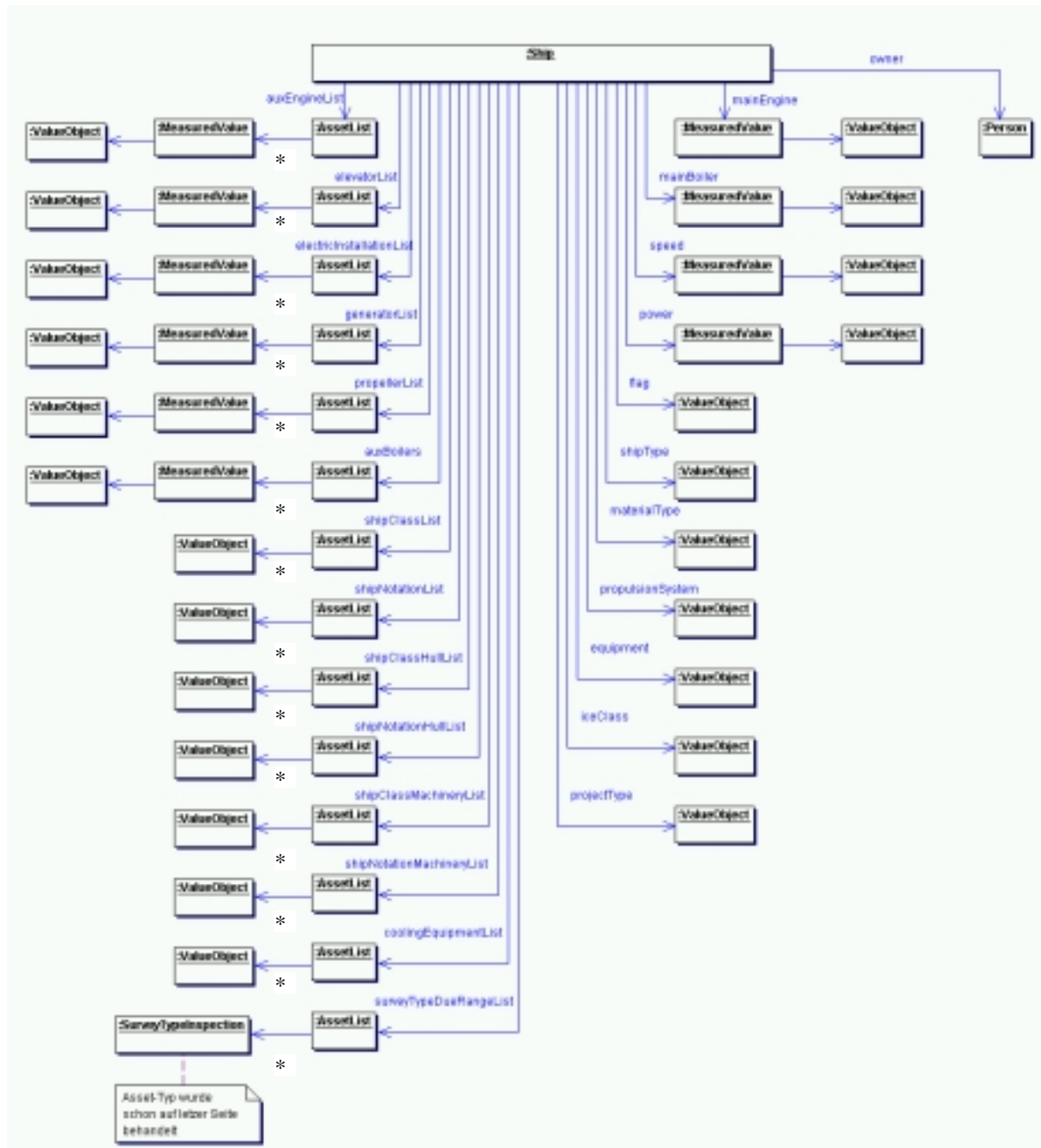


Abbildung A.2: Teil- Referenzbaum – Schiff

Anwendungsfall: „Aktualisierung von Vorschriften“

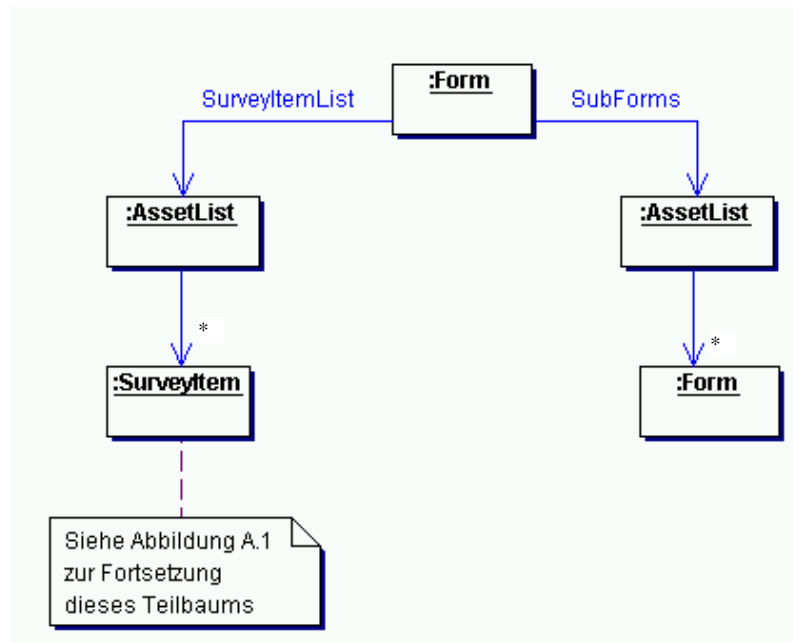


Abbildung A.3: Referenzbaum – Vorschriften

Anwendungsfälle: „Vom GL geänderte Schiffsdaten übernehmen“ und „Vom Besichtigter selbst geänderte Schiffsdaten publizieren“

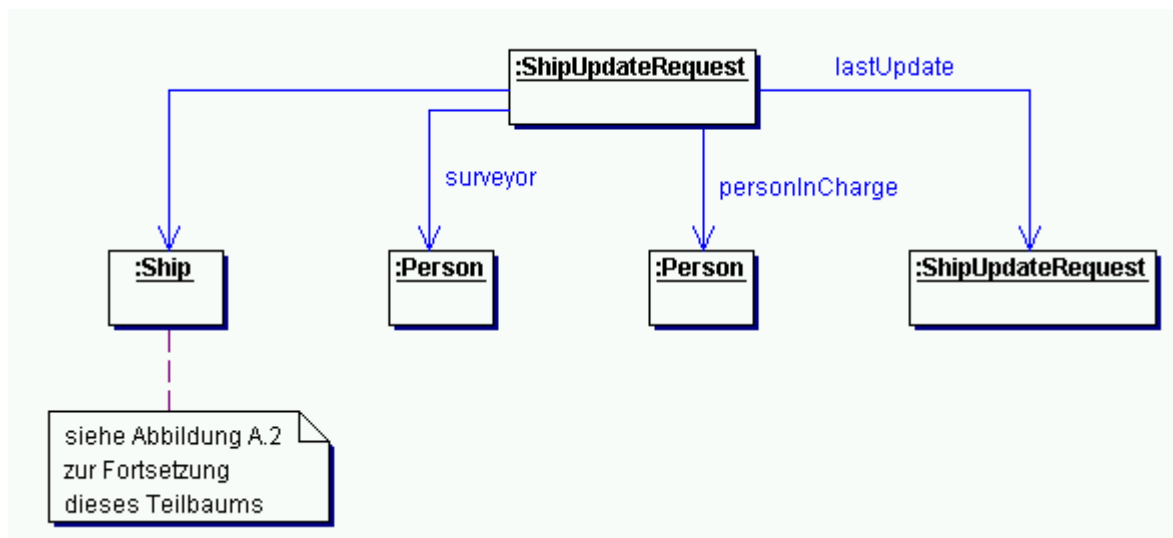


Abbildung A.4: Referenzbaum – ShipUpdateRequests

Anhang B

Klassendiagramme zum Besichtigungsprozess

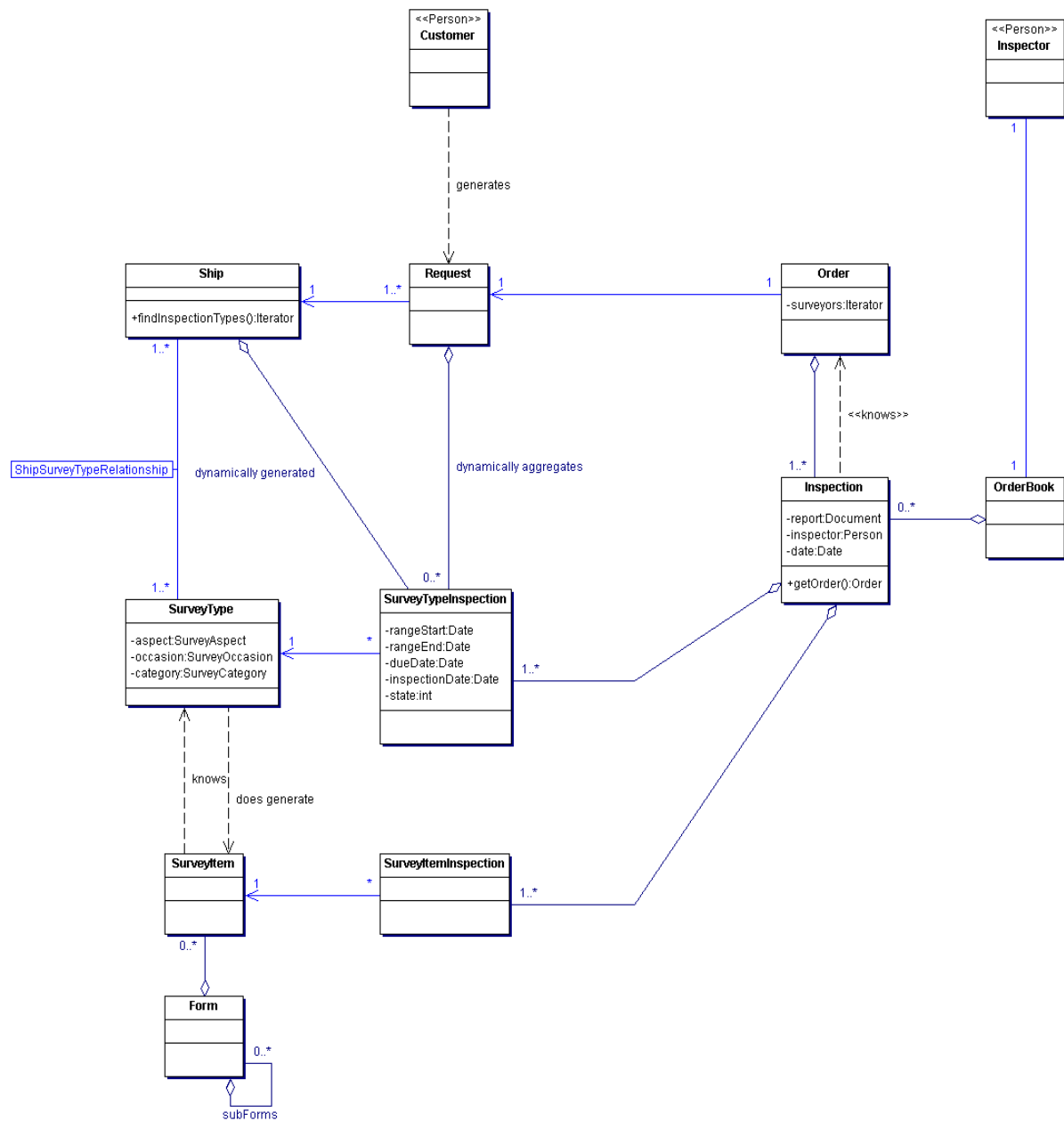


Abbildung B.1: Klassendiagramm Besichtigungen

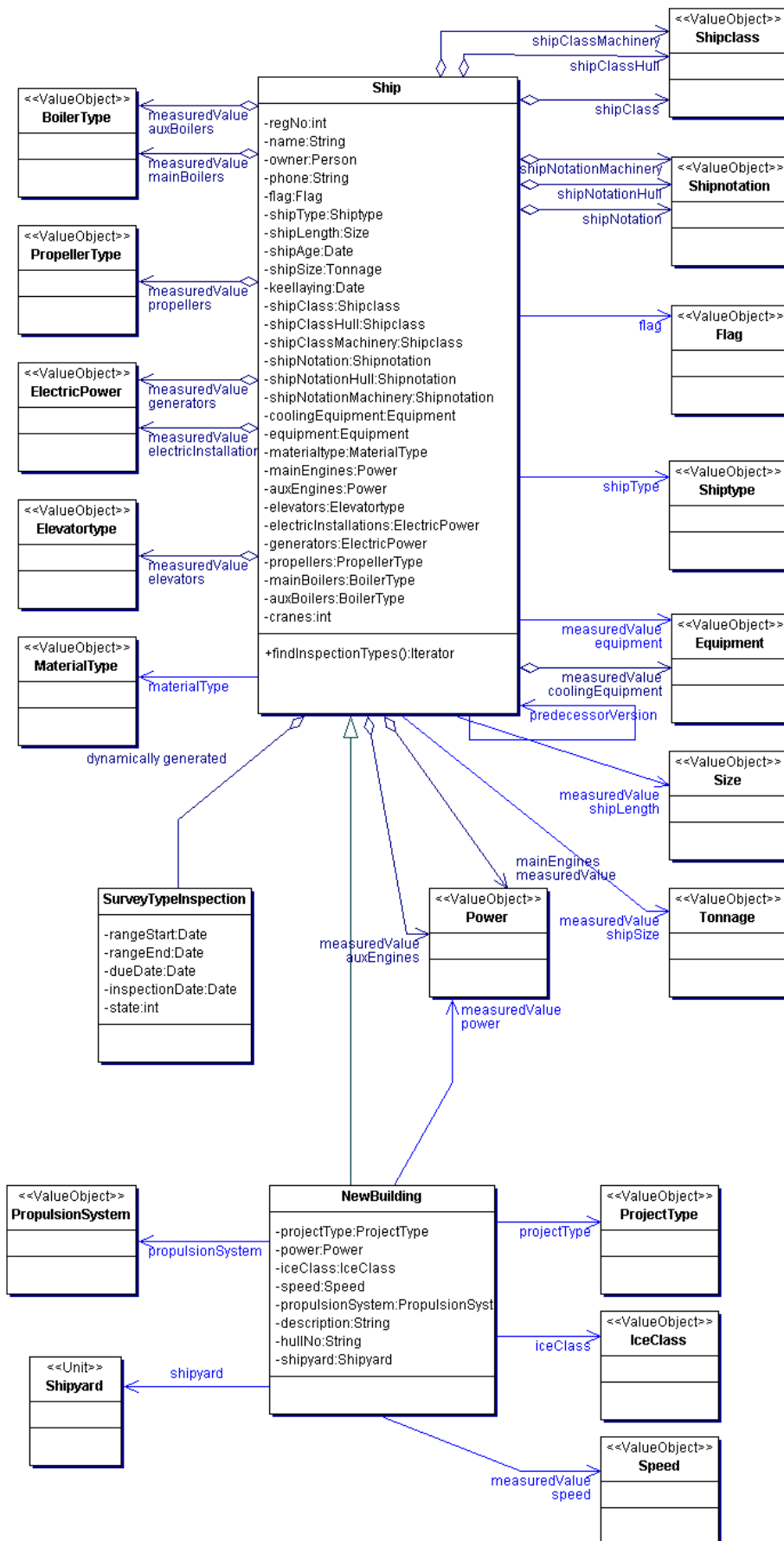


Abbildung B.2: Klassendiagramm Schiffsdaten

Anhang C

Entwicklungsumgebung

Die Portalplattform infoAsset Broker als Grundlage des WIPS IT1.1 Informationsportals sowie das zugehörige Synchronisationsmodell sind vollständig in Java realisiert. In der vorliegenden Studienarbeit wird daher ebenfalls die Java-Technologie als Entwicklungs- und Laufzeitumgebung eingesetzt.

Die Entwicklungsumgebung weist folgende Systemeigenschaften auf (vgl. Tabelle C.1):

Prozessor	→ Intel mobile PIII 850 MHz
Hauptspeicher	→ 256 MB
Betriebssystem	→ Windows 2000

Tabelle C.1: Systemeigenschaften der Entwicklungsumgebung

Die eingesetzten Entwicklungstools für die Realisierung der Synchronisation von Besichtigungsinformationen zwischen WIPS IT1.1 Informationsportalen sind in Tabelle C.2 aufgeführt.

Java-Version	→ JDK 1.4.1
Java IDE	→ IntelliJ IDEA 3.0.1
Erstellung der HTML-Templates	→ Microsoft Frontpage 2002
UML-Analyse und -Entwurf	→ Together ControlCenter 6.0
Versionskontrollsystem	→ Perforce 99.2
Bearbeitung von Graphiken	→ Microsoft Paint 5.1
Erstellung von Dokumenten	→ Microsoft Office 2000

Tabelle C.2: Entwicklungstools

Literaturverzeichnis

- [Dud00] DUDEN, Bibliographisches Institut & F.A. Brockhaus AG, Mannheim 2000
- [BaHa00] BHATTI, N. ; HASSAN, W. Object Serialization and Deserialization Using Xml. 2000.
- [Emnid] TNS EMNID ONLINE, Website, Juli 2003.
<http://www.tns-emnid.com>
- [Ger02] GERKENS, Christoph , Studienarbeit, voraussichtlicher Titel “Entwicklung einer generischen, metadaten-orientierten, Business-Schicht eines Portalssystems”, voraussichtlich 2003
- [gl] GERMANISCHER LLOYD AG , Website, Juli 2003.
<http://www.gl-group.com>
- [iA] INFOASSET AG Hamburg, Website, Juli 2003.
<http://www.infoasset.de>
- [InEx] INTERNET EXPLORER 6, Standard-Web-Browser, Website, Juli 2003.
<http://www.microsoft.com/windows/ie/default.asp>
- [JBR99] JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. The Unified Software Development Process, Addison-Wesley, 1999.
- [JTB00] JANSEN, C., THIESSE, F., BACH, V. Wissensportale aus Systemsicht in: Business Knowledge Management in der Praxis: Prozessorientierte Lösungen zwischen Knowledge Portal und Kompetenzmanagement, Springer Verlag, Berlin 2000
- [KLT00] KOENEMANN, Jürgen; LINDNER, Hans-Günther; THOMAS, Christoph. Unternehmensportale: Von Suchmaschinen zum Wissensmanagement. In: nfd Information –Wissenschaft und Praxis 51 (2000), September, Nr. 6
- [Leh02] LEHEL, Vanda, Synchronisation von Informationen zwischen Portalen, Diplomarbeit, Hamburg im September 2002
- [MaLe02a] MATTHES, Florian; LEHEL, Vanda. Dokument- und Kontaktsynchronisation mit mobilen Datenbanken: Anforderungen und Lösungsansätze aus Sicht von Unternehmensportalen, Workshop Mobile Datenbanken und Informationssysteme, Magdeburg, 2002.

- [MaLe02b] MATTHES, Florian; LEHEL, Vanda. Persönliche Informations- und Wissensportale als dualer Ansatz zu Unternehmensportalen, Informatik 2002.
- [NeNav] Netscape Navigator 7.0, Standard-Web-Browser, Website, Juli 2003.
<http://www.netscape.de/netscapeprodukte/netscape70/netscapenavigator>
- [sts] ARBEITSBEREICH SOFTWARESYSTEME, Technische Universität Hamburg-Harburg, Website, Juli 2003.
<http://www.sts.tu-harburg.de>
- [tuhh] TECHNISCHE UNIVERSITÄT HAMBURG-HARBURG, Website, Juli 2003.
<http://www.tuhh.de>
- [Wegn00] WEGNER, Holm. Der infoAsset Broker: Architektur, Anpassung & Erweiterung, infoAsset AG 2000.
- [Wegn02] WEGNER, Holm. Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement, Dissertation, 2002.
- [wips] FORSCHUNGSPROJEKT WIPS, Website, Juli 2003.
<http://www.wipsnet.de>