
Modellgetriebene homogenisierte Datenhaltung und Synchronisation mit vorhersagbaren Eigenschaften für plattformübergreifende Anwendungen

Förderkennzeichen: 01 S12012

# AP 5

## Entwicklung der initialen Migrationsstrategie auf Basis der Referenzimplementierung und –architektur

**Autor:** B. Klatt (FZI)

**Co-Autoren:** G. Hübsch (CAS)

E. Gailus (B2M)

A. Schwichtenberg (CAS)

**Fertiggestellt am:** 31.03.2013

**Schlagworte:** Migration, Datenhaltung

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

Änderungshistorie

| Version | Änderungen von | Datum | Anmerkung |
|---|---|---|---|
| 01 | Gailus, Hübsch, Schwichten-berg | 11.03.2013 | Initiale Gliederung |
| 02 | Schwichten-berg | 13.3.2013 | Überarbeitung d. Gliederung |
| 03 | Alle | 4.4.2013 | Review zur Finalisierung |
| | | | |
| | | | |
| | | | |
| | | | |

ToDos

| Abschnitt | von | Beschreibung | Pritorität | Todo |
|---|---|---|---|---|
| | | | | |

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

# Inhalt



# Abbildungen



© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

# 1 Einleitung

Im Rahmen des MOHITO Projektes sollen Methoden und Mechanismen zur Generierung von Datenhaltungs-Stacks für verschiedene Plattformen und Endgeräte entwickelt werden. In dem vorliegenden Dokument wird beschrieben, wie eine bestehende Anwendung zu einer zur MOHITO Referenzarchitektur konformen Anwendung mit generierter Datenhaltung migriert werden kann.
Grundlage für die hier beschriebene Migrationsstrategie ist die MOHITO Referenzarchitektur. Um zu verdeutlichen, wie eine bestehende Anwendung zu der Referenzarchitektur migriert wird, werden zunächst mögliche Typen von Bestandsanwendungen grob hinsichtlich der hier entscheidenden Aspekte des Datenmodells und der Datenhaltung klassifiziert und darauf aufbauend die entscheidenden Abweichungen zur Referenzarchitektur beschrieben. Hieraus ergeben sich für die unterschiedlichen Klassen von Bestandsanwendungen spezifische Problemstellungen für eine Migration, die in Kapitel 5 adressiert werden, indem die allgemeinen Prozessschritte der Migration auf die jeweiligen zu migrierenden Anwendungen angepasst werden. Grundlage für die spezifischen Migrationsprozesse sind sowohl der in Kapitel 5 definierte generelle Migrationsprozess, als auch existierende Standard-Entwicklungswerkzeuge, die für einzelne Prozessschritte genutzt werden können.

# 2 Referenzarchitektur

Um eine homogene Datenhaltung modellieren und generieren zu können wird im MOHITO Projekte vorausgesetzt, dass die entsprechende Anwendung konform zu einer definierten Referenzarchitektur ist. Diese Referenzarchitektur wurde aus bestehenden Anwendungen und für verteilte Multiplattform-Anwendungen typischen Entwürfen abgeleitet. Details hierzu sind in dem entsprechenden Dokument zur Referenzarchitektur[1] festgehalten.

Hauptmerkmal der Referenzarchitektur ist die Kapselung der Datenhaltung. Auf jeder Plattform stehen unterschiedliche Fähigkeiten, wie zum Beispiel die Offline-Fähigkeit einer Mobilen App, ein Caching oder die Kompression und Verschlüsselung von Daten zur Verfügung. Unabhängig von den konkret gewählten Fähigkeiten und der konkreten Plattform werden erstere in einer Datenhaltungsschicht gekapselt. Als Folge dessen entwickelt der Programmierer nur gegen eine schlanke Schnittstelle, die auf allen Plattformen gleichartig gestaltet ist. Die konkrete Behandlung der Daten, wie deren Verschlüsselung oder die Logik Daten

[1] s. B. Klatt u. K. Krogmann: *L 3.2: Erweiterte Referenzarchitektur für Multi-Plattformhomogenisierung von Datenhaltungsschichten*, 2013

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten. 

aus dem lokalen Speicher oder von einem entfernten Server zu laden, werden von der Datenhaltungsschicht transparent für den Benutzer übernommen.

Die Konformität zu dieser Referenzarchitektur bedeutet daher alle Datenzugriffs- und Datenhaltungsfunktionalitäten in einer expliziten Datenhaltungsschicht zu kapseln und sie nicht verstreut durch die gesamte Anwendung durchzuführen.

# 3 Klassifizierung bestehender Anwendungen

Da im Rahmen von MOHITO das Ziel der Migration eine Anwendung mit generierter Datenhaltungsschicht ist, haben vor allem jene Charakteristika von Bestandsanwendungen einen entscheidenden Einfluss auf die Migrationsstrategie, die die Speicherung und den Austausch von Daten betreffen. Klarerweise können nicht alle möglichen Konstellationen im Umgang mit den Daten betrachtet werden und man wird auch nicht eine Migrationsstrategie entwickeln können, die auf alle Systeme anwendbar ist. Es werden aber jene zentralen Charakteristika untersucht, die zur Klassifikation der gängigsten Architekturen und der Referenzimplementierungen des MOHITO Projektes angewandt werden können und die zur Definition der Migrationsstrategie für diese Referenzimplementierungen benötig werden. Generell ist außerdem zu erwähnen, dass die Migrationsstrategie nur auf solche Anwendungen abzielt, die objektorientiert entwickelt wurden und eine interne Logik beinhalten. Beispielsweise sind Mobile Clients, die wie ein Browser nur Daten lesen und sie durch ein sogenanntes Rendering geeignet darstellen nicht im Fokus der Migrationsstrategie.

Als wichtigste Faktoren wurden die folgenden identifiziert:

- Datenmodelle in der Systemlandschaft
- Datenhaltung
- Datenaustausch

## 3.1 Datenmodelle in der Systemlandschaft

Bei der Frage, wie sich die Systemlandschaft auf die Migrationsstrategie auswirkt, ist vor allem von Bedeutung, ob die zu migrierende Anwendung dem Client-Server-Modell folgt und wenn ja, wie viele verschiedene Clients mit dem Server kommunizieren. Bei Client-Server-Architekturen muss in einem ersten Migrationsprozessschritt festgestellt werden, ob die Clients und der Server ein einheitliches Datenmodell haben. Wenn dies nicht der Fall ist, müssen die verschiedenen Datenmodelle entweder vereinheitlicht oder aber durch ein Gesamtmodell abgebildet werden. Es kann auch in Betracht gezogen werden, das Datenmodell des Servers unangetastet zu lassen und nur auf den mobilen Clients ein einheitliches Datenmo-

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten. 

dell zu definieren - das zumindest mit dem Datenmodell des Servers kompatibel sein muss. Im Fall einer stand-alone Anwendung gibt es diesen Migrationsprozessschritt nicht.

Ein besonderer Fall tritt ein, wenn die Systemlandschaft der zu migrierenden Anwendung unterschiedlichen Varianten bzw. Versionen von Serverimplementierungen umfasst. Eine solche Situation kann historisch bedingt sein. Sie entsteht beispielsweise durch den schrittweisen Umstieg auf eine neue Serverplattform, bei dem Altsystem und Neusystem solange parallel betrieben und weiterentwickelt werden, bis das Altsystem vollständig abgelöst werden kann. Typischerweise wird dabei versucht, die Datenmodelle von Alt- und Neusystem anzugleichen, um die Auswirkungen auf die Clientseite zu minimieren. Der MOHITO Ansatz ist deshalb für ein solches Szenario besonders interessant. Bis zur vollständigen Angleichung ist das Datenmodell des Neusystems dabei im Normalfall eine Teilmenge des Altsystems. Das Altsystem bildet damit die Grundlage für die Definition des einheitlichen Datenmodells im Rahmen der Migration. Eine Voraussetzung für die Migration ist in diesem Fall die Existenz geeigneter Mittel auf Modell- und Generatorebene, mit denen sich die Unterschiede zwischen Alt- und Neusystem abbilden lassen.

## 3.2 Datenhaltung

Einen wichtigen Einfluss auf die Migrationsstrategie hat die Datenhaltung auf den unterschiedlichen Plattformen der oben beschriebenen Systemlandschaft. Abgesehen von der Art, wie die Daten gehalten werden ist es vor allem relevant, wie auf die Daten zugegriffen wird. Im Rahmen von MOHITO wird von einer Datenhaltung in Datenbanken ausgegangen, Triplestores aus dem Bereich der semantischen Technologien sowie direkte Dateisystem-Zugriffe sind nicht im Scope des Projektes. Hierbei ist zum einen relevant, wo die Datenbankzugriffe realisiert sind und zum anderen, wie sie implementiert wurden. So ist generell eine Anwendung mit gekapselter Datenschicht (Abbildung 1 b) einfacher zu migrieren als eine Anwendung, bei der die Datenzugriffe in der gesamten Anwendung verstreut sind (Abbildung 1 a). Letzteres erfordert einen zusätzlichen Migrationsschritt um zunächst eine Kapselung der Datenbankzugriffe an einer zentralen Stelle der Anwendung zu realisieren.

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

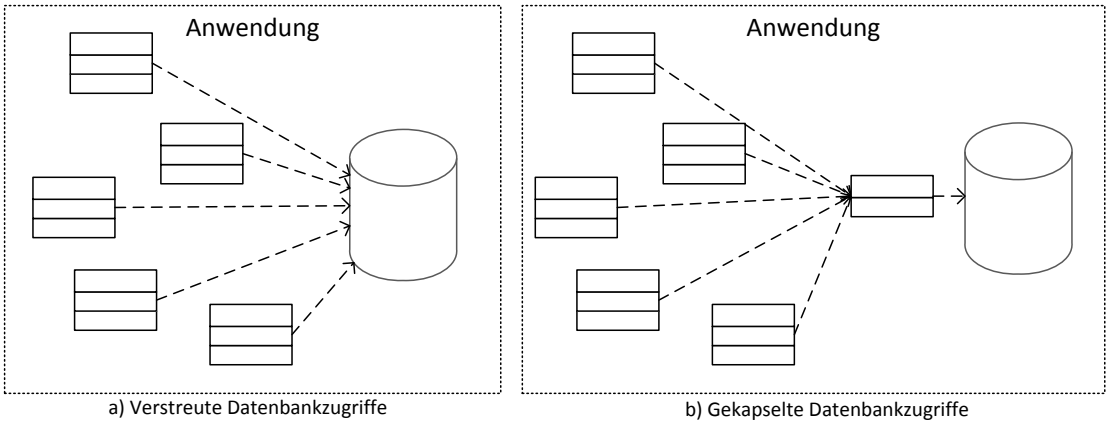


*Abbildung 1 Verstreuter versus gekapselten Datenzugriff*

Bei der Art, wie die Datenbankzugriffe realisiert sind, ist zu unterscheiden zwischen Systemen, die einen bestimmten OR Mapper (wie Hibernate) einsetzen und solchen, die direkt SQL (-ähnliche) Aufrufe absetzen. Prinzipiell sind beide Arten der Zugriffe in der MOHITO Referenzarchitektur vorgesehen. Für die konkrete Realisierungstechnik muss gegebenenfalls ein entsprechender Erweiterungspunkt des MOHITO Frameworks ausgenutzt werden, falls diese Technik noch nicht mitgeliefert wird. Im Rahmen des Projektes kann und soll nur eine Auswahl von Realisierungstechniken unterstützt werden.

## 3.3 Datenaustausch

Neben den Datenmodellen und der Datenhaltung ist bei Client-Server Systemen auch der Austausch der Daten von besonderem Interesse, um eine geeignete Migrationsstrategie definieren zu können. Hierbei ist vor allem entscheidend, wo der Datenaustausch stattfindet und wie der Datenaustausch realisiert ist. Generell sind Anwendungen mit gekapselter Kommunikationsschicht (Abbildung 2 b) einfacher zu migrieren als solche, bei denen die Kommunikation in der gesamten Anwendung verteilt ist (Abbildung 2 a). Letzteres erfordert einen zusätzlichen Migrationsschritt um zunächst eine gekapselte Kommunikationsschicht herzustellen. Genauergesagt soll die Datenkommunikation ebenfalls in der einheitlichen Datenzugriffsschicht, transparent für den Entwickler gekapselt werden.

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

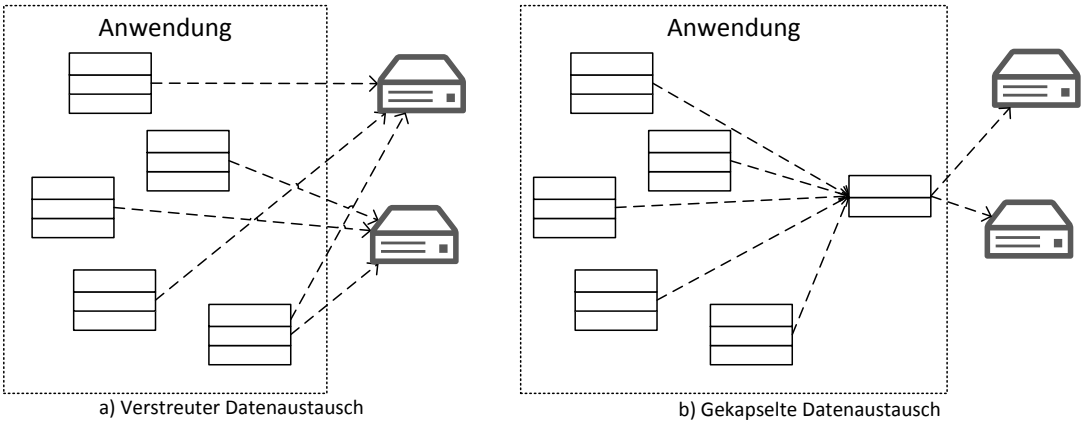


*Abbildung 2 Verstreuter versus gekapselten Datenaustausch*

Bei dem "Wie" des Datenaustausches sind verschiedene Faktoren relevant, wie z.B. ob Daten gecacht werden, wie die Konfliktlösung umgesetzt ist und vor allem auch, ob diese von der Datenhaltungsschicht implementierten Fähigkeiten für den Aufrufer transparent sind. Wenn gewährleistet ist, dass eine einheitliche Schnittstelle zur Verfügung steht, die die Kommunikation und eventuelles Caching kapselt, dann kann diese durch eine MOHITO basierte Schicht ausgetauscht werden. Wenn diese Art der Transparenz nicht vorliegt, dann sind die direkten Interaktionen mit dem Cache und der Kommunikationsinfrastruktur zunächst in einem zusätzlichen Migrationsschritt in der Datenzugriffsschicht zu kapseln.

# 4 State of the Art

## 4.1 Migrationsprozesse

In der Softwareentwicklung allgemein gibt es eine Vielzahl von Prozessmodellen, die von extrem abgesicherten, teuren Prozessen (beispielsweise in der Entwicklung von medizinischen Geräten), bis hin zu flexiblen und leichtgewichtigen Prozessen im Bereich der neuen Medien reichen. Letzteres trifft auch auf die Entwicklung im Bereich multiplattform und mobiler Anwendungen zu, da man durch den hohen Marktdruck zeitnah neue Produkte oder Produktversionen bereitstellen muss. Hiervon sind natürlich auch Migrationsprozesse betroffen.

Ein grundlegendes Element der flexibleren und leichtgewichtigeren Migrationsprozesse ist das iterative Vorgehen und damit auch die Entwicklung von kleineren Inkrementen mit schnelleren, überschaubareren Schritten. Dies ermöglicht vor allem eine schnelle Rückmeldung und ebenso eine schnelle Fehlerkorrektur.

Neben den klassischen Software Entwicklungsmodellen, wie dem Spiralmodell, dem Prototyping oder den agilen Vorgehensweisen, gibt es auch Modelle, die speziell auf Migrationsprozesse abzielen. Hierzu gehört beispielsweise das DUBLO (DUal Business LOgic) Migrati-

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten. 

onsmodell speziell für die Migration von monolithischen Altanwendungen hin zu einer Dreischichtenarchitektur. Hierbei wird die Businesslogik schrittweise migriert und zeitweise auch doppelt vorgehalten um einen weichen und realisierbaren Übergang zwischen alter und neuer Architektur zu ermöglichen.

Zwar ist das DUBLO Modell speziell auf die Geschäftslogik bei der Migration hin zu einer Dreischichtenarchitektur ausgelegt, die generelle Vorgehensweise ist aber auch auf die in MOHITO betrachtete Migration der Datenschicht anwendbar. So ist, wie in Kapitel ??? beschrieben, zunächst eine schrittweise Migration hin zu einer gekapselten und homogenisierten Persistenz-Schicht möglich, die anschließend in einem weiteren Schritt durch eine generierte Persistenz-Schicht ersetzt werden kann.

## 4.2 Erstellung Ecore Datenmodell

Grundlegende Idee des MOHITO Projektes ist die Modellierung eines Datenmodells, aus dem die Persistenz-Schichten für unterschiedliche Plattformen generiert werden können. Für die Modellierung der Datenstruktur selbst wurde das Eclipse Modeling Framework (EMF) und das darin enthaltene Ecore Metamodell als sinnvollste Technologie identifiziert.

Das Eclipse Modeling Framework bietet selbst Unterstützung für einige Datenformate, um aus bestehenden Datenmodellen ein Ecore Datenmodell zu gewinnen. Wie in Abbildung 3 dargestellt erlaubt der Model Import Wizard unter anderem die Auswahl eines UML Models, eines XML Schema oder eine IBM Rose Klassenmodells.

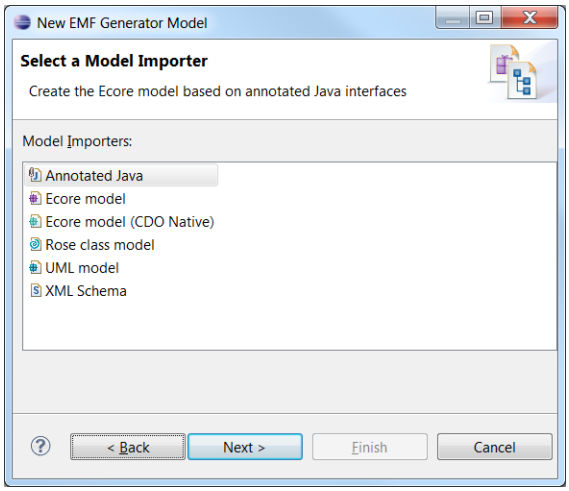


*Abbildung 3 Eclipse Modeling Framework Ecore Import Wizard*

Zusätzlich ist es möglich ein Ecore Modell direkt aus Java Code zu gewinnen. Der Java Code muss hierbei mit entsprechenden Annotationen versehen werden, die der Importer verarbeiten kann.

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

Da sich das Ecore Metamodell inzwischen weit verbreitet hat gibt es auch für viele gängige Modellierungswerkzeuge Export / Migrationswerkzeuge zur Speicherung / Erstellung eines Ecore-Modells. Beispielsweise das Werkzeug Argo2Ecore um aus einem ArgoUML Modell ein Ecore Modell zu gewinnen (http://argo2ecore.sourceforge.net/).

Eine generelle Möglichkeit zur Überführung von bestehenden Datenmodellen in Ecore Modelle ist eine Transformation beispielsweise mittels XSLT. Vor allem OR-Mapper wie Hibernate verfügen über XML basierte Konfigurationsdateien. Da auch Ecore die Möglichkeit bietet mit einem XML Format (xmi) zu arbeiten kann eine entsprechende Transformation geschrieben werden um von dem existierenden XML Format (bspw. eine Hibernate Mapping Datei) in eine Ecore xmi Datei zu transformieren.

# 5 Migrationsstrategie

Um Bestandsanwendungen, wie sie in Kapitel 3 beschrieben wurden in einen zur MOHITO Referenzarchitektur konformen Zustand zu bringen wurde ein mehrstufiger Migrationsprozess entwickelt. Um das identifizierte Spektrum von Bestandsanwendungen abdecken zu können entspricht der Migrationsprozess einem flexiblen Modulkonzept. Je nach IST-Zustand des Systems können gegebenenfalls einzelne Migrationsschritte entfallen, falls das System in einem Punkt bereits zu der Referenzarchitektur konform ist. Der Gesamtprozess beinhaltet die 7 im Folgenden beschriebenen Schritte.

## 5.1 Kapselung Schreib- und Lesezugriffe

Das MOHITO Framework stellt, wie eingangs beschrieben, einen kompletten Datenhaltungs-Stack mit Caching- und Synchronisationsmechanismen zur Verfügung. Dementsprechend muss in einem ersten Migrationsschritt sicher gestellt werden, dass die Datenhaltungsschicht einer entsprechenden Bestandsanwendung, welche migriert werden soll, so gestaltet ist, dass sie durch das MOHITO Framework ersetzt werden kann ist. Daher ist die Kapselung aller Schreib- und Lesezugriffe auf lokal persistierte Daten in einer einheitlichen Schicht eine Grundvoraussetzung der Migration.

Unter Umständen muss hierbei auch das Datenzugriffskonzept angepasst werden. Das MOHITO Framework ist im Zuge einer einfacheren Generierbarkeit auf die Verwendung mehrerer einfacher anstelle einer einzigen komplexen Datenanfrage ausgelegt. Dies kann bedeuten, dass komplexe Datenanfragen auf kleinere atomare Anfragen herunter gebrochen werden müssen um die MOHITO Schnittstelle effizienter zu nutzen. Auch bietet das MOHITO Framework nur synchrone Datenanfragen an. Grund dieser Designentscheidung ist das

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten. 

Framework einfach zu halten und nicht mit zusätzlicher Thread Behandlung und Logik zu überfrachten. Daher muss darauf geachtet werden, dass entsprechende Datenanfragen mit längerer Laufzeit auf Seiten der Clientanwendung in einen eigenen Thread ausgelagert werden.

## 5.2 Kapselung Datenaustausch

Der Teil der zu migrierenden Anwendung, welcher sich mit dem Austausch von Daten beschäftigt, muss ebenfalls austauschbar gestaltet sein, um durch das MOHITO Framework ersetzt werden zu können. Das bedeutet, dass die komplette Datenaustauschlogik in einer Schicht gekapselt werden muss. Damit kann sie in einem weiteren Migrationsschritt ausgetauscht werden. Wichtig ist auch hierbei darauf zu achten, dass ein Datenaustauch über entfernte und daher langsame Verbindungen in einen eigenen Thread auszulagern ist. Auch bei der Art der Anfrage sind der Datenumfang und entsprechende Paging Mechanismen zu berücksichtigen, die je nach Ausgangsszenario in MOHITO anders umzusetzen sind.

Ebenfalls gilt die schon vorher erwähnte Einschränkung in der Ausdrucksmächtigkeit, so dass auch hier eventuell zu komplexe Anfragen in atomare Bestandteile zerlegt werden sollten.

## 5.3 Kapselung Konfliktlösung

Um eine konsistente lokale Datenpersistenz und Cachelogik in Clientanwendungen gewährleisten zu können, müssen Daten, die während einer offline Phase modifiziert wurden, wieder mit den Datensätzen eines Servers, bzw. anderer Clients synchronisiert werden. Hierbei können Konflikte auftreten, die mit geeigneten Konfliktlösungsstrategien beseitigt werden müssen. Solche Strategien sind Bestandteil des MOHITO Frameworks und werden während entsprechender Synchronisationsvorgänge angewandt. Hat eine Bestandsanwendung eigene Konfliktlösungsstrategien implementiert müssen diese durch das MOHITO Framework ersetzt werden. Daher ist es sinnvoll in diesem Migrationsschritt sämtliche Synchronisationsvorgänge in einer Schicht zu kapseln, um sie später einfach ersetzen zu können. Ist eine von der Bestandssoftware genutzte Konfliktlösungsstrategie nicht durch MOHITO adäquat ersetzbar, bietet MOHITO dafür entsprechende Erweiterungspunkte an, um eigene Strategien zu realisieren.

## 5.4 Abgleich Datenmodelle der Teilsysteme

Generell ist zu erwarten, dass die Datenmodelle der einzelnen Teilsysteme zumindest Überschneidungen in den Daten haben, die zwischen ihnen ausgetauscht werden. Dennoch wei-

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten. 

sen separat entwickelte Teilsysteme oftmals auch Abweichungen in ihren Datenmodellen auf. Dies kann entstehen da ein Teilsystem nur mit einer Teilmenge der Daten arbeitet oder aber auch, da unterschiedliche Entwickler / Entwicklerteams verschiedene Datentypen oder -bezeichner verwenden, sofern es hier keine einheitlichen Konventionen gibt.

Zu einer Homogenisierten Datenhaltung gehört natürlich auch ein homogenes Datenmodell. Daher müssen die unterschiedlichen Datenmodelle der Teilsysteme miteinander abgeglichen werden. Idealerweise können alle Teil-Datenmodelle zu einem Datenmodell vereinheitlicht werden in dem einfach ihre Klassen-, Attribut- und Referenznamen abgeglichen werden. Besitzen die Teilsysteme Datenmodelle mit unterschiedlichem Umfang, so sollte ein Gesamtdatenmodell erfasst und wo möglich und sinnvoll konsolidiert werden. Später kann dann aus diesem Gesamtdatenmodell die einzelnen Teilmodelle abgeleitet werden. Wichtig ist aber zunächst Redundanzen und Abweichungen zu identifizieren und soweit möglich abzugleichen.

Können die Datenmodelle nicht vollständig vereinheitlicht werden und bleiben Redundanzen, beispielsweise aufgrund von unterschiedlichen Behandlungen oder Kapazitäten der einzelnen Teilsysteme bestehen, so sollten diese ausführlich beschrieben und begründet werden, da die Synchronisierung der darin enthaltenen Daten später im Betrieb durch den Umgang mit der Datenschnittstelle bzw. durch die generierte Datenhaltung direkt behandelt werden müssen.

Ein besonderes Augenmerk sollte auf Referenzen gelegt werden. Sie sind oft durch die Verwendung der Daten in den einzelnen Teilsystemen, beispielsweise durch eine bestimmte Richtung der Navigation zwischen den Daten, geprägt. Hier ist die Auflösung von Redundanzen oftmals anspruchsvoller aber umso wichtiger.

Gleiches gilt auch für die Datentypen der Attribute der Datenklassen. Sie sind besonders von technischen Gegebenheiten oder plattformspezifische Kapazitäten geprägt. So werden beispielsweise numerische Felder als String an den Client übertragen und dort als solche gespeichert oder in der Länge begrenzte String Felder auf dem mobilen Client statt unbegrenzter String Felder wie auf dem Server verwendet um Speicherplatz zu sparen. Ebenso ist es denkbar, dass bestimmte Konsistenzbedinungen im Datenmodell nur für Teilsysteme gelten. Beispielsweise kann auf dem Server für ein Attribut gefordert werden, dass es beim Persistieren in der Datenbank immer mit einem Wert ungleich *null* belegt sein muss. Für die Zwischenspeicherung im lokalen Cache eines Client gilt diese Einschränkung jedoch nicht zwangsläufig. Dies muss bei der Gestaltung des einheitlichen Datenmodells berücksichtigt und dokumentiert werden um es später entsprechend annotieren zu können (siehe Schritt 5.6).

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten. 

## 5.5 Erstellung Ecore Datenmodell

Der erste produktive Schritt der Migration ist die Abbildung des in Schritt 4 konzipierten homogenisierten Datenmodells als Ecore Datenmodell, auf dem die restlichen Schritte der Migration aufbauen. In Abhängigkeit vom Ausgangszustand sind drei Wege denkbar:

- Manuelles Erstellen des Ecore Datenmodells
- Annotieren des Java-Codes existierender Datenklassen und anschließender Import über den Model Import Wizard
- Implementierung einer XSL Transformation für die Abbildung deklarativ beschriebener Datenmodelle, beispielsweise Hibernate OR Mappings, auf XMI und anschließender Import des XMI

Das manuelle Erstellen des Ecore Datenmodells kann ohne besondere Voraussetzungen erfolgen. Der Vorteil liegt in der unmittelbaren Abbildung des in Schritt 4 konzipierten homogenen Datenmodells in Ecore.

Existieren in der zu migrierenden Anwendung bereits Datenklassen in Java, so bieten sich deren Annotation und ihr anschließender Import über den Model Import Wizard (siehe Abschnitt 4.2) an. Der Vorteil gegenüber der manuellen Erstellung des Modells liegt im geringeren Aufwand, der besseren Überprüfbarkeit der Abdeckung des existierenden Datenmodellcodes, der Reproduzierbarkeit und der Nachvollziehbarkeit der Abbildung zwischen Modellartefakten und dem ursprünglichen Datenmodellcode. Der Umfang der notwendigen Nacharbeiten zur Homogenisierung des so erzeugten Ecore Datenmodells hängt primär vom Grad der Homogenität ab, den das in Code beschriebene Datenmodell bereits aufweist.

Vorbedingung für die Implementierung einer XSL Transformation ist das Vorhandensein eines deklarativ in einem XML Dialekt beschriebenen Datenmodells. Dabei ist abzuwägen, ob der Aufwand für die Erstellung der Transformation und die anschließenden Nacharbeiten am Ecore Modell signifikant geringer ist als das manuelle Erstellen des Ecore Datenmodells. Dies kann beispielsweise bei einer großen Anzahl existierender deklarativer Datenmodellbeschreibungen der Fall sein. Je nach Gestaltung der Transformation lässt sich eine Nachvollziehbarkeit der Abbildung zwischen dem erzeugten XMI und der ursprünglichen Datenmodellrepräsentation erreichen. Der Umfang der notwendigen Nacharbeiten zur Homogenisierung des Ecore Datenmodells hängt auch hier vom Grad der Homogenität ab, den das ursprüngliche Datenmodell bereits aufweist.

## 5.6 Datenmodellanreicherung

Das Ecore Datenmodell muss nach seiner Erstellung um Informationen angereichert werden. Ziel dieser Anreicherung sind die Differenzierung zwischen Teilsystemen (Clients und Server

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten. 

bei verteilten Anwendungen), Festlegungen zur Laufzeitbehandlung der Daten und Konsistenz- sowie Persistenzregeln.

### 5.6.1 Differenzierung zwischen den Teilsystemen Client und Server

In der Praxis sind die Datenmodelle von Client und Server in einer verteilten Anwendung nicht immer vollständig identisch, sondern unterscheiden sich in Details. Beispielsweise müssen oder sollen Clients nicht alle komplexen Typen bzw. alle Attribute eines komplexen Typs kennen, die vom Server verwendet werden (z.B. Systemfelder). Umgekehrt muss der Server nicht alle Felder und Attribute kennen, mit denen die Clients arbeiten. Beispielsweise kann der Client Nutzerpräferenzen lokal persistieren, ohne den Server zu verwenden. Diese Unterschiede müssen im Rahmen der Teilsystemdifferenzierung festgelegt und im Ecore Datenmodell annotiert werden. MOHITO stellt dafür entsprechende Annotationen bereit.

### 5.6.2 Laufzeitbehandlung

Caching und Synchronisation sind Mechanismen, die das MOHITO Framework als optionalen Bestandteil des generierten Datenhaltungs-Stacks bereitstellt. Für das Caching muss der Entwickler dabei festlegen, für welche Teile des Datenmodells dieser Mechanismus angewendet werden soll. Beispielsweise soll sich für vertrauliche Daten festlegen lassen, dass diese nicht im Cache eines mobilen Clients abgelegt werden dürfen, wo sie ein Angreifer durch Entwenden des Geräts auslesen könnte. Außerdem muss der Entwickler definieren, welche Verdrängungsstrategie für Daten im Cache verwendet werden soll.

Synchronisation ist notwendig, um konkurrierende Änderungen von Daten im Cache eines Clients und auf dem Server zu behandeln. Die Festlegung, ob eine Synchronisation stattfinden soll und welche Konfliktlösungsstrategie anzuwenden ist, muss ebenfalls vom Entwickler im Datenmodell festgelegt werden. Bietet MOHITO keine passende Konfliktlösungsstrategie "out of the box" an, müssen in Schritt 7 entsprechende Erweiterungspunkte ausimplementiert werden.

Festlegungen für Caching und Synchronisation werden durch das Hinzufügen entsprechender Annotationen im Ecore Datenmodell getroffen.

### 5.6.3 Konsistenz- und Persistenzregeln

Eine weitere Festlegung, die vom Entwickler im Ecore Datenmodell getroffen werden muss, sind Annotationen zur Konsistenz und Persistenz der definierten Datenstrukturen. Für komplexe Datentypen und einzelne Attribute wird im Ecore Datenmodell festgelegt, ob sie persistiert werden sollen oder nicht. Konsistenzregeln werden über Multiplizitäten an Assoziationen und auf Attributebene (Pflichtattribut, *not null*) festgelegt. Da einige Konsistenzregeln nur für Teilsysteme gelten (siehe Abschnitt 5.4), lassen sie sich im Ecore Datenmodell teilsystemspezifisch definieren.

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

## 5.7 Generierung und Austausch der Datenzugriffsschicht

Der letzte Schritt der Migration einer existierenden Anwendung umfasst die Generierung der zentralen Datenzugriffsschicht aus dem angereicherten Datenmodell und der Austausch der Datenzugriffsschicht in der zu migrierenden Anwendung. Für große und komplexe Anwendungen kann dabei ein iteratives Vorgehen gewählt werden, bei dem die Umstellung auf die generierte Datenhaltungsschicht schrittweise erfolgt, z.B. Teilsystem für Teilsystem. Zusammengefasst sind in jedem Fall die folgenden Aktivitäten durchzuführen:

1. Generierung plattformspezifischer Artefakte
2. Ausimplementieren von Template und Hooks
3. Austausch der Datenzugriffspunkte in der zu migrierenden Anwendung
4. Generierung und Implementierung von Tests

Eine detaillierte Diskussion des letzten Schritts der Migration erfolgt im Ergebnisdokument L 5.2.

# 6 Zusammenfassung

Die Migration von bestehenden Anwendungen hin zur MOHITO Referenzarchitektur ist maßgeblich durch die Lokalität des Datenzugriffs geprägt. Sowohl für Datenaustausch, Synchronisation, Caching als auch den einfachen Lese- und Schreibzugriff gilt, dass eine Migration umso einfacher ist, je besser die Zugriffe bereits gekapselt sind.

Idealerweise sind bereits alle Formen des Datenzugriffs in einer einzigen, zentralen Datenzugriffsschicht gekapselt. Ist dies der Fall gestaltet sich eine Migration recht gradlinig und man muss nur diese Zugriffsschicht selbst anpassen. Ist dies nicht der Fall, so muss zunächst ein entsprechender Zustand hergestellt werden.

Gleiches gilt auch für die Ausgestaltung des Datenmodells bzw. der Datenmodelle im Fall einer verteilten Anwendung. Nutzen alle Teilsysteme das gleiche logische Datenmodell so kann dies direkt in eine von MOHITO unterstütze Modellierung überführt werden. Gegebenenfalls können sogar bestehende Transformationen / Imports genutzt werden um automatisiert das benötigte Ecore Datenmodell zu erhalten. So wird von der Eclipse Plattform zum Beispiel direkt ein Import von UML Datenmodellen in Ecore Datenmodelle angeboten.

Wird kein einheitliches Datenmodell genutzt, so sind die unterschiedlichen Datenmodelle der verschiedenen Teilsysteme zunächst untereinander abzugleichen. Idealerweise zu einem einheitlichen Modell, gegebenenfalls aber auch in ein Gesamtmodell, von dem jedes Teilsystem nur einen Teilbereich nutzt.

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.

Zusammenfassend ergibt sich ein maximaler Migrationsprozess, von dem je nach Zustand der Bestandssoftware einzelne Prozessschritte ausgelassen werden können:

1. Je Teilsystem: Kapselung der Schreib- und Lesezugriffe in zentraler Datenzugriffsschicht.
2. Je Teilsystem: Kapselung von Datenaustausch und -kommunikation in zentraler Datenzugriffsschicht.
3. Je Teilsystem: Kapselung von Konfliktlösung in zentraler Datenzugriffsschicht.
4. Abgleich der Datenmodell zwischen den Teilsystemen
5. Überführung des Datenmodells in ein Ecore Datenmodell
6. Datenmodell anreichern / annotieren
7. Generierung und Austausch der zentralen Datenzugriffsschicht

Der letzte Prozessschritt (7) stellt bereits die technische Migration der Anwendung hin zu einer MOHITO-basierten Implementierung dar und ist nicht mehr Teil der Migration zur Kompatibilität zur MOHITO-Referenzarchitektur.

© 2014 MOHITO Projektkonsortium. Alle Rechte vorbehalten.