Integration von Legacy-Anwendungen durch eine Beobachter-Architektur

Martin Meinhold, Fred Stefan

martin.meinhold@mmsoft.de
stefan@informatik.uni-leipzig.de

Abstract: Zentrales Anliegen bei der Erstellung großer Software-Systeme ist die Integration bereits existierender Anwendungen. Diese Aufgabe ist für Altsysteme, die über keine geeigneten Schnittstellen verfügen, besonders schwierig. In diesem Artikel wird eine Architektur vorgestellt, die eine Integration der bestehenden Systeme auf Datenebene erlaubt. Entgegen den meisten Integrationsansätzen, die auf aktiven Techniken (Push von Nachrichten) beruhen, setzt dieser Beitrag auf einen passiven Ansatz durch einen so genannten Beobachter. Im Gegensatz zu anderen Methoden, wie zum Beispiel SOA, ist dieses Integrationsmuster auch für Bestandssysteme geeignet, die über keine adäquaten Schnittstellen verfügen, da ein Eingriff in die originären Anwendungen hierbei nicht nötig ist.

1 Einführung

Heute befinden sich Unternehmen in einem Umfeld kontinuierlicher Veränderungen, die durch den Wettbewerb auf globalen Märkten, und der damit verbundenen Vernetzung bedingt werden. Um die Potenziale dieser Märkte ausnutzen zu können, ist eine integrierte IT-Infrastruktur nötig, die ausreichende Flexibilität bietet, um den wechselnden Anforderungen zeitnah gerecht zu werden. [ST08, S. 263] Speziell bei großen Anwendern gelten Legacy-Anwendungen als „Altlasten eines Unternehmens“ [Sne02, S. 1], da für eine Integrationsaufgabe oft geeignete Schnittstellen fehlen.

Im Folgenden soll eine Realisierung der, in [Bar01] beschriebenen, Integrationsmethode vorgestellt werden, die einen Beobachteransatz zur prozessorientierten Integration von Anwendungssystemen umsetzt. Anhand eines beispielhaften Anwendungsfalls werden eine Altanwendung und eine moderne J2EE-Anwendung integriert. Dabei kommt als Bestandssystem eine CICS-Cobol-Anwendung zum Einsatz, die auf einem Großrechner der Serie System z aus dem Hause IBM ausgeführt wird.

1.1 System z

Nach zahlreichen Umbenennungen trägt die Großrechnerarchitektur der Firma IBM heute den Namen „System z“. Sie kann auf eine 40jährige Geschichte zurückblicken, stellt

aber dennoch eines der innovativsten Serversysteme in der heutigen IT-Landschaft dar. Besondere Alleinstellungsmerkmale sind dabei die extrem hohe Zuverlässigkeit sowie die hohe Ein- und Ausgabeleistung. Damit sind diese Systeme besonders zur Datenhaltung und Transaktionsverarbeitung geeignet.

Aufgrund dieser Eigenschaften ist weltweit ein sehr großer Teil aller Unternehmensdaten auf Systemen dieser Art gespeichert. Das hat zur Folge, dass Integrationsfragestellungen sehr häufig im Umfeld dieser Großrechner auftreten.

1.2 Legacy-Anwendungen

Als Legacy-Anwendungen werden Softwaresysteme bezeichnet, die unternehmenskritische Funktionen abdecken, jedoch bereits längere Zeit im Einsatz sind. Sie sind in der Regel sehr groß und komplex, so dass eine Ablösung oder Modernisierung sehr aufwändig, risikoreich und kostenintensiv ist. Derartige Systeme arbeiten weitgehend autonom mit wenigen oder keinen Schnittstellen zu anderen Anwendungen. [BS95, S. 1 ff.] Es fällt den Unternehmen zunehmend schwerer, Mitarbeiter zu finden, die das Wissen haben, die aufkommenden Erweiterungs- und Integrationsaufgaben zu lösen. [Pau00, S. 15]

2 Problemstellung

Im vorliegenden Beitrag wird eine konkrete Integrationsfragestellung im Bereich des E-Procurement betrachtet. Derartige Systeme werden häufig zusätzlich zu bestehenden betrieblichen Informationssystemen installiert und müssen demnach in die bestehende IT-Landschaft integriert werden.

Traditionelle Anwendungssysteme, wie sie häufig auf Großrechnern zu finden sind, sind oft bereits über Jahrzehnte gewachsen und bieten nicht immer geeignete Schnittstellen zur Interaktion mit neuen Systemen. Um dennoch unter diesen Bedingungen eine Integration der einzelnen Anwendungen vorzunehmen, wird in diesem Artikel eine Integrationslösung dargelegt, die auf einem regelbasierten Beobachter beruht. Dieser Ansatz ermöglicht eine schnelle und effiziente Integration von traditionellen und neuen Anwendungssystemen, auch wenn diese über keine adäquaten Schnittstellen für diese Anforderung verfügen.

Als traditionelle Anwendung kommt exemplarisch der CICS Catalog Manager zum Einsatz. Diese in Cobol geschriebene Anwendung stellt eines der Standardbeispiele im Großrechnerbereich für Integrationsszenarien dar und simuliert die Lagerverwaltung eines ERP-Systems. Die Bedienung erfolgt über das 3270-Protokoll und die Datenhaltung in diesem konkreten Fall mit Hilfe einer relationalen Datenbank. Weitere vorhandene Schnittstellen, wie zum Beispiel WebServices, werden im folgenden nicht betrachtet, um die Integration mit Hilfe eines Beobachters darzustellen.

Ein moderner Online-Shop soll in diese Altanwendung integriert werden, der Bestellungen aus dem gleichen Katalog ermöglicht. Dieses neue System soll auf der E-Commerce-

Software Intershop Enfinity basieren, die dazu eine eigene Produktdatenbank unterhält. Durch einen Beobachter sollen die beiden Datenbanken synchronisiert werden, wodurch in beiden Systemen der gleiche Produktkatalog verwendet wird und der Online-Shop ebenfalls automatisiert Bestellungen im Catalog Manager vornehmen kann.

3 Integrationskonzepte

Durch den Integrationsprozess werden mehrere verschiedene Subsysteme, wie Programme oder Subprogramme, zu einem größeren, kooperierenden System kombiniert. Die verwendeten Ansätze lassen sich dabei in eng und lose gekoppelt unterscheiden. Dabei ist die ereignisbasierte Integration sicherlich der am weitesten verbreitete Ansatz zur losen Kopplung von Systemen. [BCTW96, S. 378]

Die zu integrierenden Anwendungen verfügen in der Regel über eine mehrschichtige Architektur, wobei jede dieser Schichten einen Ansatzpunkt für die Integration bietet. Ausgehend von einer klassischen 3-Schichten-Architektur, ergeben sich dabei die Methoden zur Präsentations-, Funktions- und Datenintegration.

Unabhängig von den Ebenen der Integration muss die verwendete Integrationstechnik gleichermaßen eine flexible Lösung für Legacy-, Web- und Standardanwendungen renommierter Softwarehäuser sowie andere externe Anwendungen bereitstellen. Die gegenwärtige Situation kann wie folgt zusammengefasst werden: Auf dem Markt existiert eine Reihe namhafter Anbieter, die Integrationslösungen bereitstellen. Serviceorientierte Architekturen (SOA) und Middleware-Lösungen bilden eine zentrale Säule unternehmensweiter und -übergreifender Anwendungsintegration.

3.1 Präsentationsintegration

Damit werden Methoden zur Integration auf der Ebene der Benutzeroberflächen der zu integrierenden Anwendungen bezeichnet. Diese als Screenscraping bezeichneten Methoden können benutzt werden, um Daten aus Altanwendungen automatisiert abzufragen und um Benutzereingaben zu simulieren. Ein Beispiel für diese Vorgehensweise ist das External Programming Interface des CICS Transaction Gateway. [HKS04, S. 175,267 f.]

3.2 Funktionsintegration

Die Funktionsintegration entspricht der Integration auf Applikationsebene und basiert auf der Bereitstellung von Schnittstellen, die von anderen benutzt werden können. Häufig verwendete Methoden für die Verwendung derartiger Schnittstellen sind RPC oder RMI. [Sne02, S. 66] Spezialfälle solcher Schnittstellen sind WebServices oder das External Call Interface des CICS Transaction Gateway. [HKS04, S. 175,274 ff.]

3.3 Datenintegration

Die Integration mehrerer Anwendungen kann über eine gemeinsame Datenbasis erfolgen. Zu diesem Zweck müssen aus einer bestehenden Datenbasis die zugrunde liegenden Modelle zurückgewonnen werden. Dieser von [Bla98] beschriebene Prozess gliedert sich in die Phasen „Implementation Recovery“, „Design Recovery“ und „Analysis Recovery“. Das Ergebnis der einzelnen Phasen ist jeweils ein Datenbasismodell, wobei das letzte Modell die zu integrierenden Geschäftsobjekte enthält.

3.4 Serviceorientierte Architekturen

SOA sind seit den neunziger Jahren bekannt und lösen durch das Zusammenspiel von Diensten (Services[1]) die starren Prozesse der monolithischen Alt-Systeme ab. Dies soll unter anderem auch den Betrieb, die Wartung und die Pflege der Legacy-Anwendungen anpassungsfähiger und günstiger machen.

Grundkonzept einer SOA ist die Kapselung von Anwendungsfunktionalitäten in Services, die über eine wohldefinierte und standardisierte Schnittstelle angesprochen und genutzt werden können. Erst die so genannte Orchestrierung der einzelnen Services[2] macht in einer SOA eine Anwendung aus.

Letztlich erschwert aber nicht nur eine große Anzahl unterschiedlicher Anwendungen und deren Heterogenität die Integration, sondern auch die nicht vorhandene Flexibilität der Legacy-Systeme. Um SOA-konforme Schnittstellen für Altanwendungen bereitzustellen, sind häufig aufwändige Änderungen an den bestehenden Systemen nötig. [Mas07, S. 4] Um den SOA-Erwartungen nach größerer Agilität und Flexibilität bei geringeren Kosten gerecht zu werden, ist daher meist eine umfangreiche, kostenintensive Restrukturierung der Anwendungslandschaft nötig, die mit einem hohen Planungsaufwand verbunden ist.

Einen möglichen Lösungsansatz bietet hier das CICS Service Flow Feature. Es bietet einen alternativen Ansatz zur Modernisierung von Legacy CICS Anwendungen, bei dem bereits vorhandene Anwendungen weiterverwendet und in eine neue Architektur integriert werden können. Jedoch sind hierfür umfangreiche Definitionen in CICS sowie die Service Flow Runtime nötig. [SHS09, S. 58 f.]

3.5 Middleware-Lösungen

Die Software für eine Integration von Anwendungen bzw. die strukturierte Kopplung von Systemen kann unter dem Begriff „Enterprise Application Integration (EAI)-Software“ zusammengefasst werden. [Lin00, S. 1 ff.] EAI verknüpft verschiedene Systeme (sowohl intern als auch extern) über eine einheitliche Integrationsplattform miteinander. Allerdings

[1] oft als Web Service realisiert
[2] also das Zusammenspiel mit anderen Services

benötigen die einzubindenden Anwendungssysteme eine Schnittstelle zum EAI-System. Erst dann können EAI-Systeme Daten von der Quelle zum Ziel transportieren bzw. konvertieren. Eine EAI-Lösung ist meist sehr aufwändig und kostspielig, da die Middleware-Systeme eng an eine bestimmte Basistechnologie anknüpfen. [Lie07, S. 132] Als Bestandteil von Middleware-Lösungen ist häufig ein Enterprise Service Bus (ESB) zu finden, der eine Kombination aus EAI, SOA und Message Oriented Middleware (MOM) darstellt.

Letztendlich bieten jedoch Services und rein technologische Plattformen nach [SKJ06, S. 184] keinen Mehrwert für Unternehmen, sondern vielmehr:

- umfangreiche Wiederverwendung von Code
- leichte Integration unterschiedlicher Produkte
- geringer Wartungsaufwand von Code
- einfachere Zusammenarbeit mit Geschäftspartnern

3.6 Ereignisbasierte Integration

Die zuvor beschriebenen traditionellen Lösungen erfordern eine stark vorausschauende Vorgehensweise sowohl bei der IT-Organisation, als auch in den einzelnen Fachbereichen. Aufgrund der vielen involvierten Ressorts sind diese Lösungen äußerst bürokratisch, planungs- und dokumentenorientiert. In ihrer etablierten Form sind diese Verfahren zu schwergewichtig.

Eine in allen drei Integrationsebenen anwendbare leichtgewichtige Methode ist die ereignisbasierte Integration. Sie ermöglicht die Kooperation mehrerer Anwendungen durch die Auslösung und die Reaktion auf Ereignisse. In [BCTW96, S. 384 ff.] wird ein generisches Referenzmodell vorgestellt. Darin übertragen die Teilnehmer Nachrichten als Reaktion auf ein eingetretenes Ereignis. Ein Teilnehmer kann gleichzeitig Sender und Empfänger von Nachrichten sein. Bevor jedoch der Versand von Nachrichten möglich wird, müssen sowohl der Sender, als auch der Empfänger bei einem so genannten „Registrar“ registriert sein. Ein Router verwendet dann diese Informationen um die Nachrichten zu transportieren.

4 Beobachter-basierter Integrationsansatz

Im Folgenden wird die Entwicklung eines regelbasierten Beobachters betrachtet, der die prozessorientierte Integration auf eine leichtgewichtige Art und Weise umsetzt.

Dieser Ansatz realisiert das in [Lut00, S. 71 f.] vorgestellte Integration Mediator Pattern. Bei diesem Integrationsschema liegt die gesamte nötige Logik außerhalb der zu integrierenden Anwendungen. Damit kann eine Entkopplung von den zu integrierenden Anwendungen erreicht werden, was ein hohes Maß der geforderten Agilität und Flexibilität ermöglicht.

Analog zu der in [Bar01] vorgestellten Architektur, ist die Aufgabe des Beobachters die Erkennung von Zustandsänderungen der Geschäftsobjekte der beteiligten Anwendungen.

Häufig können diese Zustandsänderungen nicht direkt ermittelt werden. Sie müssen somit indirekt beobachtet und anhand der gegebenen Regeln erkannt werden. Diese indirekte Erkennung von Zustandsänderungen kann zum Beispiel über Objekte in der Datenhaltungsschicht der betroffenen Anwendungen erfolgen. Diese Datenbasisobjekte lassen sich deutlich einfacher beobachten und lassen in Verbindung mit den Regeln Rückschlüsse auf Zustandsänderungen der Geschäftsobjekte zu.

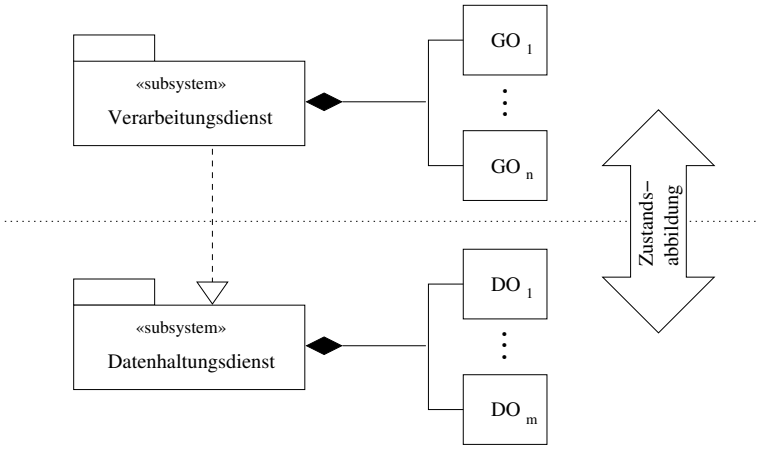

Abbildung 1: Zustandsabbildung zwischen Geschäfts- und Datenobjekten, nach [Bar01, S. 38]

In Abbildung 1 ist die Abbildung zwischen Geschäfts- und Datenhaltungsobjekten dargestellt. Ein Geschäftsobjekt kann auf ein oder mehrere Datenhaltungsobjekte abgebildet werden. Es ist jedoch ebenfalls möglich, dass mehrere Geschäftsobjekte auf ein Datenhaltungsobjekt abgebildet werden. [Bar01, S. 39 ff.] Somit kann eine Regel als Zuordnung von Datenhaltungs- zu Geschäftsobjekten betrachtet werden. Um mehrere Geschäftsobjekte auf ein Datenhaltungsobjekt abbilden zu können, müssen zusätzliche Bedingungen angegeben werden können, wann eine bestimmte Abbildung benutzt werden soll.

Diese Änderungen in der Datenhaltungsschicht werden als Sub-Zustandsänderungen bezeichnet. Die Änderung des zugehörigen Geschäftsobjektes hingegen wird als Zustandsänderung bezeichnet. Die Änderung eines Geschäftsobjektes kann mehrere Sub-Zustandsänderungen zur Folge haben. Der Beobachter beobachtet also indirekt einen Zustand (das Geschäftsobjekt), der aus mehreren Sub-Zuständen (Datenobjekten) bestehen kann.

Hat der Beobachter anhand der definierten Regeln eine Zustandsänderung eines Geschäftsobjektes in einer der integrierten Anwendungen erkannt, löst er eine entsprechende Aktion in den anderen Anwendungen aus. Damit werden die Datenbestände aller Anwendungen synchronisiert.

4.1 Szenario

Anhand des nachstehenden Szenarios soll die beispielhafte Umsetzung einer Beobachter-basierten Integrationsarchitektur demonstriert und bewertet werden. Im Folgenden werden zunächst die beiden Anwendungen sowie die betrachteten Anwendungfälle vorgestellt. Danach wird der Entwurf und die Realisierung des Beobachters erläutert.

4.1.1 Vorstellung des CICS Catalog Managers

Die im Folgenden betrachtete Anwendung „CICS Catalog Manager“ ist eine beispielhafte Cobol-Anwendung, die von IBM entwickelt wurde, um die Interoperabilität von CICS-Anwendungen mit anderen Applikationen zu zeigen. Sie verwaltet einen einfachen Produktkatalog und stellt eine einfache Bestellfunktion zur Verfügung. Außerdem kann der Anwender Lagerbestände zu Produkten aus dem Katalog abfragen. [RAB+06, S. 83 f.]

Die Bedienung der Anwendung ist recht gewöhnungsbedürftig und erfordert einen 3270-Emulator oder ein entsprechendes Terminal. Eine Bildschirmansicht des CICS Catalog Managers ist in Abbildung 2 dargestellt.

```
CICS EXAMPLE CATALOG APPLICATION  - Inquire Catalog

Select a single item to order with /, then press ENTER

Item    Description                                  Cost   Order
-----------------------------------------------------------------
0010    Ball Pens Black 24pk                         2.90     _
0020    Ball Pens Blue 24pk                          2.90     _
0030    Ball Pens Red 24pk                           2.90     _
0040    Ball Pens Green 24pk                         2.90     _
0050    Pencil with eraser 12pk                      1.78     _
0060    Highlighters Assorted 5pk                    3.89     _
0070    Laser Paper 28-lb 108 Bright 500/ream        7.44     _
0080    Laser Paper 28-lb 108 Bright 2500/case      33.54     _
0090    Blue Laser Paper 20lb 500/ream               5.35     _
0100    Green Laser Paper 20lb 500/ream              5.35     _
0110    IBM Network Printer 24 - Toner cart        169.56     _
0120    Standard Diary: Week to view 8 1/4x5 3/4    25.99     _
0130    Wall Planner: Eraseable 36x24               18.85     _
0140    70 Sheet Hard Back wire bound notepad        5.89     _
0150    Sticky Notes 3x3 Assorted Colors 5pk         5.35     _

F3=EXIT    F7=BACK   F8=FORWARD   F12=CANCEL
```

Abbildung 2: Katalogansicht im CICS Catalog Manager

Der CICS Catalog Manager bietet die Möglichkeit die Datahandler, also die Datenhaltungsschicht, auszutauschen. In einer Konfiguration ist dabei immer genau ein Datahandler aktiv. Um in diesem beispielhaften Anwendungsfall die Integrationsaufgabe möglichst übersichtlich zu halten, wurde der Datahandler `DFH0XDDS` in Abbildung 3 gewählt, der die Kataloginformationen in der Tabelle `PRODUCTS` der DB2-Datenbank

CMDB ablegt. Damit wird die Verwendung von Triggern zur Beobachtung der Datenbasisobjekte möglich. Die Verwendung des VSAM-Datahandlers DFH0XVDS hätte die wesentlich komplexere Entwicklung des CICS User Exits[3] XFCAREQ zur Folge gehabt.

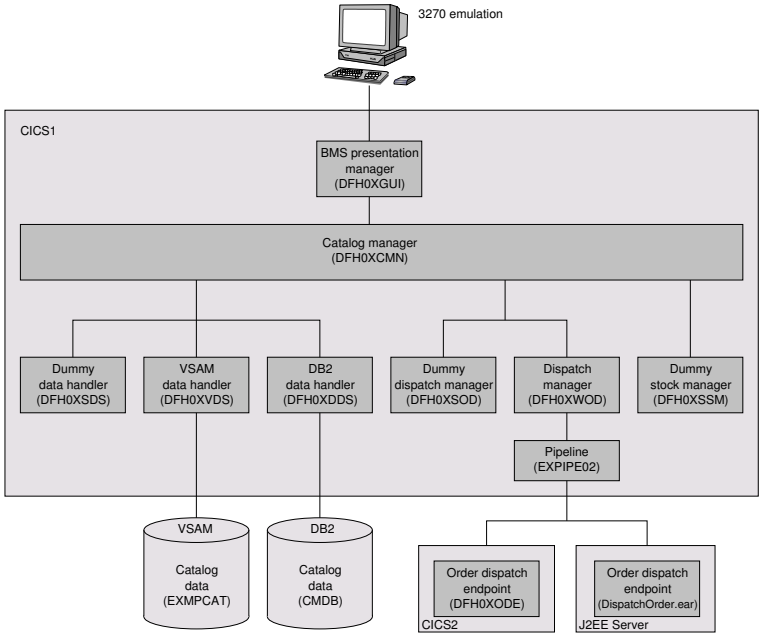

Abbildung 3: Überblick CICS Catalog Manager

4.1.2 Vorstellung Intershop Enfinity Suite

Die E-Commerce-Software Intershop Enfinity Suite ist eine Standardsoftware für den Absatz über das Internet. Neben Standardfunktionen, die auch komplexeren Unternehmensanforderungen gerecht werden, besteht die Möglichkeit, das System je nach Kundenwunsch an eigene Bedürfnisse anzupassen.

Eine Installation der Intershop Enfinity Suite besteht aus zahlreichen Soft- und Hardwarekomponenten, wie Loadbalancing, Applikationsservern, Fileservern und Datenbanken. [HH08, S. 205 ff.] Über sechs Module lassen sich unterschiedliche Geschäftskanäle in der E-Commerce-Anwendung zusammenführen und steuern. Für den hier dargestellten B2C (Business-to-Consumer) Bereich wird der Consumer Channel verwendet. Hiermit wird der direkte Verkauf an Endkunden simuliert.

[3] Ein CICS User Exit ist ein Punkt in einem CICS-Modul, an dem CICS die Kontrolle an ein externes Programm übergibt. Dieses Programm kann vom Administrator angepasst werden. Nach der Ausführung des Programms erhält CICS die Kontrolle zurück. [IBM08, S. 3]

4.2 Anwendungsfälle

Zur Integration der beiden Applikationen werden im Folgenden zwei Anwendungsfälle betrachtet. Einerseits die „Aktualisierung der Produktdaten“ des Catalog Managers in DB2. Dabei werden die aktualisierten Produktdaten durch einen Trigger an den Adapter des Beobachters übergeben. Andererseits wird der „Export von Bestellungen“ aus dem Online-Shop betrachtet. In diesem Zusammenhang wird eine XML-Datei generiert, die alle nötigen Informationen über geänderte Datenbestände enthält.

Aktualisierung der Produktdaten Dieser Anwendungsfall beinhaltet das Hinzufügen, Ändern und Löschen von Produkten im CICS Catalog Manager bzw. der zu Grunde liegenden Datenbank. Die jeweiligen Änderungen in der Produkttabelle werden anhand von entsprechenden Triggern erkannt.

Wurde ein neues Produkt angelegt oder ein bestehendes Produkt verändert, wird ein Action-Flag und alle in der zugehörigen Datenbanktabelle enthaltenen Informationen an den Adapter des Beobachters übergeben. Wurde ein Produkt gelöscht, wird nur das Action-Flag sowie der Produktidentifikator übergeben.

Export von Bestellungen Dieser Anwendungsfall dient der Übergabe der Bestellungen aus dem Online-Shop an den CICS Catalog Manager. Dieser ändert daraufhin die Bestände im Katalog und teilt diese Änderungen dem Online-Shop durch den ersten Anwendungsfall mit. Die übertragenen Daten bestehen aus den Identifikatoren der einzelnen Produkte und den zugehörigen Mengen.

4.3 Ereignisse und Zustände

Die Bestimmung der zu beobachtenden Datenbasisobjekte und der resultierenden Ereignisse erfolgt nach den Methoden von [Bla98] und [HHH+97]. Dabei werden die Schritte „Implementation Recovery“, „Design Recovery“ und „Analysis Recovery“ der Reihe nach durchlaufen. Das Ergebnis der dritten Phase ist ein konzeptionelles Modell, das die zu beobachtenden Geschäftsobjekte beschreibt [Bar01, S. 21].

Auf der Seite des CICS Catalog Managers sind die zu beobachtenden Geschäftsobjekte dieses Anwendungsfalls die Produktinformationen des Katalogs. Diese beinhalten einen Produktidentifikator, eine Beschreibung, eine Kostenstelle, einen Bestand sowie eine Information über die nachbestellte Menge.

Auf der Seite des Intershop-Systems sind die zu beobachtenden Geschäftsobjekte die erzeugten Bestellungen, die zu jeder Position den Produktidentifikator sowie die bestellte Menge beinhalten.

Jede Änderung des Zustandes eines dieser Geschäftsobjekte wird als Ereignis bezeichnet. Änderungen der Subzustände, also der Datenbasisobjekte, werden analog dazu als Sub-Ereignisse bezeichnet.

Daraus ergeben sich die zu betrachtenden Ereignisse mit ihren Sub-Ereignissen. Die Trigger „feuern“ im CICS Catalog Manager für jede Änderung und jeden Datensatz einmal. Damit wird für jeden geänderten, erstellten oder gelöschten Datensatz genau ein Sub-Ereignis erzeugt. In diesem konkreten Anwendungsfall besteht das zugehörige Ereignis aus genau einem Sub-Ereignis. Für jede Bestellung im Intershop können jedoch potentiell mehrere Sub-Ereignisse erzeugt werden, wenn eine Bestellung mehrere Posten enthält. Das zugehörige Ereignis besteht jedoch ebenfalls nur aus einem Sub-Ereignis. Damit ist es möglich, dass bei einer Bestellung mehrere Ereignisse auftreten.

4.4 Systementwurf und Realisierung

In Abbildung 4 ist das beschriebene Integrationsszenario schematisch dargestellt. In der Anwendungsschicht sind die beiden zu integrierenden Anwendungen dargestellt, die jeweils auf eine darunter liegende Datenbank zugreifen. In der obersten Schicht ist der in Java implementierte Beobachter mit den beiden Adaptern, dem Regelwerk und dem Regelprozessor dargestellt. An jedem Übergang zwischen den drei Schichten kann potentiell eine Systemgrenze liegen.

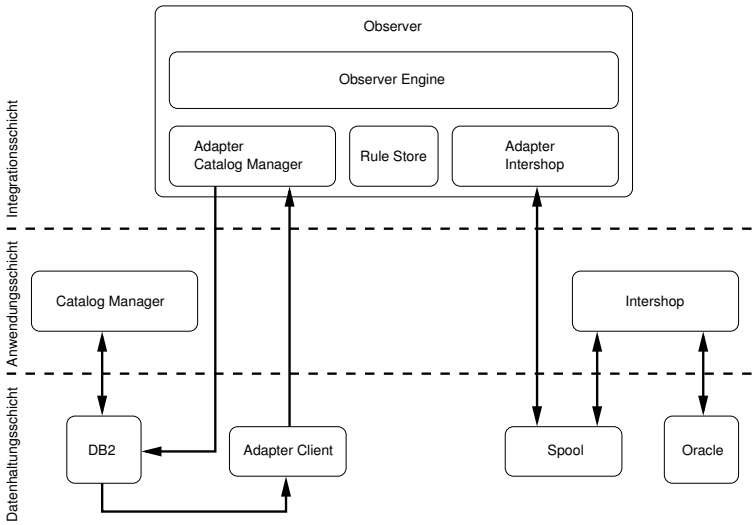

Abbildung 4: Architektur der Integrationsinfrastruktur

Die zu beobachtenden Geschäftsobjekte sind die Katalogeinträge in den beiden Anwendungen. In der Intershop Enfinity Suite können diese Informationen über die Import- und Export-Schnittstellen in XML-Dateien abgelegt werden. In diesem Szenario wird davon ausgegangen, dass der Catalog Manager keine derartigen Schnittstellen besitzt. Die einzige Möglichkeit ist hier die Beobachtung der Datenhaltungsschicht.

4.4.1 Adapter Catalog Manager

Die Abbildung der Geschäfts- auf Datenbasisobjekte ist im CICS Catalog Manager durch die verschiedenen Datahandler (Vgl. Abbildung 3) austauschbar. Die Verwendung des DB2-Datahandlers ermöglicht die einfache Erkennung von Änderungen durch Trigger.

Der Adapter des Catalog Managers gliedert sich in zwei Teile. Einerseits der Adapter-Server, der ein Modul des Beobachters ist, und andererseits der Adapter-Client, der als gespeicherte Prozedur in DB2 implementiert ist.

In der Datenbank wurden drei Trigger (für `INSERT`, `UPDATE` und `DELETE`) implementiert, die die geänderten Daten an die gespeicherte Prozedur übergeben. Diese Prozedur baut eine Netzwerkverbindung auf und übermittelt das Sub-Ereignis mit Hilfe eines Java-Object-Stream an den Adapter-Server. Dieser übergibt es danach an den Regelprozessor.

Wurde ein neues Ereignis durch den Regelprozessor erkannt, wird es an den Adapter-Server übergeben, der daraufhin eine JDBC-Sitzung zur Datenbank erstellt und die nötigen Änderungen durchführt.

4.4.2 Adapter Intershop Enfinity

Die verfügbaren Schnittstellen ermöglichen an dieser Stelle einen sehr einfachen Adapter im Vergleich zum CICS Catalog Manager. Es wird ein Spool-Verzeichnis verwendet, welches vom Adapter und von Intershop Enfinity ständig beobachtet wird. Darin wird für jedes Ereignis eine neue Datei abgelegt, die einen Namenspräfix und einen Zeitstempel enthält. Daran kann sowohl die Reihenfolge der Ereignisse, als auch das Zielsystem bestimmt werden.

Stellt der Adapter fest, dass eine neue Datei im Verzeichnis enthalten ist, die für ihn bestimmt ist, dann wird sie eingelesen und umbenannt. Die darin enthaltenen Sub-Ereignisse werden an den Regelprozessor weitergegeben.

4.4.3 Regelwerk

Das betrachtete Szenario ist so gewählt, dass die benötigten Regeln direkt angegeben werden können. Sie könnten ebenfalls mit den Methoden der induktiven logischen Programmierung gelernt werden [BSZS99, SB99].

Für jeden Anwendungsfall, der in Kapitel 4.2 spezifiziert wurde, wird eine Regel benötigt.

Für jedes Sub-Ereignis und Ereignis wurde eine Java-Klasse erstellt, die die entsprechenden Attribute des Ereignisses enthält. Die zugehörigen Regeln werden in einer Konfigurationsdatei angegeben und haben die in Auflistung 1 dargestellte Form.

Der Regel wird ein Name sowie eine implementierende Klasse und ein Zieladapter zugewiesen. Danach können mehrere Elemente vom Typ `<subevent>` oder `<assertion>` angegeben werden. Sub-Ereignisse haben ebenfalls einen Namen und eine Implementierung. Es werden verschiedene Bedingungen unterstützt, die über das Attribut `condition` angegeben werden können. Die möglichen Bedingungstypen sind „equal“, „not equal“,

```
<event name="Eventname" class="Eventklasse"
       target="Zieladapter">
  <subevent name="Subeventname" class="Subeventklasse"/>
  <assertion condition="Bedingungstyp">
    <left  type="Datentyp" value="Wert"/>
    <right subevent="Subeventname"
       attribute="Attributname"/>
  </assertion>
</event>
```

Auflistung 1: Beispielhafte Regel

„less than“, „less or equal“, „greater than“ und „greater or equal“. Dazu wird für jede Bedingung der linke und der rechte Operand angegeben. Dabei ist es möglich entweder ein Objekt zu instantiieren oder ein Attribut eines Ereignisses anzugeben. Ein Integer-Objekt mit Wert 7 kann durch `<left type="java.lang.Integer" value="7"/>` erstellt werden.

Sollten im Intershop-System zusätzliche oder angepasste Informationen benötigt werden, wie zum Beispiel Bilder, Versandkosten oder Währungsumrechnungen, so können diese durch einfaches Ergänzen von Regeln in das Shopsystem eingebunden werden.

4.4.4 Regelprozessor

Dieser Teil des Beobachters führt die Zuordnung der Sub-Ereignisse zu Ereignissen durch. Dazu werden die ankommenden Sub-Ereignisse zunächst in eine Warteschlange eingereiht. Danach wird systematisch versucht, aus diesen Informationen anhand der Regeln und unter Berücksichtigung der Bedingungen ein Ereignis zu erzeugen.

Zunächst werden alle Permutationen aus Sub-Ereignissen aufgestellt, die ein Ereignis ergeben könnten. Danach wird über diese Permutationen iteriert und alle Bedingungen evaluiert. Konnte aus einer Permutation ein Ereignis erstellt werden, wird es an den Zieladapter übergeben. Danach werden die zugehörigen Sub-Ereignisse aus der Warteschlange entfernt und alle Permutationen, die mindestens eines dieser Sub-Ereignisse enthielten, gelöscht. Sollten dann noch Permutationen übrig sein, wird mit deren Überprüfung fortgesetzt.

4.5 Einsatz und Bewertung der Beobachter-Architektur in einem Beispielprojekt

Der Funktionsumfang des Beobachters umfasst die Übernahme und Aktualisierung von Produktdaten aus einem Bestandssystem in einen neuen Online-Shop und die Übernahme der Bestellungen aus der neuen Anwendung in das Bestandssystem. Die Zielsetzung konnte damit vollständig erreicht werden.

Um die selbe Funktionalität zu erreichen, wäre es ebenfalls möglich gewesen, die Altanwendung entsprechend anzupassen, was jedoch einen deutlich höheren Aufwand zur Folge hätte. Es müsste unter Umständen ein Reverse-Engineering großer Teile der Anwendung durchgeführt werden und es müsste Quelltext in der Sprache der Altanwendung erstellt werden, was heute immer weniger Programmierer beherrschen.

Weiterhin unterscheidet sich dieser Beobachter von anderen Ansätzen darin, dass er zum Einen sehr leichtgewichtig ist und zum Anderen der benötigte Zeitbedarf für die Integration vergleichsweise gering ist.

5 Zusammenfassung

Mithilfe des in dem Beitrag beschriebenen Beobachters kann eine spezielle Klasse von Anwendungssystemen integriert werden, ohne dass hierfür spezielle Schnittstellen existieren müssen. Einzige Voraussetzung hierfür ist, dass die verwendete Datenhaltungsschicht geeignete User Exits, Trigger oder etwas Vergleichbares zur Verfügung stellt.

Die vorgestellte Beobachter-basierte Integrationslösung ist besonders für Situationen geeignet, in denen traditionelle Methoden, wie SOA oder EAI, aufgrund der Aufgabenstellung einfach nicht gerechtfertigt sind. Als eine besondere Klasse von Problemen hat sich die Integration von Legacy-Anwendungen auf Großrechnersystemen herausgebildet. Dabei geht es darum, die Vorzüge der bewährten Anwendungen zu erhalten und durch neue Funktionalitäten zu verbessern. Es wird ein Integrationsszenario aufgezeigt, bei dem eine traditionelle Altanwendung auf Cobol-und Mainframebasis um E-Procurement-Funktionalitäten erweitert wird. Die dargelegte Integrationslösung zeichnet sich dadurch aus, dass eine typische Cobol-Legacy-Anwendung ohne Eingriff in die Anwendung mit einem äußerst geringem Aufwand um neue Funktionalitäten erweitert werden kann.

Die Funktionslogik der Altanwendung wird durch den direkten Zugriff des Beobachters auf die Datenhaltungsschicht jenes Bestandssystems umgangen. Somit empfiehlt sich dieser Integrationsansatz nur für einzelne, überschaubare Anwendungsfälle, deren Priorität auf einer einfachen und schnellen Integrationslösung liegt. Weiterhin kann durch den zusätzlichen Netzwerkverkehr, der durch die Adapter entsteht, die Reaktionszeit der Anwendungen zunehmen. Diese Zunahme tritt genau dann ein, wenn die Anwendungen auf die erfolgreiche Ausführung der Übertragung der Sub-Ereignisse warten, bevor ein Bearbeitungsvorgang abgeschlossen wird. Werden diese Informationen nicht blockierend im Hintergrund über das Netzwerk übertragen, können diese Latenzen versteckt werden.

6 Ausblick

Für eine produktive Verwendung dieses Integrationsansatzes sind weitere Anforderungen zu beachten, die üblicherweise bei Integrationsfragen auftreten. So können zum Beispiel durch einen Absturz des Beobachters Informationen über Sub-Ereignisse verloren gehen.

Das könnte vermieden werden, indem die Sub-Ereignisse sofort nach Erhalt persistent gespeichert werden. Damit könnten diese Informationen nach einem Neustart des Beobachters erneut eingelesen und der ursprüngliche Zustand rekonstruiert werden.

In der jetzigen Architektur sind keine Methoden zur Konsistenzerhaltung der Datenbestände der beteiligten Anwendungen enthalten. Wünschenswert wäre also die Realisierung eines Zwei-Phasen-Commit-Protokolls [SS83]. Diese Anforderung ist jedoch keinesfalls so einfach umzusetzen, wie die im letzten Absatz beschriebene. Da eine Zustandsänderung eines Geschäftsobjektes Änderungen in mehreren Datenhaltungsobjekten hervorrufen kann, müssen für eine erfolgreiche Erkennung des geänderten Zustandes des Geschäftsobjektes zunächst alle zugehörigen Sub-Ereignisse aufgesammelt werden. Erst danach könnte ein Zwei-Phasen-Commit-Protokoll ansetzen, um die Konsistenz zwischen den integrierten Anwendungen zu gewährleisten. In CICS wäre dies zum Beispiel durch einen so genannten „Task-related User Exit“ [IBM08, S. 267 ff.] möglich, der vom Syncpoint Manager aufgerufen wird. Damit könnte immer vor einem Commit überprüft werden, ob alle anderen integrierten Anwendungen die Daten ebenfalls korrekt übernommen haben.

Außerdem ist die Authentifizierung zwischen Server und Client des Adapters für den CICS Catalog Manager nicht geregelt. Um den sicherheitstechnischen Anforderungen von Unternehmen gerecht zu werden, könnte entweder eine dedizierte Netzwerkverbindung oder ein VPN-Tunnel benutzt werden. Es ist ebenfalls möglich, den Adapter so anzupassen, dass verschlüsselte Verbindungen benutzt werden können.

Literatur

[Bar01] Thomas Barnekow. *Ein regelbasierter Beobachter zur prozessorientierten Integration betrieblicher Informationssysteme*. Dissertation, Universität Stuttgart (Fakultät für Konstruktions- und Fertigungstechnik), 2001.

[BCTW96] Daniel J. Barrett, Lori A. Clarke, Peri L. Tarr und Alexander E. Wise. A framework for event-based software integration. *ACM Transactions on Software Engineering and Methodology*, 5(4):378–421, 1996.

[Bla98] Michael Blaha. On Reverse Engineering of Vendor Databases. *Proceedings of the Fifth Working Conference on Reverse Engineering*, Seiten 183 – 190, 1998.

[BS95] Michael L. Brodie und Michael Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach*. Morgan Kaufmann Publishers, 1995.

[BSZS99] Thomas Barnekow, Steffen Staab, Jürgen Ziegler und Rudi Studer. An architecture for recovering business events bottom-up. In Hans-Jörg Bullinger und Jürgen Ziegler, Hrsg., *Human-Computer Interaction: Communication, Cooperation, and Application Design, Proceedings of HCI International '99 (the 8th International Conference on Human-Computer Interaction), Munich, Germany, August 22-26, 1999, Volume 2*, Seiten 614–618. Lawrence Erlbaum, 1999.

[HH08] Peter Hänsgen und Benjamin Holtz. Werkzeug zur grafischen Modellierung von Enfinity-Installationen. In Klaus-Peter Fähnrich, Stefan Kühne und Maik Thränert,

Hrsg., *Model-Driven Integration Engineering - Modellierung, Validierung und Transformation zur Integration betrieblicher Anwendungssysteme*, Jgg. XI of *Leipziger Beiträge zur Informatik*, Seiten 205–215. Eigenverlag Leipziger Informatik-Verbund (LIV), September 2008.

[HHH+97] J-L. Hainaut, J-M. Hick, J. Henrard, D. Roland und V. Englebert. Knowledge Transfer in Database Reverse Engineering - A Supporting Case Study. *Reverse Engineering, Working Conference on*, 0:194–204, 1997.

[HKS04] P. Herrmann, U. Kebschull und W.G. Spruth. *Einführung in z/OS und OS/390*. Oldenbourg Wissenschaftsverlag GmbH, 2004.

[IBM08] International Business Machines Corporation. *CICS Trancaction Server for z/OS Customization Guide*, 2008. Version 3 Release 2, IBM Publication No. SC34-6814-01.

[Lie07] Daniel Liebhart. *SOA goes Real Service-orientierte Architekturen erfolgreich planen und einführen*. Hanser Fachbuchverlag, München, 2007.

[Lin00] David S. Linthicum. *Enterprise Application Integration*. Addison-Wesley Longman Ltd., Essex, UK, UK, 2000.

[Lut00] Jeffrey C Lutz. EAI architecture patterns. *eAI Journal*, Seiten 64–73, März 2000.

[Mas07] Dieter Masak. *SOA?: Serviceorientierung in Business und Software (Xpert.press)*. Springer-Verlag GmbH, Berlin, 2007.

[Pau00] Linda Dailey Paulson. Making Legacy Assets Work in an Internet World. *IT Professional*, 2(3):10–15, 2000.

[RAB+06] Chris Rayns, Isabel Arnold, Chris Backhouse, Leigh Compton, David Evans, Jim Hollingsworth und William Yates. *Application Development for CICS Web Services*. IBM International Technical Support Organization, Poughkeepsie, NY, USA, 1. Auflage, 2006.

[SB99] Steffen Staab und Thomas Barnekow. Towards Learning Notification Triggers. In *Proceedings of the international Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business (IJCAI-99). Stockholm, Sweden*, 1999.

[SHS09] Fred Stefan, Paul Herrmann und Wilhelm G. Spruth. Was geschrieben ist, ist geschrieben - Legacy CICS-Anwendungen im neuen Gewand. *it - Information Technology*, 51(1):57–61, 2009.

[SKJ06] August-Wilhelm Scheer, Helmut Kruppke und Wolfram Jost. *Agilität durch ARIS-Geschäftsprozessmanagement*. Springer, Berlin [u.a.], 2006.

[Sne02] Harry M. Sneed. Integration statt Migration. *HMD Praxis der Wirtschaftsinformatik*, 225:3–4, 2002.

[SS83] D. Skeen und M. Stonebraker. A Formal Model of Crash Recovery in a Distributed System. *IEEE Transactions on Software Engineering*, 9(3):219–228, 1983.

[ST08] Fred Stefan und Maik Thränert. Minimal-invasive Integration von Anwendungssystemen. In Klaus-Peter Fähnrich, Stefan Kühne und Maik Thränert, Hrsg., *Model-Driven Integration Engineering - Modellierung, Validierung und Transformation zur Integration betrieblicher Anwendungssysteme*, Jgg. XI of *Leipziger Beiträge zur Informatik*, Seiten 263–275. Eigenverlag Leipziger Informatik-Verbund (LIV), September 2008.