2009

WIEDERGEWINNUNG VON WEB SERVICES AUS VORHANDENEN ALTSYSTEMEN

Harry M. Sneed
Universität Regensburg

WIEDERGEWINNUNG VON WEB SERVICES AUS VORHANDENEN ALTSYSTEMEN

Harry M. Sneed[1]

Kurzfassung
Dieser Beitrag beschreibt einen Werkzeugkasten für die Ableitung von Web Services aus bestehenden COBOL Programmen auf dem IBM Mainframe. Das erste Werkzeug - ***COBAudit*** *– dient dazu potentielle Web Services zu identifizieren. Das zweite Werkzeug* ***-COBStrip*** *– markiert den Programmpfad zum gewünschten Ergebnis und kommentiert den Rest des Codes aus. Das dritte Werkzeug* - ***COBWrap*** *– kapselt den ausgewählten Codebaustein und transformiert ihn in eine ausführbare Komponente. Das vierte Werkzeug –* ***COBLink*** *– verbindet die gekapselte Komponente mit dem Web über eine WSDL Schnittstelle. Die Werkzeuge werden zurzeit an einem großen Bausparsystem mit 12 Millionen COBOL Anweisungen erprobt. Der Beitrag beschreibt den Stand dieser Arbeit relativ zum allgemeinen State of the Art auf dem Gebiet.*

1. Migration zur Service-orientierten Architektur

Konventionelle Anwendungssysteme sind bekanntlich sehr starr und es dauert viel zu lange sie zu verändern. Es kann Monate dauern bis ein Änderungsantrag genehmigt, umgesetzt und getestet wird. Größere Änderungen wie die Baseler Vereinbarungen für das Kreditgeschäft können sogar Jahre dauern. Alte Software ist schwer zu verstehen und noch schwieriger zu verändern ohne unerwünschte Seiteneffekte. Sie hat sich über die Zeit erhärtet. Sie aufzubrechen um neue Funktionen einzuschieben verursacht einen unverhältnismäßig hohen Aufwand. Anderseits verlangt unsere schnelllebige Zeit ständige Anpassungen mit schnellen Reaktionszeiten. Service-orientierte Architekturen versprechen eine größere Flexibilität der Geschäftsprozesse und eine rasche Umsetzung von Änderungen zu der Software. Die einzelnen Softwarebausteine werden als allgemein gültige Web Services bereitgestellt. Es obliegt den Prozessverantwortlichen, sie beliebig in ihren Prozessen einzubauen. Er kann auch die Web Services ohne weiteres austauschen können. Heute benutzt er Web Service A, morgen Web Service B. Zwischen den bereits verwendeten Web Services kann er neue Web Services jederzeit einschieben. Seine Geschäftsprozesse kann er selber gestalten. Er trägt die Verantwortung für den jeweiligen Gesamtprozess. Die IT muss nur garantieren, dass die Web Services das leisten, was in deren Spezifikation versprochen ist.

Die wichtigste Voraussetzung für eine service-orientierte Architektur ist das Vorhandensein ausreichender Web Services. Wenn wir davon ausgehen, dass ein Web Service eine feine Granularität haben sollte, um in möglichst viele Geschäftsprozesse hineinzupassen, wird ein Web

[1] Universität Regensburg, Germany, Institut für Wirtschaftsinformatik Lehrstuhl-I Prof. Pernul

Service in etwa einem Modul mit maximal 500 Anweisungen entsprechen. Demnach wird ein einziger betriebswirtschaftlicher Prozess wie die Genehmigung eines Kredits eine Menge von 10 bis 50 Web Services benötigen. Eine ganze Applikation, wie z.B. das Kreditwesen, wird mehrere hundert Web Services in Anspruch nehmen. Die Frage stellt sich, woher sollen die vielen Web Services herkommen. Dieser Autor hat schon in einer früheren Veröffentlichung darauf hingewiesen:

- man kann sie kaufen
- man kann sie mieten
- man kann sie übernehmen
- man kann sie entwickeln
- man kann sie wiederaufbereiten.

Das Kaufen heißt, sie als Standard Softwaresteine von einem Anbieter zu kaufen und bei sich auf der eigenen Anlage zu installieren. Das Mieten heißt sie auf einer fremden Plattform nach Bedarf zu verwenden – „Software on Demand", bzw. „Software as a Service". Das Übernehmen heißt, sie von der Open Source Gemeinde zu übernehmen, anzupassen und in der eigenen Umgebung einzusetzen. Entwickeln heißt, sie selber nach den eigenen Anforderungen zu spezifizieren, zu implementieren und auszutesten. Die Wiederaufbereitung bedeutet letztlich, dass sie aus den vorhandenen Altsystemen entnommen und entsprechend aufbereitet werden. Alle fünf Alternativen haben Vor- und Nachteile, auf die hier nicht weiter eingegangen wird. Sie wurden in früheren Veröffentlichungen ausführlich diskutiert. In diesem Beitrag geht es ausschließlich um das fünfte Alternativ – die Gewinnung aus der Altsoftware. Hier wird ein Weg aufgezeichnet wie man Codebausteine in vier Schritten aus dem bestehenden Source Code herausnimmt und in ausführbare Web Services verwandelt. Ob dies das beste Alternative zur Gewinnung von Web Services ist, bleibt dahingestellt. Auf jeden Fall ist es schnell und relativ preiswert.

2. Zum Stand der Altsysteme

Gegenwärtig haben viele Anwenderbetriebe der ersten Stunde mit den Systemen, die sie damals mühevoll entwickelt haben, zu kämpfen. Sie kommen davon nicht ohne weiters los. Im Laufe der Jahre sind diese Systeme gemäß den Gesetzen der Softwareevolution immer größer und komplexer geworden [1]. Angesichts ihrer aktuellen Größe und Komplexität, ist es ist kaum möglich sie zu ersetzen. Zum einen, wäre es viel zu teuer, zum anderen weis keiner genau was in ihnen steckt. Das Wissen der Vergangenheit liegt in dem Code begraben. Auch eine Migration ist fragwürdig, ob es sich lohnt die Altsoftware auf eine neue Hardware Plattform oder in eine andere Sprache zu versetzen. Die Komplexität wird dadurch nicht weniger, die Qualität nicht besser. Das migrierte System bleibt ein schwarzer Kasten. Haben Systeme einmal die Million Codezeilen Grenze überschritten, ist es kaum noch möglich sie ohne eine erhebliche Investition zu erneuern. Der Anwender sitzt in der so genannten Legacy Falle.

Um, z.B. ein Bausparsystem mit 12 Millionen Codezeilen und 8 Millionen Anweisungen neu zu entwickeln würde es nach der COCOMO-II Methode 1.175 Personenjahre, nach der Function-Point Methode 1.668 Personenjahre und nach der Data-Point Methode 1.233 Personenjahre kosten. Allein um das System mit einer jährlichen Änderungsrate von 7% zu erhalten, werden circa 80 Personen pro Jahr benötigt. Sogar eine Sanierung würde in der Größenordnung von 60 Personenjahren kosten. Gleichzeitig gehen die Einnahmen durch das Bauspargeschäft zurück, d.h. der Anwender ist gezwungen zu sparen. Obwohl seit Jahren darüber geredet wird, gibt es in Deutschland immer noch keine Standardsoftware für die Bausparbranche. Ergo gibt es wenig Hoffnung, weder auf eine Neuentwicklung, noch auf eine Sanierung, noch auf die Ablösung durch ein Standardprodukt. Was bleibt ist die Migration in eine Service-orientierte Architektur. Wenn es

gelingt die Web Services automatisch aus der Altsoftware zu gewinnen, könnte diese Alternativ unter der 40 Personenjahre Grenze bleiben, wobei der Hauptanteil davon für den Test wäre.

Anderseits nimmt der Wettbewerb ständig zu. Der Markt der Finanzdienstleistungen wird immer härter umkämpft. Um bestehen zu können, muss ein Anbieter sowohl den Vertretern im Außendienst als auch den Kunden Zugriff auf die zentralen Datenbestände über das Internet gewähren. Vertreter und Kunden sollten in der Lage sein, verschiedene Sparmodelle durchzurechnen. Dazu brauchen sie einzelne, überschaubare Web Services. Außerdem wird verlangt die Geschäftsprozesse flexibler zu gestalten und im Hinblick auf Personalreduktion zu optimieren. Weder das eine noch das andere Ziel kann ohne den erleichterten Zugang zur Funktionalität der bestehenden Applikationssoftware erreicht werden.

Es gebe daher zwei gute Gründe für eine service-orientierte Architektur, die Funktionen innerhalb der Altsysteme zugänglich macht. Der erste Grund ist der Internetanschluss für die Online Transaktionen auf dem Mainframe. Der zweite Grund ist die Möglichkeit bisherige Transaktionen als Schritte in die neu gestalteten Geschäftsprozesse einzubauen. Anderseits, darf der Übergang zur neuen Architektur die Kontinuität der laufenden Produktion nicht gefährden. Die Altsysteme müssen ununterbrochen ihren Dienst weiter leisten. Die Anforderung, eine neue Service-Architektur mit der bestehenden Fachlichkeit zu schaffen und die Einschränkungen dies mit möglichst wenig Kosten und ohne den laufenden Betrieb zu stören, stellt die Reengineering Technologie vor einer großen Herausforderung [2].

3. Virtuelle Migration

Der Begriff „Migration“ wird normalerweise benutzt um die Versetzung eines Systems in eine andere Umgebung, bzw. die Übertragung der Daten in eine andere Art Datenbank oder die Übersetzung der Programme in eine andere Programmiersprache, zu bezeichnen [3]. Eigentlich trifft diese Definition hier nicht ganz zu, denn die Software bleibt in derselben Sprache und in derselben Umgebung in der sie bisher war. Nichts ändert sich außer den Schnittstellen zu der Software. Folglich handelt es sich weniger um eine herkömmliche Migration als viel mehr um eine Integration, bzw. um eine virtuelle Migration [4]. Die bestehenden IMS Transaktionen auf dem Mainframe werden in eine neue service-orientierte Architektur integriert. Statt mit den Endanwendern direkt über feste MFS Masken auf 3270 Terminals zu kommunizieren, kommunizieren sie mit den Web Clienten der Geschäftsprozesse indirekt über WSDL Schnittstellen. Sie werden zu Knoten in einem unternehmensweiten Netz von Web Services [5].

4. Stand der Forschung auf dem Gebiet der Web Service Mining

Dieses Projekt ist nicht das Erste dieser Art. Forschung auf dem Gebiet der Web Service Mining findet schon seit Jahren statt, allerdings ohne all zu viele praktische Ergebnisse. Mit „Mining“ ist die Gewinnung der Services aus der bestehenden Codemasse gemeint. Die Vision einer Service-orientierten Architektur ist für die überlasteten Anwender mit ihren veralteten Legacysysteme so etwas wie eine Fata Morgana. Sie verspricht Ihnen die Befreiung von den Fesseln ihrer Software-technischen Vergangenheit. Kommerzielle Produktanbieter und Service-Dienstleister nähren dieser Hoffnung mit neuen verheißungsvollen Produkten und Services [6]. Dennoch um in den Genuss jener Produkte und Services zu kommen, müssen die Anwender erst den funktionalen Inhalt bereitstellen. Es liegt nahe, diese Funktionalität aus den bestehenden Anwendungen zu entnehmen. Das haben Forscher der Software-Reengineering Gemeinde bald erkannt und einige haben schon 2004 begonnen, Methoden und Werkzeuge zu entwickeln mit deren Hilfe die Funktionalität alter Programme sich wiederaufbereiten lässt. Die ersten Forschungsprojekte für diesen Zweck

begannen gleichzeitig an den Universitäten Benevento und Bari in Italien, Durham in G.B. sowie Victoria und Waterloo in Kanada im Jahre 2005. Das amerikanische Software Engineering Institute hat auch damit begonnen, diese Möglichkeit der Softwarewiedergewinnung zu erforschen. Die Ergebnisse der Forschung sind auf den WCRE, der ICSM und der CSMR Konferenzen vorgestellt worden.

Das RCOST Institut in Benevento hatte schon 2002 ein Projekt zur Kapselung alter COBOL Systeme in einer Webanwendung für die Gemendeverwaltung der Campania durchgeführt [7]. Forscher an der Universität Victoria haben sich spezialisiert auf die Migration von Altsysteme in Web Anwendungen. Diese Forschung wurde inzwischen auf die Migration zu einer service-orientierten Architektur ausgedehnt [8]. Die SEI hat eine Forschungsgemeinde mit eigner Website zu diesem Zweck gegründet [9]. Dieser Autor hat über seine ersten Forschungsergebnisse im Jahre 2006 auf der CSMR in Bari berichtet [10]. Es mangelt also nicht an Forschung auf diesem Gebiet. Was fehlt, ist eine echte industrielle Anwendung. Das vorliegende Projekt ist ein erster Schritt in diese Richtung.

5. Ziele des Pilotprojektes

Wie mit allen Migrations- und Integrationsprojekten, die Neugrund betreten, wurde auch hier ein Pilotprojekt vorgesehen. Pilotprojekte werden durchgeführt um neue Technologien zu testen und daraus Erfahrungswerte zu sammeln, mit deren Hilfe das IT Management Entscheidungen treffen kann. Zum einen, ist noch offen, ob das Unternehmen überhaupt in Richtung Service-orientierte Architektur gehen sollte. Diese Entscheidung hängt davon, ob genügend billige Web Services zur Verfügung stehen. In dem fraglichen Unternehmen kommt weder eine Neuentwicklung noch Standardsoftware in Frage. Eine Neuentwicklung wäre zu teuer und Standardsoftware ist nicht vorhanden. Open Source Services decken nur einen kleinen Teil der erforderlichen Services ab. Also bleibt nur das Alternativ, die Services aus der bestehenden Software zu übernehmen. Die Frage ist nur wie. Der Anwender könnte die bestehenden Online-Programme so kapseln wie sie sind. Man brauchte nur die IMS Masken durch Webseiten zu ersetzen, aber das ergibt keine Web Services sondern nur gekapselte Transaktionen. Die einzige Möglichkeit echte Web Services aus der bestehenden Software zu bekommen ist sie herauszuschneiden. Dies ist jedoch ein gewagtes Unterfangen und musste erprobt werden. Daher das Pilotprojekt. Die Ziele des Pilotprojektes sind wie folgt:

1) die Größe, Komplexität und Qualität der IMS/COBOL Programme zu messen, um den Umfang der Kapselungsarbeit abschätzen zu können.
2) die potentielle Wiederverwendung der Programme als Web Services zu messen.
3) die Möglichkeit zu prüfen, Codescheiben herauszuschneiden, um als Web Services wieder zu verwenden.
4) den Beweis zu liefern, dass es möglich ist ausgewählte Programmscheiben zu kapseln ohne die ganze IMS-TP Transaktion zu kapseln.
5) den Zugriff auf die gekapselten Web Services über Websphere zu testen.

5.1. Messung der Kandidaten-Programme

Das erste Ziel war die Messung der vorhandenen Software bezüglich ihrer Größe, Komplexität und Qualität. Alle 7,297 COBOL Programme wurden mit dem Werkzeug COBAudit gemessen. Die Größenmessungen waren unter anderem die Anzahl Codezeilen, die Anzahl Anweisungen, die Anzahl Datengruppen, die Anzahl Datenelemente, die Anzahl Dateien, die Anzahl Datenbanktabellen, die Anzahl Datenbankzugriffe, die Anzahl Entscheidungsknoten, die Anzahl

Unterprogramme und die Anzahl Unterprogrammaufrufe. Insgesamt wurden 56 Code-Quantitäten gezählt.

Die Programmkomplexität wurde im Bezug auf acht Komplexitätsmetriken gemessen, die Datenkomplexität von Chapin, die Datenflusskomplexität von Elshof, die Datenzugriffskomplexität von Card, die Schnittstellenkomplexität von Henry, die Steuerflusskomplexität von McCabe, die Entscheidungskomplexität von McClure, die Verzweigungskomplexität von Sneed, die Sprachkomplexität von Halstead. Diese Komplexitätsmaße sind in früheren Beiträgen des Autors veröffentlicht worden [11]. Es genügt hier darauf hinzuweisen, dass die Programmkomplexität als Koeffizient auf der rationalen Skala von 0 bis 1 ausgedruckt wird, mit 0,5 als mittlere Komplexität. Eine Komplexitätsnote über 0,6 zeigt, dass ein Programm übermäßig komplex ist. Eine Note unter 0,4 lässt schließen, dass die Komplexität kein Problem ist. 0 bis 0,4 steht für niedrige Komplexität, 0,4 bis 0,6 mittlere Komplexität, 0,6 bis 0,8 hohe Komplexität und über 0,8 bedeutet der Code sei nicht veränderbar.

Die Programmqualität wurde im Bezug auf die Modularität, die Portabilität, die Wiederverwendbarkeit, die Konvertibilität, die Flexibilität, die Testbarkeit, die Konformität und die Wartbarkeit gemessen. Auch diese Metriken sind in der Literatur mehrfach beschrieben worden. Von besonderer Bedeutung für die Wiederverwendung als Web Service sind die Qualitätsmerkmale Modularität, Wiederverwendbarkeit und Flexibilität. Modularität versteht sich als hohe Kohäsion und niedrige Kopplung. Je weniger Verbindungen ein Codebaustein zu anderen Bausteinen hat, desto leichter ist es sie herauszuschneiden. Die Wiederverwendung ist eine Frage der Unabhängigkeit von den anderen Bausteinen sowie von der technischen Umgebung. Wiederverwendbare Bausteine haben keine IO Operationen, keine Datenbankzugriffe und keine Aufrufe fremder Bausteine. Flexibilität ist schließlich die Unabhängigkeit von fachlichen Daten. Der Code sollte von eingebauten Daten – hard coded Data – frei sein damit sie mit beliebigen Daten verwendbar ist [12]

Die Messung des Codes sollte zeigen welche Programme sich als Web Services gut eignen und welche sich dafür gar nicht eignen. Sie soll auch darüber Aufschluß geben, was es kosten würde die Programme zu kapseln. Die Kapselungskosten werden vor allem von der Kapselungsfähigkeit bestimmt. Diese sei wiederum eine Frage der Modularität, der Wiederverwendbarkeit und der Flexibilität.

5.2. Bewertung der Wiederverwendbarkeit

Nachdem der Source Code gemessen worden ist, ist es möglich zu bewerten ob und zu welchem Grade sich der Code wieder verwenden lässt. Im alten Code liegt sehr viel Redundanz. Er enthält auch viele Operationen die mit dem Aufbau der Benutzeroberfläche oder mit der Bedienung systeminterner Schnittstellen beschäftigt sind. Dieser Code ist in einer Web Service überflüssig, bzw. er wird durch anderen Code ersetzt. Je mehr dieser Code mit dem fachlichen Code verflochten ist, desto schwieriger ist es, den fachlichen Code davon zu trennen, d.h. der fachliche Teil des Codes sollte möglichst wenig Verbindungen zu dem anderen Code haben. Das Gleiche trifft für die Verbindungen zu fremden Modulen zu. Ein wiederverwendbarer Baustein soll weder auf globale Daten zugreifen noch fremde Funktionen benutzen.

Anhand der Metrik, ist es möglich zu erkennen ob ein Programm zur Wiederverwendung geeignet ist oder nicht. Insofern als es aus überschaubaren, abgeschlossenen Codebausteinen besteht, die von außen angesteuert werden, können diese Bausteine in einem anderen Zusammenhang wieder verwendet werden. Falls das Programm einen einzigen Eingang mit einer begrenzten Anzahl

Parameter hat, lässt sich das Programm als ganzes wieder verwenden. Besonders wichtig für Online-Programme ist, dass ein Programm nur eine Maske bearbeitet. Diese eine Maske kann durch eine Webschnittstelle ersetzt werden. Wenn mehrere Masken der Reihe nach verarbeitet werden, sind auch mehrere synchrone Webschnittstellen erforderlich. Das eine Web Service muss vor dem nächsten beendet sein. Dies steht wiederum im Widerspruch zur Asynchronität von Web Services. Daher die Notwendigkeit die Benutzeroberflächen von einander zu entkoppeln.

Durch die Bewertung der Wiederverwendbarkeit werden die Programme in drei Klassen geteilt – Programme, die grundsätzlich nicht wieder zu verwenden sind, Programme, die leicht wieder zu verwenden sind und Programme die nur nach einer Refaktorierung wieder zu verwenden sind. Falls ein signifikanter Anteil der Programme zur ersten Kategorie gehört, lohnt es sich nicht das Projekt weiter zu betreiben. In diesem Falle gab es genüg Programme in der zweiten und dritten Klasse. Allerdings müssten einige Programme refaktoriert werden und dass bedeutete ein zusätzlicher Aufwand. Nach der Refaktorierung mussten sie in der alten Umgebung wieder getestet werden um sicher zu stellen, dass die Veränderung keine Fehler verursacht hat. (siehe Abbildung 1)

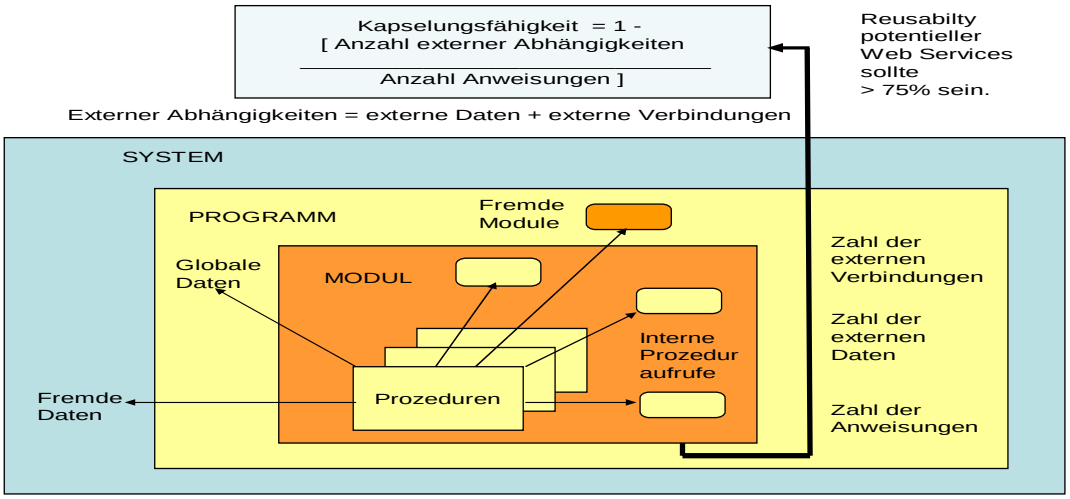

Abbildung 1: Bewertung der Kapselunsfähigkeit

5.3. Code Stripping

Code-Stripping ist eine Testtechnik zur Isolierung einzelner Pfade durch ein Programm damit jeder Pfad allein für sich getestet werden kann. Der eine Pfad bleibt, der Rest des Codes wird aus kommentiert. Es wird so viele Versionen eines Programmes generiert, als es zu testende Pfade hat. Diese Technik ist aus dem ESPRIT Trust Projekt hervorgegangen, dessen Aufgabe es war den Test von Realtime-Embedded Software zu messen. Da man den Code nicht instrumentieren konnte, war es erforderlich ein Pfad nach dem anderen zu testen, unabhängig von den anderen. Ein Pfad ist hier zu verstehen als ein einmaliger Weg vom Eingang bis zum Ausgang durch den Code. Er besteht aus einer endlichen Anzahl aufeinander folgender Anweisungen, wobei einige Anweisungen mehrfach wiederholt werden können. Das Code Stripping wurde automatisch mit Hilfe eines Werkzeuges bewerkstelligt. Das Werkzeug musste nur einen Startpunkt und einen Endepunkt mitgeteilt bekommen. Vom Startpunkt aus verfolgte er jeden Ablaufpfad durch den Code bis zum Ausgang, markierte die Anweisungen auf dem Pfad und blendete die anderen Anweisungen aus. Auf dieser Weise war es möglich jeder Pfad getrennt für sich zu testen. [13]

Im Zusammenhang mit der Kapselung von Code, wird Code-Stripping eingesetzt um Codestreifen heraus zu schneiden. Jede Codestreife ist ein Pfad, der zu einem bestimmten fachlichen Ergebnis führt. Diese Codestreifen bilden potentielle Web Services. Im alten prozeduralen Code wie auch im neuen object-orientierten Code sind die Anweisungen zur Implementierung der Geschäftsregel miteinander verwoben. Es kann vorkommen, dass eine Anweisungsgruppe, bzw ein Codeblock oder eine Methode, in mehreren Pfaden vorkommt. Mit Rücksicht auf die Flexibilität der Geschäftsprozesse ist es aber wichtig, dass ein Web Service einer einzigen Geschäftsregel entspricht. Deshalb müssen die Pfade einzeln auseinander genommen werden. Der Bediener des Werkzeuges muss für jeden Stripping Lauf das gewünschte Ergebnis identifizieren. Danach verfolgt das Werkzeug den Pfad vom Endergebnis zurück zum Ausgangspunkt und blendet den Rest des Codes aus. Was bleibt ist der Code für das Web Service. Dies wird für jede Geschäftsregel wiederholt. Am Ende gibt es mehrere Versionen des bisherigen Programms – eine für jede Geschäftsregel [14]. Dies führt zwar zu einer Vielzahl winziger Services, bietet aber ein Maximum an Flexibilität.

Zur Feststellung ob dies wirklich machbar ist, war es notwendig einige Beispielprogramme zu nehmen und sie durch das Stripping Automat – COBStrip - zu senden. COBStrip erfordert eine menschliche Interaktion. Der Benutzer muss jene Ausgabedaten identifizieren, die zur Response einer Web Service gehören. Zu diesem Zweck werden alle Programmausgaben in einer Liste angezeigt. Bei Online-Programmen befinden sich jene Ausgaben normalerweise in den Datenstrukturen der Masken. Hier handelte es sich um die Message Format Service Masken in IMS-DC. Die Ergebnisdaten von Batchprogrammen liegen in den Berichten und den Ausgabedateien. Durch die Methode des Data-Slicing identifiziert das Tool alle Anweisungen, die zu den erwünschten Ergebnissen führen. Der Rest der Anweisungen wird auskommentiert. Was bleibt ist ein Rumpfprogramm bestehend aus den durchgelaufenden Anweisungen und den von diesen Anweisungen verwendeten Daten. In der Regel enthält ein solches Rumpfprogramm weniger als 20% des ursprünglichen Programmes. D.h. die potentiellen Web Services sind immer nur Teilprogramme. (siehe Abbildung 2)

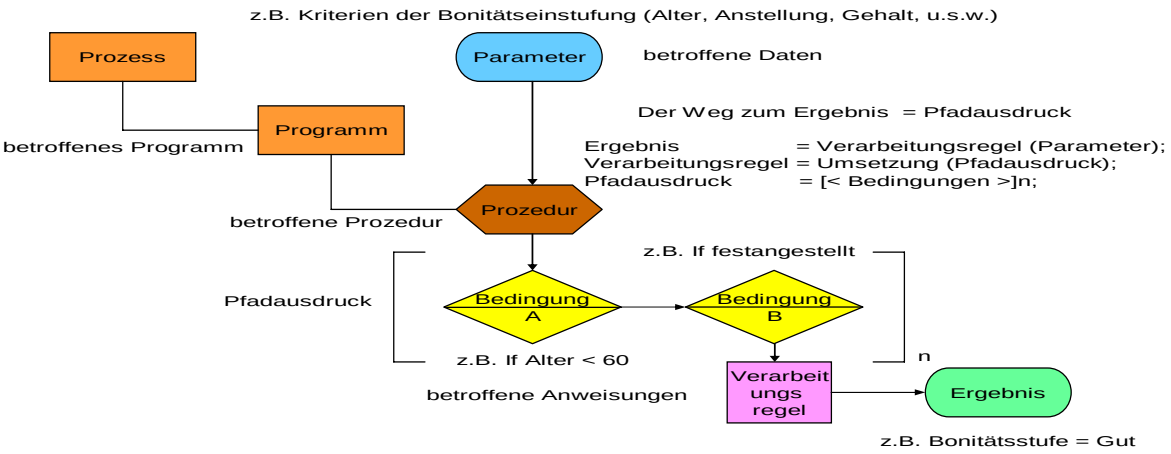

Abbildung 2: Pfadauswahl

5.4. Kapselung der Codestreifen

Nachdem die Programmstreifen als eigene, einzelne Programme vorliegen und erfolgreich compiliert sind, kommt es darauf an, sie hinter einer WSDL Schnittstelle zu kapseln. Dafür sind zwei weitere Werkzeuge erforderlich – COBWrap und COBLink.

COBWrap verarbeitet die Programmstreifen, um die Bildschirmein- und ausgaben mit Aufrufen zu Methoden in den Wrapper-Modulen zu ersetzen. Außerdem werden die Argumente und Ergebnisse in einen Parameterdatenbereich verlagert. Im Falle von IMS, werden die Eingabemasken über einen Aufruf zum TP-Monitor empfangen.

CALL ’DLITCOB’ USING PARAM-NR, IO-PCB, INPUT-MAP

Ein ähnlicher Aufruf bewirkt das Senden der Ausgabemaske an den Endterminal.

CALL ‘COBTDLI’ USING PARAM-NR, IO-PCB, OUTPUT-MAP
COBWrap ersetzt diese originalen Aufrufe durch Aufrufe zum Wrapper-Modul.
CALL ‘WSDL2COB’ USING PARAM-NR, XML-PCB, INPUT-MAP
CALL ‘COBTWSDL’ USING PARAM-NR, XML-PCB, OUTPUT-MAP

Die Daten in dem Parameterbereich werden an den Wrapper-Modul übergeben. Unter CICS sind die Anwenderprogramme Unterprogramme, die Daten in einem Kommunikationsbereich mit dem übergeordneten TP-Monitor austauscht. Im Falle von IMS ist es gerade umgekehrt. Das Anwendungsprogramm ist das Hauptprogramm, der TP-Monitor das Unterprogramm. Es ist deshalb einfacher IMS-Programme zu kapseln. Die Aufrufslogik bleibt unverändert. Nur die Namen und Parameter werden ausgewechselt. Der Autor hat schon über ein ähnliches Kapselungsprojekt bei den deutschen Sparkassen berichtet, bei dem diese Kappselungstechnik bereits erfolgreich verwendet wurde [15]. Bei jenem Projekt ging es um die Kapselung von Assemblerprogrammen und zwar ohne Werkzeug. Hier konnten die COBOL Programme mit Hilfe eines Werkzeuges automatisch gekapselt werden. Eine Kapselung dauert nur Sekunden. (siehe Abbildung 3)

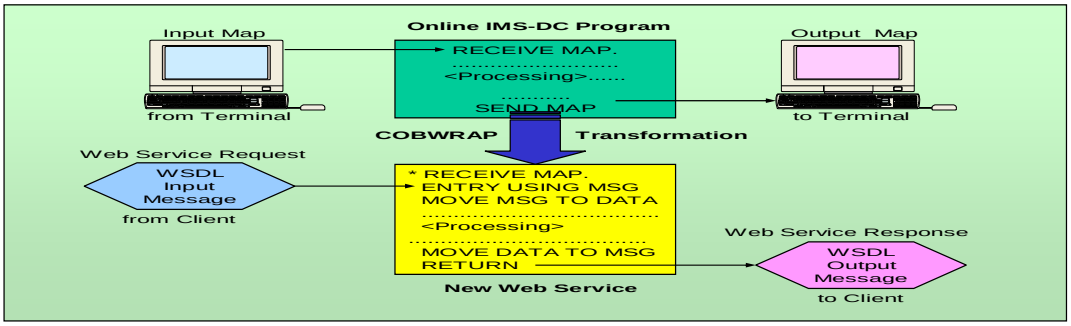

Abbildung 3: Programmkapselung

5.5. Einbindung der gekapselten Teilprogramme

Der letzte Schritt auf dem Wege zu den Web Services ist die Einbindung der gekapselten Teilprogramme in die SOA Umgebung. Dies ist die Aufgabe des Werkzeuges COBLink. COBLink produziert zwei Wrapper-Module für jedes Teilprogramm – einen für die Umsetzung der Web Service Requests und einen für die Umsetzung der Web Service Responses. Die Eingabe dazu ist der Source-Code des Teilprogramms. Daraus entnimmt das Tool welche Parameter zu welchen Schnittstellen gehören. In einem Durchlauf wird ein WSDL Schema für die Eingabeparameter und den dazu gehörigen Wrapper - WSDL2COB - erzeugt. Im anderen Durchlauf wird ein WSDL Schema für die Ausgabeparameter und den dazu gehörigen Wrapper - COB2WSDL - generiert.

Beide Module erben von einer vordefinierten Schablone. Der erste Modul empfängt ein Web Service Request, übersetzt die XML Datentypen in entsprechenden COBOL Datentypen und überträgt sie an das COBOL Programm. Der zweite Modul übersetzt die COBOL Datentypen in entsprechenden WSDL Typen, überträgt sie in die Response und sendet die Response zurück an den Sender. (siehe Abbildung 4)

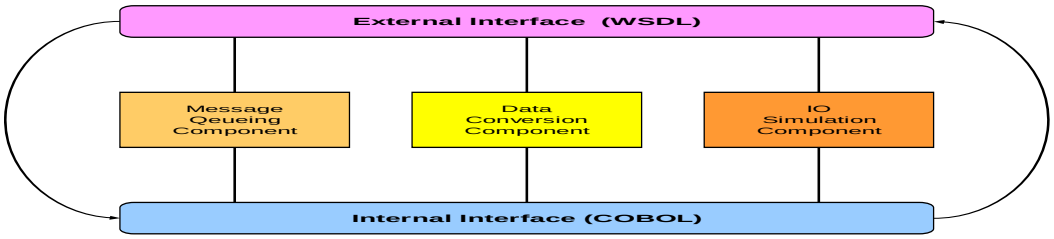

Abbildung 4: Aufbau des Wrappers

Die vier Ergebnisse von COBLink sind die zwei WSDL Schemen – einen Request Schema und einen Response Schema – plus die beiden COBOL Wrapper-Module, die mit dem Web Service Programm zusammengelinkt werden. Das Web Service Programm wird von dem Websphere Scheduler gestartet sobald ein Service-Request in der Warteschlange auftaucht. Es ruft dann den Wrapper-Modul auf, der das Request als Datenstrom aus der Warteschlange liest und an das aufrufende Service-Programm überträgt. Wenn es fertig ist ruft das Service-Programm den Wrapper-Modul auf, um seine Ergebnisse als Response in die Warteschlange zu schreiben. Das Service-Programm übergibt dann die Steuerung zurück an Websphere. Auch diese Technik des Datenaustausches zwischen Web-Server und Web-Clients ist in einer früheren Veröffentlichung beschrieben worden [16].

Im Falle des Pilotprojektes wurde dieser letzte Schritt auf dem PC-Arbeitsplatz similiert, da eine entsprechende Hostumgebung noch nicht aufgebaut war. Es ginge nur darum das Proof of Concept zu liefern und dazu genügte es die Lauffähigkeit der gekapselten Teilprogramme auf dem PC zu demonstrieren. Natürlich sollte später die gekapselten Programme auch auf dem Host getestet werden.

6. Zusammenfassung

Das Pilotprojekt für die Wiederaufbereitung alter COBOL Programme als Web Services hat bestätigt, dass es technisch möglich ist Codescheiben aus dem alten Code heraus schneiden. Diese Scheiben lassen sich auch kapseln und als Web Services in eine Service Architektur einbinden. Dennoch bleibt dieser Ansatz umstritten. Es bleiben viele offene Fragen. Eine Frage ist die der Kosten. Die Erstellung der Web Services ist zwar weitestgehend automatisiert und dadurch mit wenig Aufwand verbunden, aber es bleibt noch der Test. Bei jeder Codetransformation treten Seiteneffekte auf. Diese können wiederum zum Fehlverhalten führen. Ob ein Service sich wirklich wie erwartet verhält, wird man erst wissen wenn alle potentiellen Nutzungsarten getestet worden sind. Der Testaufwand ist hoch, aber dies gilt allgemein für Web Services. Erst muss der Service allein für sich und dann im Zusammenhang mit beispielhaften Geschäftsprozessen getestet werden.

Eine weitere Frage ist die der Eignung der aus dem alten Code herausgenommenen Services. Werden sie wirklich zu den Anforderungen der neuen Geschäftsprozesse passen? Wenn nicht wäre

die ganze Arbeit umsonst gewesen. Diese Frage wird immer auftreten wenn die Services erstmals ohne Bezug zu bestimmten Prozessen entstanden sind. Da hier die Prozesse noch fehlen, kann keiner wissen ob die bereitgestellten Services passen werden oder nicht.

Es weiteres Problem ist die feine Granularität der wieder gewonnenen Services. Ein Web Service hier entspricht einem einzigen Ergebnis – einem Wert oder einer Gruppe logisch zusammengehöriger Werte. So ist die Ermittlung einer Postleitzahl ein eigenständiger Service. Für einen einzigen Geschäftsprozess wird der Prozessverantwortlicher mehrere Hundert Services aufrufen müssen. Ob er dann noch den Überblick behält ist fraglich.

Schließlich stellt sich die Frage der Wartung und –Weiterentwicklung. Da der Code in der gleichen alten Sprache mit der gleichen Qualität bleibt wird die Wartung nicht einfacher. Im Gegenteil, der Source-Code ist durch die Zerschneidung und Kapselung noch komplexer geworden und es werden noch COBOL Programmierer gebraucht um sie zu pflegen. Der Anwenderbetrieb trägt allein die Last der Erhaltung. Langfristig werden die Wartungskosten steigen. Es könnte gut sein, dass der Anwender es nach wenigen Jahren bereut, die Web Services nicht gleich neu gemacht zu haben, statt die Last der Vergangenheit mit sich in die Zukunft zu schleppen.

Literaturhinweise

[1] Lehman, M.: “On Understanding Laws, Evolution and Conservation in the Program Life Cycle”, Journal of Systems and Software, Vol. 1, No. 3, 1980, p. 213
[2] Seacord, R./Plakosh, D./Lewis, G.: Modernizing Legacy Systems, Addison-Wesley, Reading, 2003, p. 120
[3] Horowitz, E.: “Migrating Software to the World Wide Web”, IEEE Software, May 1998, p. 18
[4] Canfora, G./Fasolino, H./ Frattolillo, G.: “Migrating Interactive Legacy System to Web Services”, Proc. of CSMR-2006, IEEE Computer Society Press, Bari, March 2006, p. 23
[5] Krafzig,D./Banke,K./Schama, D.:Enterprise SOA, Coad Series, Prentice-Hall,Upper Saddle River, N.J., 2004, p. 6
[6] Aversano, L./Canfora, G./deLucia, A.: “Migrating Legacy System to the Web”, in Proc. of CSMR-2001, IEEE Computer Society Press, Lisabon, March 2001, p. 148
[7] Bodhuin, T./Guardabascio, E./Tortorella, M.: “Migrating COBOL Systems to the WEB”, WCRE-2002, IEEE Computer Society Press, Richmond, Nov. 2002, p. 329
[8] Tilley, S./ Distante, D./ Huang, S.: ”Web Site Evolution via Transaction Reengineering”, Proc. of WSE 2004, Chicago, Sept. 2004, p. 31
[9] Kontogiannis, K./ Lewis, G. Smith, D.: “ A Research Agenda for Service-Oriented Maintenance” Workshop Proceedings of CSMR-2007, Amsterdam, 2007, p. 100
[10] Sneed, H.: ”Integrating legacy Software into a Service oriented Architecture”, in Proc. of CSMR-2006, IEEE Computer Society Press, Bari, March 2006, p. 3
[11] Sneed, H. “Understanding Software through Numbers”, Journal of Software Maintenance, Vol. 7, No. 6, Nov. 1995, p. 405
[12] Sneed, H.: “Measuring Reusability of Legacy Software” in Software Process, Vol. 4, Issue 1, March, 1998, p. 43
[13] Puhr, P./Sneed, H.: “Code Stripping as a means of instrumenting embedded systems” in EU ESPRIT Project 1258 – Report-1258-3, Liverpool, 1989
[14] Bichler,M./Kwei-Jay, L.: „Service oriented Computing“ IEEE Computer, March, 2006, p. 99
[15] Sneed,H., Majnar, R.: „“A Case Study in Software Wrapping“, Proc. of Int. Conference on Software Maintenance, IEEE Computer Society Press, Washington, D.C. Nov. 1998, p. 86-.93
[16] Sneed, H.: “Wrapping Legacy COBOL Programs behind an XML Interface”, Proc. Of WCRE-2001, IEEE Computer Society Press, Stuttgart, Oct. 2001, p. 189