ICB
Institut für Informatik und Wirtschaftsinformatik

Stefan Eicker
Thorsten Spies
Christian Kahl

Softwarevisualisierung im Kontext serviceorientierter Architekturen

13

ICB-RESEARCH REPORT

ICB-Research Report No.13
February 2007
Universität Duisburg-Essen

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The ICB Research Reports comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Authors' Address:

Stefan Eicker
Thorsten Spies
Christian Kahl

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
D-45141 Essen

stefan.eicker@icb.uni-due.de
thorsten.spies@icb.uni-due.de
christian.kahl@uni-due.de

Managing Assistant and Contact:

Jürgen Jung

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
45141 Essen
Germany

Email: icb@uni-duisburg-essen.de

ISSN 1860-2770

Abstract

Die Softwarevisualisierung trägt dazu bei, die Entwicklung und Wartung von Softwaresystemen und insbesondere die Beherrschung der Systemkomplexität zu erleichtern. Der vorliegende Beitrag beschäftigt sich mit Visualisierungsansätzen im Kontext serviceorientierter Architekturen. Die Architekturen erlauben es, flexible Systemlandschaften zu schaffen, unterliegen jedoch einem stark dynamischen Umfeld. Deshalb bildet ein ausgeprägtes Verständnis sowohl aus technologischer als auch aus fachlicher Sicht einen wesentlichen Erfolgsfaktor bei der Einführung der Systeme. Im Folgenden wird dazu die Konzeption dreidimensionaler Sichten und die Entwicklung einer Visualisierungspipeline zur Erzeugung dynamischer Views vorgestellt. Die Ansätze bieten den unterschiedlichen Stakeholdern jeweils die Möglichkeit, die für sie relevanten Informationen abzurufen und in den Gesamtkontext einzuordnen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Thematik und Kontext

Die Entwicklung und Wartung von Softwaresystemen ist und bleibt ein aufwändiger und kostenintensiver Prozess, da die Komplexität von Software immer weiter zunimmt. Demgegenüber stehen eine schnell wachsende Nachfrage nach Software und neuen Funktionalitäten, die Forderung nach immer kürzeren Entwicklungszyklen sowie sinkende Budgets bei den Investitionen in Softwaresysteme. Der Erfolg bei der Entwicklung und Wartung eines Softwaresystems ist in erster Linie davon abhängig, inwieweit die Komplexität von Software beherrscht werden kann. Der Umgang mit der Komplexität von Software wird schon seit Jahrzehnten, insbesondere im Bereich des Software Engineering, untersucht und erforscht. Die Forschung zielt auf eine kontinuierliche Verbesserung bestehender Methoden, Techniken und Werkzeuge bzw. auf die Entwicklung neuer, innovativer Ansätze ab, um der steigenden Komplexität entgegen zu wirken und somit das Verständnis für Softwaresysteme nachhaltig zu fördern.

Moderne Entwicklungsprozesse sowie der Einsatz etablierter Methoden, Techniken und Werkzeuge helfen sicherlich, den Umgang mit der Komplexität zu erleichtern. Umgekehrt steigt jedoch die Komplexität der Software insbesondere durch ihren wachsenden Umfang und durch die sich immer schneller ändernden Rahmenbedingungen, und dies sowohl aus technischer als auch aus fachlicher Sicht. Diese Steigerung wird sicherlich noch langfristig anhalten.

1.2 Motivation und Zielsetzung

Unternehmensanwendungen, die eng an die Organisation, die Prozesse und das Geschäftsmodell von Unternehmen gekoppelt sind, müssen eine hohe Anzahl unterschiedlicher Anforderungen erfüllen. Nach Krafzig sind Einfachheit, Flexibilität, Wartbarkeit, Wiederverwendbarkeit und die Entkoppelung von Funktionalität und Technologien wichtige Eigenschaften, die moderne Unternehmenssoftwarearchitekturen aufweisen müssen, um die Agilität und Effizienz von Unternehmensanwendungen zu verbessern [KBS05, 6f.].

Serviceorientierte Architekturen (Service-Oriented Architecture, SOA) bieten Konzepte, die es erlauben, Unternehmensanwendungen mit den geforderten Eigenschaften zu realisieren und flexible Systemlandschaften zu schaffen. Die wachsende Verbreitung serviceorientierter Architekturen ist nicht zuletzt auf die steigende Akzeptanz von Web Services und XML-Technologien zurückzuführen, die es erlauben, technologieunabhängige Dienste mit wohl definierten Schnittstellen bereitzustellen.

Ein zusätzlicher Vorteil der Serviceorientierung besteht darin, dass unterschiedliche Perspektiven und Blickwinkel, die innerhalb einer Unternehmensorganisation existieren, berücksichtigt werden.

Das Management sieht Dienste unter dem Gesichtspunkt der Wertschöpfung: Dienste können vermarktet, kommerziell angeboten und abgerechnet werden. Operative Einheiten hingegen verfügen über das Wissen, wie Dienste bereitgestellt werden. Zudem haben sie ein Verständnis dafür, welche Erwartungen an Dienste gestellt werden und wie Dienste auf Basis von Service-Level Agreements (SLAs) differenziert werden können. Entwicklungsabteilungen wiederum wissen, wie Dienste technisch erstellt und entwickelt werden. Dienste besitzen zahlreiche Anforderungen, die es zu dokumentieren gilt, und Prozessmodelle, die iterativ verfeinert werden können. Zusammengefasst trägt die Serviceorientierung stärker zu einem effektiven Business-IT Alignment bei.

Neben den vielfältigen Vorteilen einer SOA existieren jedoch auch Nachteile, Problemfelder und neue Herausforderungen, die bei der Einführung und Adaption serviceorientierter Architekturen ent-

stehen. Zu beobachten ist, dass die Komplexität, nicht zuletzt durch den Zugewinn an Flexibilität, in der Regel noch steigt. Denn insbesondere im Gegensatz zu so genannten Insellösungen, die in sich geschlossene Systeme darstellen, besitzen serviceorientierte Systeme keine klar definierten Systemgrenzen. Das bedeutet, dass nicht nur technologisch - also bei der Implementierung von Diensten auf Basis unterschiedlicher Plattformen - ein umfangreicheres Wissen und Verständnis über das Gesamtsystem gefordert ist. Auch aus fachlicher Sicht ist ein systemweites und domänenübergreifendes Wissen und Verständnis zwingend erforderlich.

Es stellt sich somit die Frage, wie und mit welchen Mitteln serviceorientierte Architekturen gemanagt und wie wichtige Informationen effizient bereitgestellt werden können (Stichwort: „SOA Governance"). Von dieser Fragestellung sind nicht nur einzelne Personenkreise, sondern alle Stakeholder betroffen, die in den Entwicklungsprozess eines SOA-basierten Systems involviert sind und ein Verständnis für das System entwickeln müssen.

Abhängig von den verschiedenen Stakeholdern und deren Perspektiven existieren zahlreiche Sichten auf ein System, die unterschiedliche Aspekte und Informationen in Bezug auf das Gesamtsystem beinhalten. Ein Service-Engineer interessiert sich bspw. dafür, welche Prozesse einen bestimmten Dienst verwenden. Diese Informationen benötigt er insbesondere dann, wenn er die Absicht verfolgt, Änderungen an diesem Dienst vorzunehmen. Im Gegensatz dazu ist ein Prozess-Engineer eher daran interessiert, Informationen über einen Prozess und alle von ihm verwendeten Dienste zu erlangen, um Verbesserungen am Prozess vorzunehmen zu können.

Im Bereich der Softwarearchitekturdokumentation ist das Konzept der Sichten ein fundamentales Prinzip, um die unterschiedlichen Blickwinkel von Stakeholdern zu berücksichtigen [CBB02, 13]. Die Sichten ermöglichen die Betrachtung spezifischer Aspekte eines Systems unabhängig voneinander und tragen somit zu einer Verbesserung des Verständnisses bei. Dabei ist zu beachten, dass nicht eine Sicht alleine die Softwarearchitektur repräsentiert, sondern alle Sichten zusammen die Softwarearchitektur kommunizieren [CBB02, 13; BCK03, 35].

Die Fokussierung eines Stakeholders auf bestimmte Aspekte (Separation of Concern) ist und bleibt ein zentrales Prinzip bei dem Entwurf und der Entwicklung eines Systems. Die Fokussierung auf einen Aspekt bedeutet dabei nicht, dass ein anderer Aspekt ignoriert wird, sondern vielmehr, dass er aus dem gewählten Blickwinkel keine Relevanz besitzt [Dijk74].

Die Fragen, die im Zusammenhang der Gestaltung der Sichten beantwortet werden müssen, sind, welche Aspekte für einen Stakeholder relevant sind, und wie ein Stakeholder bestimmte Informationen und Zusammenhänge betrachten kann, ohne dass sie in der für ihn relevanten Form vorliegen.

Das Ziel der Forschungsarbeiten, die in diesem Beitrag vorgestellt werden sollen, besteht darin, Methoden, Werkzeuge und Darstellungsformen zu entwickeln, mit denen verschiedenen Sichten auf serviceorientierte Architekturen dynamisch (on demand) erzeugt und visualisiert werden können. Die Visualisierung soll dazu beitragen, das Verständnis sowohl aus technischer als auch aus fachlicher Sicht zu verbessern und die Komplexität bei der Erstellung, dem Einsatz und der Wartung von serviceorientierten Architekturen einzugrenzen. Von besonderer Bedeutung ist dabei, dass allen Sichten und deren visueller Darstellung konsistente Informationen und Daten zugrunde liegen.

Ein mit dem skizzierten Ziel verbundenes Teilziel besteht darin, in Kombination mit der Visualisierung Interaktions- und Navigationstechniken bereitzustellen. Die visuelle Darstellung einer Sicht soll Stakeholdern die Möglichkeit bieten, Informationen sowie deren Zusammenhänge schneller zu verstehen, mit Informationen auf unterschiedlichen Detailstufen zu interagieren und diese mithilfe geeigneter Navigationsmittel zu erschließen.

In Kapitel 2 werden zunächst der Bereich der Softwarevisualisierung vorgestellt und unterschiedliche Darstellungsformen beschrieben. Hierbei werden insbesondere die Vor- und Nachteile zwei- und dreidimensionaler Darstellungsformen diskutiert.

Ausgehend von dieser Diskussion wird in Kapitel 3 ein Entwurf für die Darstellung dreidimensionaler Sichten im Kontext serviceorientierter Architekturen vorgestellt. Des Weiteren wird eine Visualisierungspipeline entwickelt, die die Konfiguration dynamischer Sichten berücksichtigt. Die Vorteile dreidimensionaler Darstellungen, insbesondere im Hinblick auf die Erstellung dynamischer Sichten, werden zuvor anhand eines SOA-Ebenenmodells und einer Prozessimplementierung erörtert.

Kapitel 4 stellt einen Prototyp vor, mit dessen Hilfe erste praktische Erfahrungen bei der Umsetzung der Ansätze gesammelt werden sollen.

Kapitel 5 zieht ein Fazit und gibt einen Ausblick auf weitere Forschungsarbeiten.

2 Darstellungsformen der Softwarevisualisierung

Die Softwarevisualisierung als ein Teilgebiet der Informationsvisualisierung und des Software Engineerings soll dazu beitragen, das Verständnis für Softwaresysteme durch deren grafische Repräsentation zu schaffen [PBG03, 315; Balz04, 1; StBi99, 3]. Unter Visualisierung wird nach Diehl das Sichtbarmachen von Daten bzw. Informationen verstanden [Dieh03, 257]. Dementsprechend kann Softwarevisualisierung als die Nutzung grafischer Repräsentationen zur Darstellung unterschiedlicher, statischer oder dynamischer Aspekte von Software definiert werden [Balz04, 1; Dieh03, 257; StBi99, S 3]. Der Fokus der Softwarevisualisierung liegt dabei auf der Analyse und nicht auf der Konstruktion von Software [Dieh03, S. 257]. Visuelle Metaphern sollen helfen, beim Betrachter mentale Bilder, so genannte „Mental Maps“, zu erzeugen [Dieh03, 257f.; MELS95, 186f.; FSC95, 23].

Zurzeit erfolgt die Modellierung bzw. die Visualisierung von Softwaresystemen in der Regel in Form zweidimensionaler Diagramme. Der wohl bekannteste und am weitesten verbreitete Ansatz ist die Unified Modeling Language (UML) [Balz04, 1]. Eine starke Verbreitung und eine hohe Akzeptanz sind die wesentlichen Vorteile der UML [Dwye01, 77].

Die UML beinhaltet eine Sammlung von zweidimensionalen Diagrammen, die für die Modellierung und Visualisierung von Softwaresystemen verwendet werden können. Durch die Vielzahl der verschiedenen Diagramme können alle Aspekte und damit das gesamte darzustellende System abgedeckt werden [Dwye01, 77]. Allerdings ist die UML nur bedingt zur Visualisierung großer Softwarestrukturen geeignet [Balz04, 1; Dwye01, 77]. Dies hat mit generellen Problemen der Visualisierung zu tun.

Mit der Größe eines Softwaresystems wachsen auch der Umfang und die Komplexität der zugehörigen visuellen Darstellungen. Mit deren zunehmender Größe ist vor allen Dingen das Layout der Darstellungen immer schwieriger adäquat zu gestalten, weil immer mehr Elemente und deren Beziehungen mit einbezogen werden müssen [Dwye01, 77f.; StBi99, 16]. Die Menge der Elemente übersteigt ab einem gewissen Zeitpunkt die Menge der vom Benutzer verarbeitbaren Informationen und es kommt zum so genannten *Information Overload* (Informationsüberladung) [StBi99, 4]. Ein *Information Overload* ist die Folge mangelnder Skalierbarkeit wie sie nach Staples und Bieman bei vielen existierenden Ansätzen und Werkzeugen zu finden ist [StBi99, 16]. Skalierbarkeit kann in diesem Zusammenhang als die Fähigkeit verstanden werden, Softwaresysteme unterschiedlicher Größe gleichermaßen gut darstellen zu können.

Zur Vermeidung eines *Information Overload* existieren grundsätzlich zwei Möglichkeiten. Die erste Möglichkeit besteht darin, eine Gesamtsicht des Systems zu erstellen. In diesem Fall muss aber, sobald die Anzahl der Elemente des Softwaresystems steigt, die Größe der einzelnen Elemente

verringert oder müssen Elemente zusammengefasst werden, was aber mit einem Verlust von Semantik verbunden ist [StBi99, 16].

Bei der zweiten Möglichkeit wird lediglich ein Ausschnitt des Systems visualisiert und damit die Anzahl der Elemente verringert, die gleichzeitig dargestellt werden. Der Nachteil hierbei besteht darin, dass der Kontext des betrachteten Ausschnittes durch die Fokussierung verloren geht.

In der Regel existieren mehrere Modelle bzw. Darstellungen eines Softwaresystems auf unterschiedlichen Abstraktionsebenen [StBi99, 5]. Damit lassen sich auch größere Systeme relativ gut darstellen. Allerdings geht der Gesamtkontext verloren und möglicherweise sind zudem nicht alle relevanten Elemente auf einmal darstellbar [StBi99, 16]. Infolgedessen sehen Gil und Kent die mangelnde Verknüpfung der verschiedenen Darstellungen/Diagramme und den damit verbundenen Verlust der Gesamtsemantik als einen der größten Schwachpunkte von Modellierungssprachen wie UML [GiKe98, 106].

Die skizzierten Probleme gelten nicht nur für UML. Viele Programme, die in der Vergangenheit entwickelt wurden, wie etwa der „HP SoftBench Static Analyser", eignen sich nur zur Visualisierung von Systemen mit einer überschaubaren Anzahl von Elementen [StBi99, 16]. Andere hingegen, wie etwa „Imagix 4D", strukturieren das System zwar in kleine, gut visualisierbare Einzelteile und bieten damit einen Lösungsansatz für das Problem der Skalierbarkeit, aber der Gesamtüberblick, das so genannte „Big Picture", geht verloren [StBi99, 8].

Ein weiteres zentrales Problem ergibt sich aus der Zweidimensionalität der Darstellung vieler Ansätze. Wie bereits angesprochen, werden die Darstellungen mit ansteigender Menge der Elemente zunehmend komplexer. Vor allem hierarchische und geschichtete Strukturen, wie sie bei Softwaresystemen häufig zu finden sind, beanspruchen in 2D-Darstellungen sehr viel Platz [Dwye01, 77]. Außerdem kommt es bei einer Vielzahl von Elementen in der Regel zu Überschneidungen von Relationen.

Trotzdem wird, besonders in der Softwarevisualisierung, bisher eher 2D als 3D genutzt, obwohl durch die technische Entwicklung insbesondere im Bereich der 3D-Grafik die 3D-Modellierung zunehmend kostengünstig verfügbar ist. Ein entscheidender Grund hierfür ist sicherlich darin zu sehen, dass 2D-Diagramme einfacher zu erstellen sind als Modelle bzw. Visualisierungen in 3D [GiKe98, 105].

Bezüglich der Wirksamkeit dreidimensionaler Darstellungen stellt Dwyer mit Verweis auf die Ergebnisse wissenschaftlicher Studien fest, dass sie das Verständnis besser fördern als 2D [Dwye01, 78], auch wenn nicht alle empirischen Untersuchungen in diesem Punkt identische Ergebnisse zeigen [Balz04, 2]. In jedem Fall wird durch die Modellierung in 3D eine reichhaltigere Semantik ermöglicht [GiKe98, 105].

Nachteile bisheriger dreidimensionaler Ansätze sind vorwiegend im Auftreten von Verdeckungen sowie Orientierungsprobleme zu sehen [Balz04, 2]. Probleme resultieren unter anderem daraus, dass zum Teil versucht wurde, zweidimensionale Ansätze ohne größere Änderungen in den dreidimensionalen Raum zu übertragen (vgl. [GiKe98, 105ff.; Dwye01, 78]).

Unter Berücksichtigung der zuvor beschriebenen Vor- und Nachteile verschiedener Darstellungsformen wird im folgenden Kapitel eine Konzeption für die dreidimensionale Visualisierung serviceorientierter Architekturen entworfen. Diese dreidimensionale Darstellung soll *nicht* die bewährten Konzepte zweidimensionaler Darstellungen wie bspw. UML ersetzen. Es ist vielmehr der Versuch, auf Basis bestehender Artefakte und Metadaten, die während der Modellierung oder der Implementierung erzeugt werden, Informationen mithilfe einer alternativen Darstellungsform leichter „zugänglich" zu machen.

3 Visualisierungsansätze

3.1 Bestandteile einer SOA

Im Folgenden werden die verschiedenen Ebenen und Bestandteile einer serviceorientierten Architektur in Anlehnung an Erl kurz beschrieben [Erl05, 281ff.]. Erl unterscheidet eine *Business Process*, eine *Service Interface* und eine *Application Layer*. Abbildung 1 zeigt die verschiedenen Ebenen und ihren Aufbau.

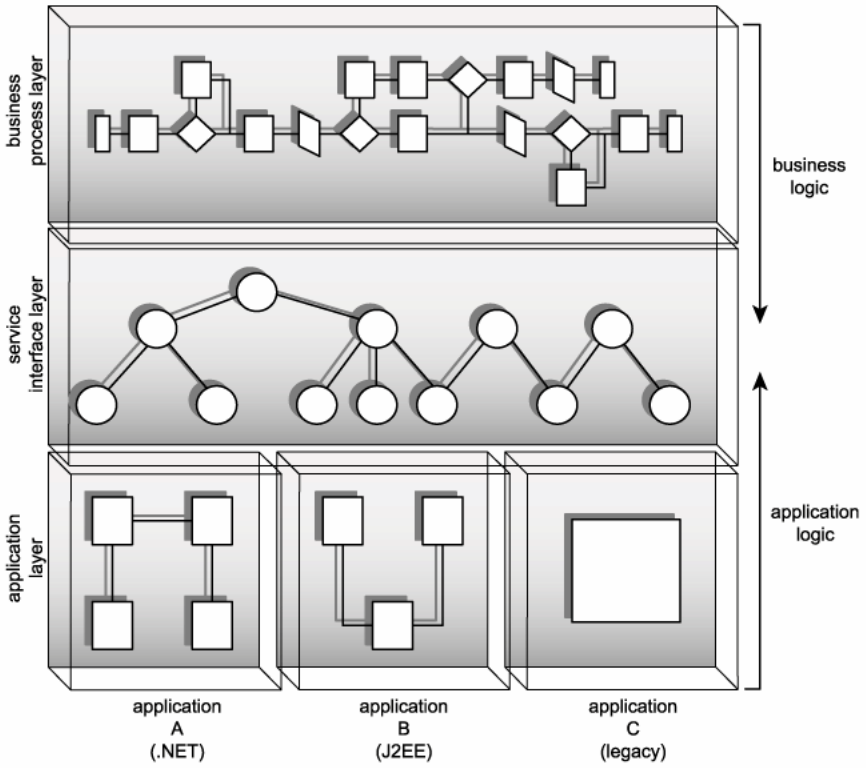

Abbildung 1: SOA-Ebenen [Erl05, 281]

Die *Business Process Layer* implementiert die fachlichen Anforderungen einer Unternehmensorganisation. Sie ist in verschiedene Geschäftsprozesse unterteilt, welche die unterschiedlichen Anforderungen, einschließlich der damit verbundenen Einschränkungen, Abhängigkeiten und externen Einflüsse, abbilden.

Die *Service Interface Layer* stellt Konzepte zur Verfügung, um Unternehmenslogik zu repräsentieren, zu modellieren und zu kommunizieren. Sie besteht aus Diensten, auf die aus anderen Schichten zugegriffen werden kann. Dienste modularisieren die Unternehmenslogik und ihre Bestandteile. Sie stellen in sich geschlossene Einheiten dar, die spezifische Leistungen zur Verfügung stellen.

Die *Application Layer* implementiert Workflows in Form von eigen- oder fremdentwickelten Softwaresystemen. Diese Softwaresysteme existieren innerhalb der IT-Infrastruktur eines Unternehmens und unterliegen den gegebene Sicherheitsbeschränkungen, technischen Fähigkeiten und möglichen Herstellerabhängigkeiten.

Die drei Layer können in weitere Abstraktionsebenen unterteilt werden. Beispielsweise identifiziert Erl innerhalb der *Service Interface Layer* drei weiter Abstraktionsebenen. Sie gliedern sich in eine *Application Service Layer*, eine *Business Service Layer* und eine *Orchestration Service Layer* [Erl05, 282].

Die hier beschriebenen Ebenen und Bestandteile einer SOA stellen nur einen groben Überblick dar, gehen nicht auf die Details ein. Für die im folgenden Abschnitt beschriebene Konzeption zur Visualisierung von serviceorientierten Architekturen ist diese „High-Level"-Perspektive jedoch ausreichend.

3.2 Konzeption dreidimensionaler Sichten

Die in Abschnitt 3.1 beschriebenen Ebenen und Bestandteile einer SOA weisen zahlreiche Abhängigkeiten auf. Auf der einen Seite existieren Abhängigkeiten zwischen den Elementen innerhalb einer Ebene. Bspw. bestehen Prozesse aus mehreren Aktivitäten, die von einander abhängig sind. Des Weiteren können sich Prozesse aus verschiedenen Sub-Prozessen zusammensetzen. Auf der anderen Seite gibt es zahlreiche ebene übergreifende Abhängigkeiten, die insbesondere durch den bereits angesprochenen Prozess der Abstrahierung und den daraus resultierenden Abstraktionsebenen sowie den SOA-typischen Technologiemix entstehen.

Um sowohl ebenen-interne als auch ebenen-übergreifende Abhängigkeiten darstellen und visualisieren zu können, wurde in Anlehnung an Abbildung 1 eine dreidimensionale Repräsentation der SOA-Ebenen erstellt.

In Abbildung 2 erstrecken sich die drei Ebenen entlang der XZ-Achsen. Die höchste Ebene in der Abbildung enthält eine Menge von Geschäftsprozessen.

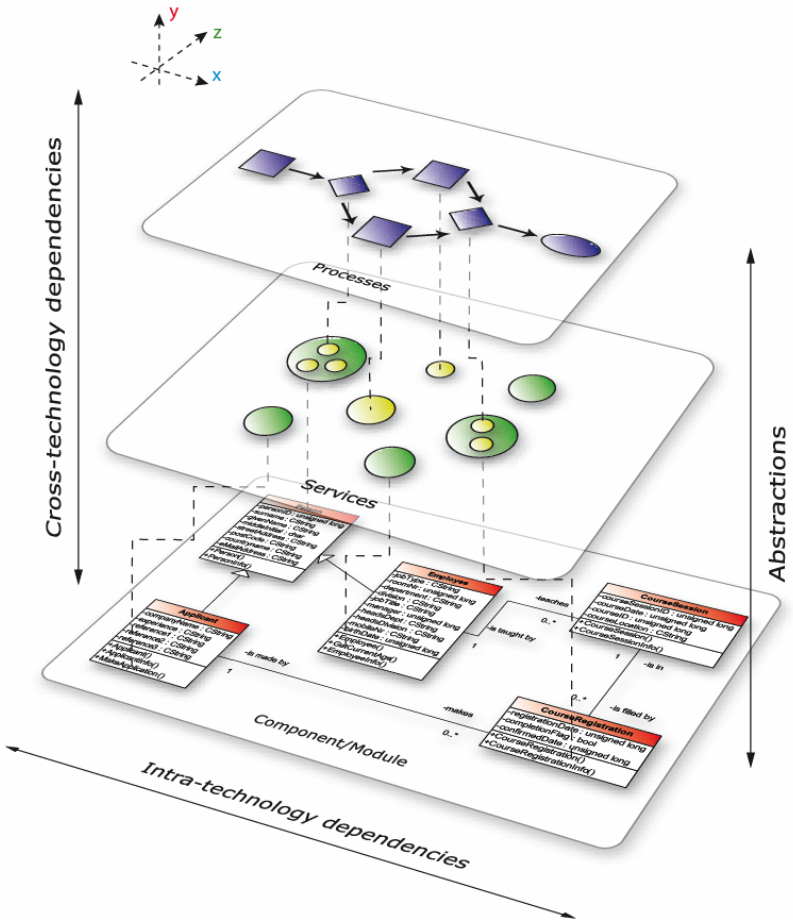

Abbildung 2: 3D-Darstellung der Ebenen

Sie können als Aktivitäten betrachtet werden, die zwischen einem Start- und einem Endpunkt in einer festgelegten Reihenfolge zu durchlaufen sind. Auf der darunter liegenden Ebene erfolgt die Abbildung von Services. Einzelne (atomare) und zusammengesetzte (orchestrierte) Services stellen Funktionalitäten bereit, welche von darüber liegenden Aktivitäten genutzt werden. Die Implementie-

rung der von den Services bereitgestellten Funktionalitäten wird durch Komponenten bzw. Module auf der untersten Ebene realisiert, die wiederum verschiedene Abhängigkeiten aufweisen können.

Der Vorteil der dreidimensionalen Repräsentation besteht darin, dass sowohl verschiedene Abstraktionsebenen als auch die technologieinternen und die technologieübergreifenden Abhängigkeiten zusammenhängend (sowohl entlang der XZ- als auch entlang der XY-Achsen) dargestellt werden können.

Eine entsprechende zweidimensionale Darstellung würde entweder zahlreiche Überschneidungen von Verbindungslinien bei der Darstellung von Abhängigkeiten aufweisen, oder in verschiedenen Diagrammen resultieren, die nicht gleichzeitig visuell dargestellt werden können. Zwar werden durch das Verlinken von Elementen und Diagrammen Konzepte wie „drill-across" oder „drill-down" unterstützt, so dass Zusammenhänge besser nachvollzogen werden können. Der Gesamtkontext geht jedoch, insbesondere bei einer steigenden Anzahl von Informationseinheiten, schnell verloren.

Im Folgenden wird anhand einer Prozessimplementierung die angesprochene Problematik zweidimensionaler Darstellungen näher diskutiert.

3.2.1 Beispiel einer Prozessimplementierung

Für die Prozessimplementierung wurde die Entwicklungsumgebung „IBM Websphere Integration Developer" verwendet, mit deren Hilfe ausführbare Geschäftsprozesse auf Basis des BPEL4WS v1.1 Standards [BPEL03] abgebildet und implementiert werden können.

Abbildung 3 zeigt einen Beispielprozess, der den Vorgang einer Darlehensabwicklung darstellt. Der Prozess ist aus einfachen und strukturierten Aktivitäten zusammengesetzt.

Strukturierte Aktivitäten geben die Reihenfolge vor, in der eine Menge von Aktivitäten stattfinden. Sie beschreiben, wie ein Prozess durch die Strukturierung von einfachen Aktivitäten erstellt wird. Durch die Strukturierung können Kontrollmuster, Datenflüsse, Fehler- und Ereignisbehandlung sowie die Koordination des Nachrichtenaustauschs ausgedrückt werden.

Neben den Aktivitäten existieren sogenannte PartnerLinks, die in der Entwicklungsumgebung als Reference Partners und Interface Partners bezeichnet werden. Mithilfe von PartnerLinks werden Dienste beschrieben, die mit einem Geschäftsprozess interagieren. Jeder PartnerLink wird über einen PartnerLinkType gekennzeichnet. Ein PartnerLinkType wiederum beschreibt die Beziehungen zwischen Diensten während einer Konversation. Er definiert, welche Rollen die Dienste bei der Konversation einnehmen und spezifiziert den PortType, der von jedem Dienst im Kontext der Konversation zur Verfügung gestellt wird, um Nachrichten zu empfangen.

BPEL4WS ermöglicht es, Nachrichten in speziellen Variablen zu speichern, die den Zustand eines Geschäftsprozesses ausmachen. Diese Nachrichten werden entweder von Partnern empfangen, oder an diese gesendet. Die Variablen können jedoch auch Daten speichern, die nur verwendet werden, um den Zustand des Prozesses beizubehalten.

Darüber hinaus unterstützt BPEL4WS die Definition von Korrelationssets, Regelgruppen, Regeln und anderen technischen Konzepten, auf die jedoch aus Gründen der Relevanz hinsichtlich des folgenden 3D-Entwurfs nicht weiter eingegangen wird.

Die vorliegende Prozessimplementierung enthält eine Vielzahl IBM-spezifischer Erweiterungen. Dies ist darin begründet, dass der BEPL4WS-Standard in der aktuellen Version bestimmte Aspekte, die im Rahmen einer Prozessimplementierung notwendig sind, nicht berücksichtigt. So bietet die Version 1.1 des Standards bspw. keine Möglichkeit, menschliche Interaktionen in die Prozessbeschreibung mit einzubeziehen. Ein weiterer Schwachpunkt des Standards ist darin zu sehen, dass BEPL4WS ohne herstellerspezifische Erweiterungen nur in Kombination mit WebServices verwendet werden kann.

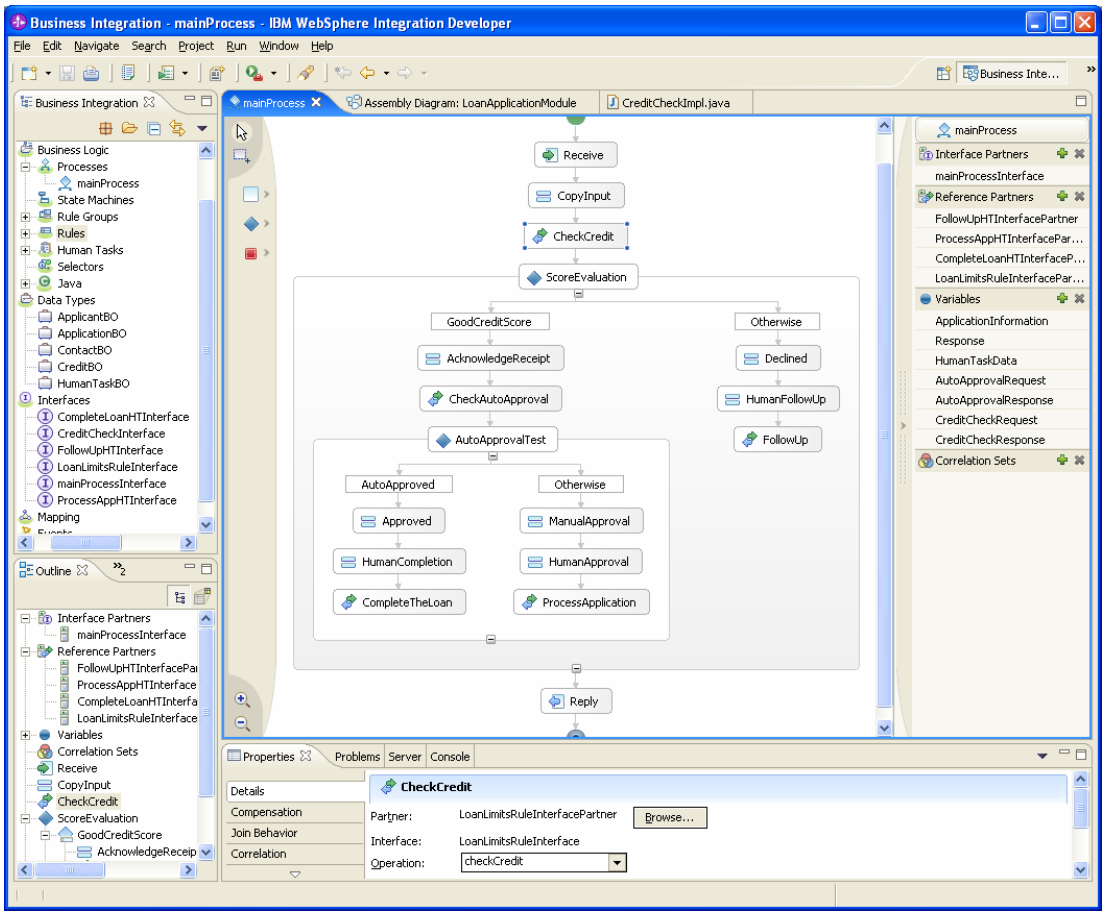

Abbildung 3: BPEL-Prozess im Websphere Integration Developer

Bei der Betrachtung des Prozesses in der Entwicklungsumgebung ist unmittelbar festzustellen, dass die Abhängigkeiten zwischen den Aktivitäten und PartnerLinks im Prozessmodell visuell nicht dargestellt werden. Erst wenn eine bestimmte Aktivität ausgewählt wird, erscheinen in einem zusätzlichen Bereich die Eigenschaften der ausgewählten Aktivität, in denen auch die Beziehung zu dem entsprechenden PartnerLink enthalten ist.

Des Weiteren kann auch bei den in WSDL definierten Schnittstellen und Datentypen keine direkte Zuordnung zu den PartnerLinks erkannt werden. Die Zuordnung wird wie bei den Beziehungen zwischen Aktivitäten und PartnerLinks über entsprechende Eigenschaftsfelder angezeigt. Die Schnittstellen und Datentypen beschreiben, wie auf die zu Grunde liegenden Dienste zugegriffen werden kann, mit denen der Prozess interagiert (Abhängigkeiten der *Business Process Layer* und der *Service Interface Layer*).

Ein Dienst kann auf Basis unterschiedlicher Technologien realisiert werden. Als Kommunikationsprotokolle stehen bspw. SOAP, RMI oder .NET Remoting zur Verfügung. Auch Komponenten bzw. Module, die die Funktionalität eines Dienstes realisieren, können mit unterschiedlichen Technologien wie z.B. EJBs, POJOs oder .NET Komponenten realisiert werden.

Im Beispielprozess werden für die Implementierung zum einen EJBs verwendet, die auf Basis der WSDL-Schnittstellen sowie der Prozessbeschreibung und der darin enthaltenen Logik automatisch generiert wurden. Zum anderen existieren einfache Java-Klassen, die zusätzliche Funktionalität enthalten und manuell implementiert wurden.

Für die Darstellung der Beziehungen zwischen WSDL-Schnittstellen und deren Implementierung steht in der IDE ein sogenanntes Assembly-Diagramm zur Verfügung (siehe Abbildung 4).

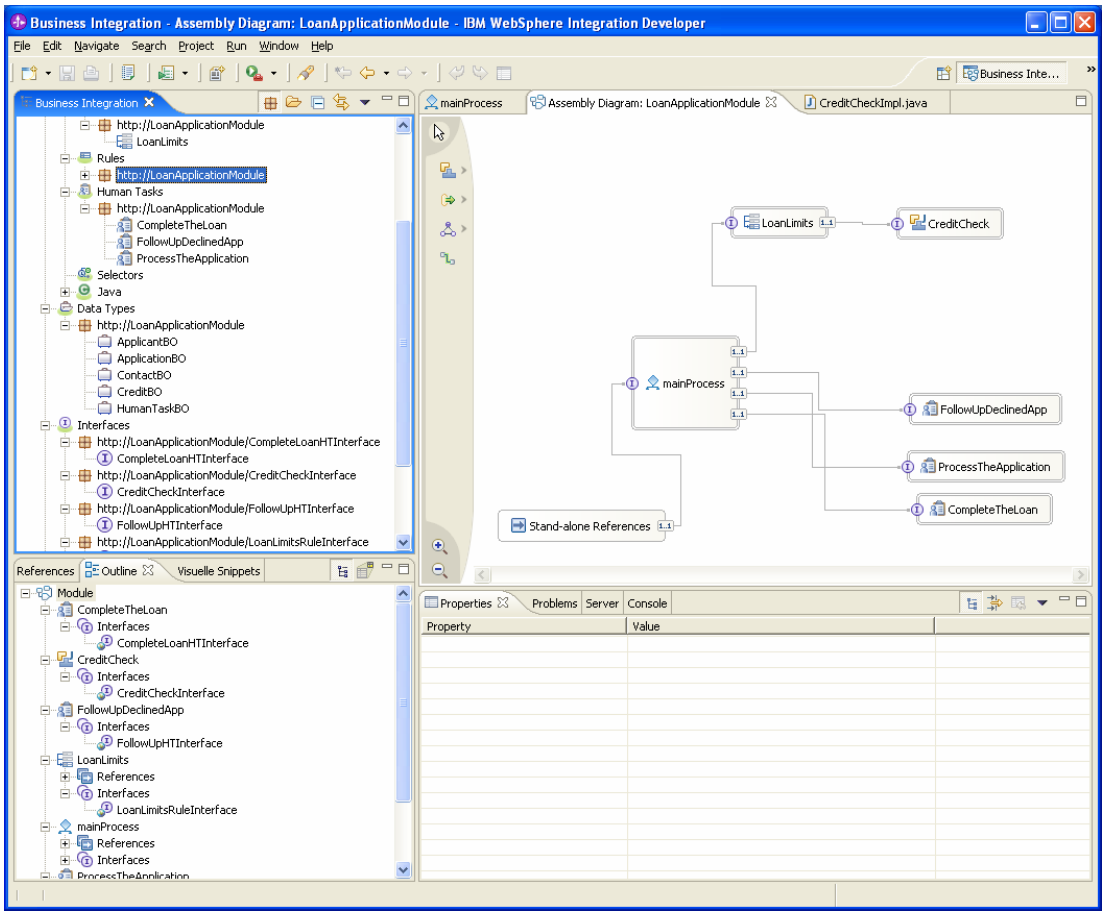

Abbildung 4: Assembly-Diagramm im Websphere Integration Developer

Auf den ersten Blick zeigt das Diagramm jedoch nur die Beziehungen der am Prozess beteiligten Komponenten (mit Komponenten sind hier Komponenten der Prozessebene gemeint, also bspw. Regelgruppen und Human Tasks). Erst bei der Auswahl einer dieser Komponenten können über den zugehörigen Eigenschaftsbereich die entsprechenden Verweise auf die WSDL-Schnittstellen und die Implementierung angezeigt werden (Abhängigkeiten der *Service Interface Layer* und der *Application Layer*). Alternativ können die Schnittstellen auch in einer Gliederungssicht angezeigt werden.

Um jedoch letztendlich an Implementierungsartefakte zu gelangen, muss in der Entwicklungsumgebung die gesamte Perspektive (Java-Perspektive) gewechselt werden, womit der Gesamtkontext in Bezug auf die Prozessimplementierung grundsätzlich verloren geht.

Für einen Stakeholder, der eine Gesamtsicht mit den zuvor beschrieben Abhängigkeiten zwischen den Elementen benötigt, um das System besser zu verstehen, liefert die IDE keine Unterstützung.

Aber auch bei anderen IDEs, wie z.B. Microsofts Visual Studio mit Erweiterungen für den Biztalk Server, ist festzustellen, dass keine optionale Darstellungsform existiert, die die Zusammenhänge der zuvor beschriebenen Informationen visuell darstellen kann. Zu hinterfragen bleibt, ob Entwicklungsumgebungen diese Funktionalität als integriertes Werkzeug zur Verfügung stellen müssen, oder ob diese Aufgabe von externen Tools (z.B. ALM-Tools) übernommen werden kann.

Ausgehend von der in Abbildung 2 gezeigten Darstellung wird im Folgenden ein dreidimensionaler Entwurf des Beispielprozesses vorgestellt, der sowohl die ebenen-internen als auch die ebenenübergreifenden Abhängigkeiten umfasst.

3.2.2 3D-Entwurf der Prozessimplementierung

Mit der Erstellung des 3D-Entwurfs wird im Hinblick auf die Problemstellung und Zielsetzung der Versuch unternommen, im Vergleich zu den bereits existierenden Darstellungsformen im Rahmen der Entwicklung und Modellierung von Software alternative Darstellungsformen zu finden.

Bei dem Entwurf wurde der Fokus auf die Struktur der Prozessimplementierung und auf die Beziehungen zwischen den Elementen gerichtet. Die Darstellung der Elemente ist in der vorliegenden Form durch die Verwendung einfacher geometrischer Primitive noch sehr abstrakt. So werden bspw. alle Aktivitäten mit den gleichen geometrischen Formen abgebildet. Um der gesamten Darstellung eine semantisch höhere Aussagekraft zu verleihen, könnte jedem Elementtyp eine eigene visuelle Repräsentationsform zugeordnet werden. Auch auf die Darstellung von Detailinformationen der einzelnen Elemente wurde in dem Entwurf zunächst verzichtet.

Der Entwurf wurde mithilfe eines 3D-Modellierungswerkzeugs erstellt. Abbildung 5 zeigt ein Rendering, das den Prozess und seine Aktivitäten, die PartnerLinks und Schnittstellen, die Dienste sowie die Bindings und Klassen darstellt. Aus Gründen der Einfachheit sowie mit Blick auf die Entwicklung eines Prototyps wurde das Szenario bezüglich der Dienstimplementierung modifiziert. Bei der Dienstimplementierung handelt es sich nicht mehr um EJBs, sondern um WebServices, deren Funktionalität in einfachen Klassen realisiert wird. Die Struktur des Prozesses zur Darlehensabwicklung bleibt unverändert.

Die Aktivitäten des Prozesses werden in Abbildung 5 durch abgerundete blaue Boxen und den darüber liegenden Nachrichtensymbolen repräsentiert. Für die Darstellung von PartnerLinks und Schnittstellen werden einfache Kugeln verwendet, die unterhalb des Prozesses und dessen Aktivitäten liegen. Um die Zuordnung zwischen PartnerLink und Schnittstelle stärker zum Ausdruck zu bringen, sind die Kugelpaare jeweils von einer transparenten geometrischen Form umschlossen.

Unterhalb der PartnerLinks und den Schnittstellen liegen die Dienste, dargestellt durch transparente Sphären. Innerhalb der Sphären befinden sich die Dienstoperationen, die von den Aktivitäten des Prozesses aufgerufen werden. Unter den Diensten werden die Bindings, welche die Zuordnung zwischen den Diensten und deren Implementierung beschreiben, wiederum durch Kugeln repräsentiert.

Klassen werden durch transparente gelbe Boxen abgebildet. Sie befinden sich unterhalb der Bindings und werden von Modulen umschlossen, wodurch die jeweilige Modulzugehörigkeit verdeutlicht wird. Die Verschachtelung mehrerer geometrischer Formen, wie es z.B. bei den Modulen und Klassen der Fall ist, wird mithilfe unterschiedlicher Transparenzstufen umgesetzt.

Die Verbindungslinien zeigen die Abhängigkeiten zwischen den Elementen. Die Pfeilsymbole an den Linien geben den Nachrichtenfluss wieder.

Der Entwurf macht deutlich, dass die gewählte dreidimensionale Darstellungsform die Zielsetzung erfüllt, sowohl ebenen-interne als auch ebenen-übergreifende Abhängigkeiten visuell abzubilden.

Darüber hinaus sind zahlreiche Erweiterungsmöglichkeiten des Entwurfs denkbar. So könnten bspw. die verwendeten Kommunikationsprotokolle mit in die Darstellung einbezogen werden. Auch ist vorstellbar, neben den statischen Informationen auch dynamische Aspekte zu berücksichtigen. Die Visualisierung könnte z.B. für die Simulation des Nachrichtenflusses verwendet werden, um das Laufzeitverhalten des Systems zu überprüfen. Aber auch fachliche Informationen, wie z.B. Qualitätsmerkmale von Diensten, könnten in die Darstellung einbezogen werden.