it 4/2004

Modellbasierter Ansatz zur Anwendungsintegration

A Model-based Approach for Enterprise Application Integration

Clemens Dorda, Universität Stuttgart,
Hans-Peter Steiert, Jürgen Sellentin, DaimlerChrysler AG

Zusammenfassung Moderne Produkte zur Anwendungsintegration in Unternehmen (EAI) bieten Werkzeuge, um Integrationsszenarien zu modellieren. Allerdings lassen sich damit heterogene IT-Umgebungen bisher immer nur ausschnittsweise darstellen, da die Modelle unterschiedlicher EAI-Produkte nicht ausgetauscht oder integriert werden können. Unser Ziel ist es, die Bildung solcher ‚Integrationsinseln' zu vermeiden. Dazu präsentieren wir einen Ansatz, der durch technologie- und herstellerunabhängige Modellierung eine integrierte Sicht erlaubt. Unser Vorgehensmodell schlägt vor, diese integrierte Sicht werkzeuggestützt auf der Basis von Repositories zu verfeinern, um die Realisierung mit konkreten Produkten und das Deployment auf konkreten Plattformen zu automatisieren. ►►► **Summary** Modern products for Enterprise Application Integration (EAI) provide tools for modelling integration scenarios. Because it is not possible to exchange or integrate the models of different EAI-products with these tools, the real heterogeneous IT-environments can only be described partially. Our goal is to avoid the creation of so-called 'integration islands'. For that purpose we present an approach which allows an integrated view by technology-independent and multivendor-capable modelling. Our process model proposes a toolset- and repository-based refinement of the integrated view to automate the implementation with real products and the deployment on real platforms.

KEYWORDS D.1.2 [Automatic Programming], D.2.2 [Design Tools and Techniques], D.2.12 [Interoperability], D.2.13 [Reusable Software], H.2.5 [Heterogeneous Databases], I.6.5 [Model Development]

1 Einleitung

Integrationsprojekte haben in Unternehmen eine fast so lange Historie wie die Verwendung von Informationssystemen an sich. Die Vorteile und positiven Ziele von Systemintegration, beispielsweise schnellere und automatisierte Abwicklung von Geschäftsprozessen, sind bereits vielfach genannt worden [1; 2]. Oft werden die Gefahren für die IT Infrastruktur allerdings vernachlässigt. So gefährden Integrationsprojekte die Agilität der IT Infrastruktur.

Unter Agilität versteht man die Fähigkeit, sich an Veränderungen anzupassen. Diese Fähigkeit wird jedoch durch jedes neue Integrationsprojekt reduziert, da dabei immer größere verteilte Systeme mit immer stärkeren Abhängigkeiten zwischen bisher autonomen Applikationen entstehen. Beispielsweise führt der Zugriff eines Online-Shops auf den aktuellen Lagerbestand dazu, dass die Lagerverwaltung nur noch dann angepasst werden kann, wenn die Schnittstellen zum Online-Shop stabil bleiben. Sehr schwierig wird es, wenn Abhängigkeiten nicht oder schlecht dokumentiert werden. Insbesondere da das Wissen über die integrierten Applikationen meist nicht in einem Team vorliegt, sondern über viele Personen und Organisationen im Unternehmen verteilt ist.

Im schlimmsten Fall führt dies dazu, dass zentrale Anwendungen nicht mehr weiterentwickelt werden können, weil die Gefahr zu groß geworden ist, andere kritische Systeme unbewusst negativ zu beeinflussen. Es wird deshalb mit zunehmender Integration immer schwieriger, aufwändiger und teurer, die Systemlandschaft an die sich ändernden Bedürfnisse anzupassen. Die Agilität geht verloren.

In diesem Artikel werden wir uns damit beschäftigen, wie sich

die Agilität über die ganze Lebensdauer erhalten lässt. Unser Fokus liegt dabei nicht auf den Produkten zur Systemintegration, da diese von den IT-Herstellern entwickelt und auch fortwährend weiterentwickelt werden. Wir wollen vielmehr einen Ansatz vorstellen, bei dem durch ein Vorgehensmodell für Integrationsprojekte und eine geeignete Werkzeugunterstützung für Entwicklung und Dokumentation insbesondere die Dokumentation gegenüber dem heutigen Stand verbessert wird.

Dieser Artikel ist wie folgt gegliedert. Im nächsten Abschnitt werden wir die Problemstellung genauer herausarbeiten und anschließend vier Szenarien entwickeln, die wir mit unserem Ansatz adressieren. Darauf folgend beschreiben wir eine Auswahl bestehender Ansätze von Herstellern und Lösungsanbietern. Anschließend stellen wir dann unseren Ansatz, das RADES-Vorgehensmodell (*Reference Architecture for the Documentation and Support of EAI-Solutions*) vor und diskutieren, wie dieser die vier Szenarien aus Abschnitt 3 adressiert. Wir schließen mit einer Zusammenfassung und einem Ausblick.

2 Problemstellung

Integrationsprojekte leiden meist unter drei inhärenten Problemen. Diese drei Aspekte wollen wir nachfolgend genauer betrachten und in einem Anforderungskatalog zusammenstellen.

2.1 Inkonsistente Dokumentation

Zunächst betrachten wir das Auseinanderlaufen von Realität und Dokumentation. Diese Situation dürfte hinreichend bekannt sein und tritt sowohl zu Beginn eines Integrationsprojekts als auch während der Durchführung einer nachträglichen Änderung und auch bei der Migration auf eine neue Integrationstechnologie auf.

Ein Blick auf die verfügbare Dokumentation in einer dieser Situationen zeigt, dass für viele der integrierten Applikationen keine solche mehr vorhanden ist. Findet sich jedoch Dokumentation, dann stimmt diese sehr oft nicht mehr mit der real implementierten Integrationslösung oder den integrierten Systemen überein. Hinzu kommt, dass die existierende Dokumentation informell und ungenau ist, wodurch vielfältige Interpretationen möglich sind. Neben der Dokumentation müssen deshalb die realen Systeme als Informationsquelle genutzt werden, was ein aufwändiges Reengineering notwendig macht.

2.2 Heterogenität aller Artefakte

Die genannten Probleme bei der Dokumentation werden zusätzlich durch zwei Effekte verschärft:

Erstens ist die Systemlandschaft über die Zeit hinweg ständigem Wandel unterzogen und wird dadurch bald extrem heterogen. Wir haben es daher mit verschiedenen Integrationsprodukten und Realisierungstechnologien zu tun. Jedes Produkt bringt dabei seine eigenen Werkzeuge und Metadaten mit. Dies macht es schwierig, ein konsistentes Bild der realen Integrationslösung zu gewinnen.

Zweitens wurden die Projekte oft mit wechselnden Projektpartnern bearbeitet. Diese verwenden verschiedene Vorgehensmodelle, Dokumentationsmethodiken und unterschiedliche Notationen. Die Heterogenität bezieht sich also nicht nur auf die IT Systeme, sondern auf alle entstandenen Artefakte des Integrationsprojekts.

2.3 Verteilung der IT und der Organisation

Der dritte Aspekt ist die inhärent verteilte Natur von Integrationslösungen.

Aufgrund der Verteilung müssen Änderungen an vielen Stellen konsistent durchgeführt werden. Dies ist sehr fehleranfällig. Die Auswirkungen einer Änderung auf andere Applikationen lässt sich zudem nur schwer nachvollziehen, was die Fehlersuche erschwert.

Die Verteilung hat auch einen organisatorischen Aspekt, da unterschiedliche Personen und Bereiche für die Applikationen verantwortlich sind. Daher ist ein mühseliges Zusammensuchen der benötigten Informationen notwendig.

2.4 Anforderungen an einen Lösungsansatz

Üblicherweise wird bisher versucht, die dargestellten Probleme technisch und organisatorisch klein zu halten:

Die technische Lösung besteht darin, dass die verwendeten Integrationsprodukte die Applikationen nur lose miteinander koppeln. Eine geeignete Architektur vorausgesetzt, lassen sich so Änderungen in einem System hinter den Schnittstellen für andere integrierte Systeme weitgehend transparent durchführen.

Die organisatorische Lösung besteht in einem guten Projektmanagement und stringenten Dokumentationsrichtlinien. Beide reduzieren die Wirkung der Probleme und sind in jedem Fall wichtig und notwendig. Wir sehen sie als Voraussetzung für unseren Ansatz. Zusätzlich muss ein Lösungsansatz aber folgende Ziele erreichen:

- Es muss eine zentrale Dokumentation geben.
- Die Dokumentation muss eine aktuelle und homogene Sicht auf eine Integrationslösung ermöglichen.
- Ein Vorgehensmodell mit geeigneter Werkzeugunterstützung muss die Konsistenz von Dokumentation und realer Lösung gewährleisten.
- Der Lösungsansatz muss sich nahtlos in Entwicklungsprozesse integrieren.

Diese Ziele präzisieren wir im nächsten Abschnitt anhand von vier Szenarien.

3 Szenarien bei der Entwicklung von Integrationslösungen

Wir werden später unseren Ansatz zur modellbasierten Entwicklung von Integrationslösungen vor-

stellen, der die oben genannten Punkte adressiert. Nach unserer Ansicht muss ein solcher Ansatz die folgenden vier Szenarien in Integrationsprojekten unterstützen.

3.1 Zentrale, einheitliche und automatisierte Dokumentation

Der grundlegende Gedanke dieses Szenarios ist, die Dokumentation dort zu erzeugen, wo sie anfällt: In den Entwicklungs-, Administrations- und Dokumentationswerkzeugen. Oft können Administratoren und Entwickler hier Kommentare hinterlegen. Um eine redundante Speicherung in Dokumenten und Werkzeugen zu vermeiden, ist es notwendig, diese aktiv zu nutzen und automatisch aus den Werkzeugen zu extrahieren.

Als weitere Informationsquellen stehen die Metadaten und Konfigurationsdateien der Werkzeuge und Integrationsprodukte zur Verfügung. Diese stellen die laufende Realisierung dar und müssen ebenfalls automatisch ausgelesen werden.

Zusätzlich fällt natürlich die übliche Dokumentation durch Menschen an. Hier ist eine Standardisierung mit Vorlagen notwendig, um zu ermöglichen, dass diese Informationen ebenfalls maschinell weiterverarbeitet werden können.

In einem definierten Prozess müssen diese Informationsquellen automatisiert und mit Unterstützung eines Autorensystems eingesammelt, homogenisiert, zusammengeführt und zentral bereitgestellt werden.

Wenn die Dokumentation, wie gefordert, nicht alleine einem menschlichen Autor überlassen wird, sondern soweit als möglich automatisch aus den genannten Informationsquellen extrahiert wird, dann lässt sich die Dokumentation einfacher mit der Realität konsistent halten.

Es bietet sich an, die vollständige Dokumentation in einem zentralen Repository abzulegen und online zugreifbar zu machen. Die Verwendung eines zentralen Repository in Verbindung mit Standards für die Dokumentation hilft auch, diese auf Konsistenz untereinander, auf Vollständigkeit, sowie auf Einhaltung von Vorgaben und Richtlinien zu prüfen. Auf diese Weise wird die Homogenität gesichert.

3.2 Neuentwicklung einer Integrationslösung

Oben haben wir beschrieben, wie die Dokumentation einer existierenden Lösung automatisch aus den verfügbaren Informationsquellen erzeugbar sein sollte. In der Entwicklung wird üblicherweise zunächst der umgekehrte Weg beschritten. Ausgehend von einem Grobentwurf wird schrittweise verfeinert und letztlich die lauffähige Lösung implementiert.

Unserer Ansicht nach muss es möglich sein, alle Zwischenergebnisse auf diesem Weg ebenfalls in einem zentralen Repository abzulegen.

Des Weiteren muss es möglich sein, das Ergebnis eines Verfeinerungsschritts auf dessen Konsistenz mit den Vorgaben des Ergebnisses des vorangegangenen Schritts zu prüfen.

Hierzu sind einige Voraussetzungen notwendig: Ein Vorgehensmodell muss klar festlegen, was die Ergebnisse der einzelnen Schritte sein sollen. Ebenso muss die Form der Ergebnisse festgelegt werden. Diese müssen in einer einheitlichen und maschinenverarbeitbaren Notation erstellt werden.

Ferner muss das Repository Sichten auf die verschiedenen Verfeinerungsstufen der Integrationslösung unterstützen, sowie die Navigation von Elementen einer höheren Stufe hin zu deren Verfeinerung und zurück ermöglichen. Die Verfeinerungsbeziehungen zwischen den Stufen müssen dazu wohl definiert und formal fassbar sein.

Die Verfeinerung muss soweit möglich sein, dass sich zuletzt aus dem Inhalt des Repository die Konfigurationsdateien erzeugen und die Metadaten-Repositories der Integrationsprodukte füllen lassen.

Sie werden dazu aus dem allgemeinen Format des Repositories in das proprietäre Format der Integrationswerkzeuge übersetzt.

3.3 Änderung und Wartung einer Integrationslösung

Was für die Erstellung eines Systems gilt, gilt natürlich auch für Änderungen über die Lebenszeit einer Integrationslösung. Dabei sind zwei verschiedene Varianten zu unterstützen, die den beiden bereits beschriebenen Fällen ähneln.

Im ersten Fall wird eine Änderung im Repository dokumentiert und dann dem Vorgehensmodell folgend über die verschiedenen Stufen bis hin zur Generierung von Programmcode und Konfigurationsdateien propagiert.

Im zweiten Fall wird eine Änderung direkt in einem der Integrationsprodukte gemacht, beispielsweise um schnell einen Fehler zu beheben. Diese Änderung muss dann zurück ins Repository propagiert und dort auf Konsistenz mit den höheren Ebenen überprüft werden. Falls die Konsistenz nicht erfüllt ist, müssen wohl definierte Schritte angestoßen werden, um die Änderung so lange nach oben zu propagieren, bis die Dokumentation wieder konsistent ist.

3.4 Migration einer Integrationslösung

Letztlich unterliegen auch Integrationsprodukte der ständigen Weiterentwicklung, sodass die Migration von einer Version auf die nächste oder von einem Produkt auf ein anderes unterstützt werden muss.

Aufgrund der geschilderten Probleme stellen sich Migrationen oft als schwierig dar, insbesondere beim Wechsel der Produkte. Meist bedeutet das eine vollständige Reimplementierung.

Bei Ablage aller Informationen in einem zentralen Dokumentationsrepository kann dieses nicht nur als Informationsquelle genutzt werden, sondern sein Inhalt kann auch als ‚Austauschformat' zwischen den Produkten dienen. Die passenden

Generatoren und Importfilter werden für die oben skizzierten Szenarien sowieso benötigt und können hier direkt in der Frühphase des Migrationsprojekts Nutzen bringend eingesetzt werden.

4 Bestehende Ansätze zur Anwendungsintegration

Bei einer näheren Betrachtung von aktuell auf dem Markt befindlichen EAI-Produkten stellt man fest, dass einige Hersteller inzwischen Mechanismen in ihre mitgelieferten Entwicklungswerkzeuge integriert haben, welche den traditionell eher technisch orientierten Bottom-Up Ansatz der Systemintegration verwerfen und eine Top-Down Entwicklung eines Systemverbundes erlauben bzw. erzwingen. Dadurch können bereits einige der in Abschnitt 2 genannten Probleme gelöst werden [5]. Die folgende Betrachtung zweier exemplarisch ausgewählter Produkte soll aber zeigen, dass dennoch einige Punkte offen bleiben, die unserer Meinung nach nur über einen anderen Ansatz gelöst werden können.

4.1 SAP Exchange Infrastructure (SAP XI)

Als erstes exemplarisches Beispiel sollen hier die Entwicklungsmethodik der SAP Exchange Infrastructure (XI) [14], die Teil der SAP NetWeaver-Plattform ist, und die dafür bereitgestellten Werkzeuge kurz vorgestellt werden. Hier werden während der Entwicklung einer Integrationslösung beteiligte Systeme auf unterschiedlichen Abstraktionsebenen modelliert, teils grafisch, teils textbasiert. Am Ende des Entwicklungsprozesses wird die Konfiguration für die Integrationslaufzeitumgebung aus diesen Daten generiert. Nach der Übertragung der Konfiguration auf die Laufzeitumgebung ist diese dann ohne weitere Anpassungen sofort einsatzfähig.

Die Abstraktionsebenen von XI spiegeln sich in drei verschiedenen Phasen wider, die wiederum jeweils auch mit drei verschiedenen

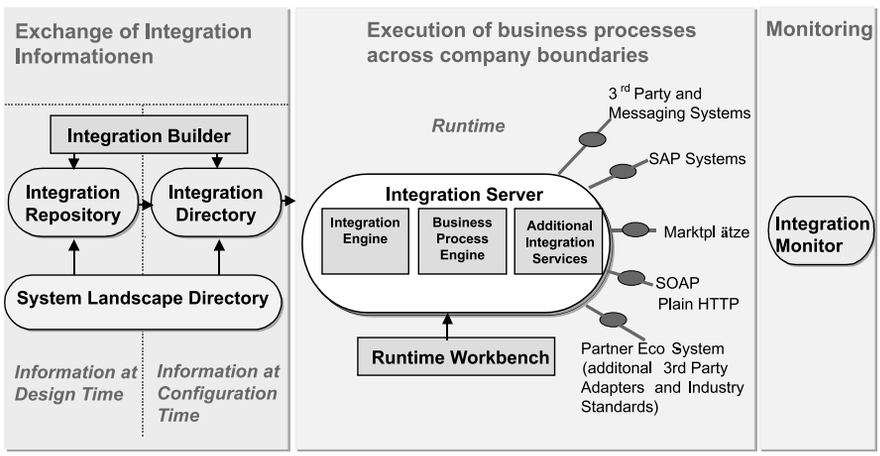

Bild 1 Architektur von SAP XI [1].

Datentöpfen in Form von Repositories und Directories verknüpft sind (Bild 1):

1. Designphase
2. Konfigurationsphase
3. Laufzeitphase.

Zunächst werden in der Designphase Informationen über die beteiligten Systeme (Angaben wie beispielsweise Anwendungsname, Hersteller) und die zu implementierenden Prozesse hinsichtlich Schnittstellen, Nachrichtenformaten, Mapping- und Routing-Regeln angelegt und im so genannten Integration Repository als Komponenten oder Bausteine gespeichert. Optional können hier auch Geschäftsprozesse grafisch modelliert werden, allerdings dient dies in der derzeit angebotenen Version 2.0 nur der Dokumentation und wird somit nicht für die spätere Entwicklung benötigt.

Zur Nutzung der Komponenten und Bausteine für die Konfigurationsphase müssen diese aus dem Integration Repository in das Integration Directory übertragen werden. Dort werden dann die Schnittstellen der Anwendungen mit den in der Designphase angelegten Nachrichtentypen und Mapping- sowie Routing-Regeln verknüpft. Nach dem erfolgreichen Deployment geht die Entwicklung somit in die letzte Phase, die Laufzeitphase, über.

4.2 IBM WebSphere Business Integration

IBM ist einer der Marktführer unter den Anbietern von Integrationsprodukten und bietet neben dem Produkt „IBM WebSphere Business Integration“ noch weitere Integrationsprodukte an, nämlich WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker, und WebSphere Application Server. Den in der Einleitung genannten Anforderungen an Integrationsprodukte kommt das Produkt „IBM WebSphere Business Integration“ (nachfolgend IBM WBI genannt) am nächsten, weswegen wir dessen Konzepte nachfolgend kurz vorstellen wollen [15].

Als Vorgehensweise für die Anwendungs- und Systemintegration schlägt IBM zunächst die Beantwortung der folgenden Fragen vor:

- Welche Probleme müssen im Geschäftsmodell des Kunden speziell gelöst werden?
- Welche Geschäftsprozesse müssen automatisiert oder integriert werden, um die Probleme im Geschäftsmodell zu adressieren?
- Wie sieht die technische Umgebung, in welche die Geschäftsprozesse integriert werden müssen hinsichtlich Anwendungen, Datenbanken und APIs aus?

Diese Sichtweise entspricht unserer in Abschnitt 1 geforderten Top-Down Vorgehensweise. Aller-

dings muss man an dieser Stelle kritisieren, dass sich diese Methodik leider nicht in der tatsächlichen Reihenfolge der Werkzeugverwendung widerspiegelt, die schließlich für ein Integrationsprojekt mit IBM WBI notwendig ist. IBM sieht hier nämlich eine Bottom-Up Vorgehensweise vor, bei der zuerst die technische Konnektivität hergestellt wird und erst am Ende des Entwicklungsprozesses die Geschäftsprozesse in Form von so genannten Kollaborationen entwickelt werden. Kollaboration sind dabei konkret Softwaremodule, welche Geschäftsprozesse beschreiben und später auf der Laufzeitumgebung ausgeführt werden. Durch dieses Konzept bietet hier IBM immerhin die Möglichkeit, losgelöst von technischen Details die dahinter liegende Prozesslogik einer Integrationslösung und somit unterschiedliche Sichten darzustellen.

Hinsichtlich der Wiederverwendung von Komponenten zeigt IBM WBI einige gute Ansätze. Alle benötigten Entwicklungsdaten werden in einem zentralen Repository gespeichert. Über Plug-Ins kann auch eine Versionierung der Entwicklungsdaten realisiert werden. Zusätzlich unterstützt teilweise die Struktur der Metadaten Wiederverwendung. Die oben genannten Kollaborationen beispielsweise gliedern sich in Kollaborationsschablonen und Kollaborationsobjekte auf. Kollaborationsschablonen stellen eine wiederverwendbare Geschäftprozessbeschreibung dar und sind als nichtausführbare Komponente im zentralen Repository des Produkts abgelegt. Sie werden mit Hilfe eines mitgelieferten Werkzeugs grafisch in UML-ähnlicher Notation modelliert [8]. Einmal entwickelt können aus ihnen von jedem nachfolgenden Integrationsprojekt Kollaborationsobjekte erzeugt werden, welche erst nach der Konfiguration durch Anbindung an Adapter oder andere Kollaborationsobjekte ausführbar werden.

4.3 Grenzen bestehender Ansätze

Von der konzeptionellen Seite zeigt der zuerst beschriebene SAP-Ansatz schon einige sehr gute Lösungen, die viele der in der Einleitung beschriebenen Probleme lösen können. Unsere dort formulierte Forderung nach einer homogenisierten Sicht auf die Integrationslösung ist hier sehr gut umgesetzt worden, da durch das definierte Vorgehensmodell eine einheitliche und klar abgegrenzte Sicht in jeder Entwicklungsphase gegeben ist. Ohne auf Freiheiten innerhalb der verschiedenen Phasen verzichten zu müssen, wird der Entwickler dadurch bis zum Ziel geführt. Infolge dessen ist auch, verbunden mit einer integrierten Versionsverwaltung, die Konsistenz der Lösungen in den verschiedenen Phasen hier besser gewährleistet und macht diese vergleichbar. Die Dokumentationsunterstützung ist dafür aber relativ schwach ausgeprägt. Zwar sind viele Informationen, die bei der Entwicklung anfallen, eine gute Ausgangsbasis für die Dokumentation, allerdings ist nur an sehr wenigen Stellen das Hinzufügen von weiterer Dokumentation vorgesehen.

Auch das anschließend vorgestellte IBM-Produkt zeigt einige gute Ansätze hinsichtlich unserer in der Einleitung aufgestellten Forderungen. Durch die fehlende scharfe Trennung zwischen logischen und technischen Daten, sowie die fehlende Festlegung auf ein einheitliches Vorgehensmodell, gestaltet sich hier die Bildung homogenisierter Sichten auf die Integrationslösung wesentlich schwieriger. Dafür unterstützt die UML-ähnliche, grafische Notation der Geschäftsprozesse deren Verwendbarkeit für die Dokumentation, und die Aufteilung in Kollaborationsschablonen und -objekte erleichtert die Wiederverwendung. Auch bei diesem Produkt können aus den bei der Entwicklung angefallenen Daten viele Informationen für Dokumentationszwecke gewonnen werden. Zusätzliches Hinzufügen von Dokumentation ist aber auch hier nur an wenigen Stellen vorgesehen, beispielsweise bei der Implementierung der Kollaborationen in Form von Quellcodekommentaren.

Betrachtet man allerdings die Evolution von IT-Infrastrukturen innerhalb von Unternehmen über mehrere Jahre hinweg, so werden gleichzeitig aber auch schnell die Grenzen beider Ansätze deutlich. Speziell in großen Unternehmen lassen sich schon alleine aufgrund unterschiedlicher Anforderungen selten einheitlich die gleichen Produkte über alle Unternehmensbereiche durchsetzen und dann über Jahre hinweg erhalten. Eine gewisse Heterogenität der Systemlandschaft lässt sich einfach nicht vermeiden. Für die Entwicklung, Dokumentation und Wartung von Integrationslösungen heißt das letztlich, dass sich die anfangs formulierten Probleme, beispielsweise die unterschiedlichen Datenformate von Metadaten, mit bestehender Technologie nicht vollständig lösen lassen [3].

5 Das RADES-Vorgehensmodell

Aufgrund dieser Erfahrungen möchten wir nun einen neuen, allgemeineren Ansatz vorgeschlagen, der die Existenz heterogener Systemlandschaften und Intergrationsarchitekturen berücksichtigt.

Die nachfolgend beschriebenen Konzepte sind im Rahmen des RADES-Projekts (*Reference Architecture for the Documentation and Support of EAI-Solutions*) entstanden, das seit 2002 an der Universität Stuttgart zusammen mit Daimler-Chrysler Research & Technology läuft. Ziel des Projektes ist die Konzeption einer Referenzarchitektur, die Methoden und Werkzeuge zur Verfügung stellt, um Integrationsszenarien einheitlich zu realisieren, dokumentieren und warten. Dabei sollen so weit wie möglich Standards eingesetzt werden.

Die Vision hinter dem Ansatz ist dabei die bereits in Abschnitt 1 formulierte Einrichtung eines zentralen Repositories, das alle Informationen über diejenigen Anwendungen und Systeme beinhal-

tet, welche durch Integrationslösungen miteinander verbunden sind. Es wird hauptsächlich während der Entwicklung einer Integrationslösung infolge eines Integrationsprojektes mit neuen Informationen gespeist. Dazu schlagen wir einen definierten Entwicklungsprozess vor (Bild 2), der einerseits bestimmte Ergebnisse nach jeder Entwicklungsphase fordert und somit die Entwickler des Systemverbundes an eine definierte Vorgehensweise über den gesamten Entwicklungsprozess hinweg bindet, andererseits aber die Vorgehensweise innerhalb der einzelnen Phasen möglichst frei lässt und deswegen hier auch gewisse Freiheiten erlaubt [11].

Für die technische Umsetzung müssen einige Rahmenbedingungen erfüllt sein:

- die Ergebnisse der jeweiligen Phasen müssen in einem Datenformat darstellbar sein, das sich gut strukturiert ablegen lässt und die Formulierung von Abhängigkeitsbeziehungen zu anderen Datensätzen erlaubt,
- die Ergebnisse müssen sich gut zu Dokumentationszwecken verwenden lassen, idealerweise kann das Datenformat auch so dargestellt werden, dass es leicht grafisch aufbereitet werden kann.

Unserer Meinung nach ist dazu eine modellbasierte Entwicklung und Darstellung am besten geeignet, wie sie in den vergangenen Jahren auch immer mehr von renommierten Forschern postuliert wird [16]. Für die Realisierung unseres Vorgehensmodells setzen wir deswegen auf eine Modellierungsnotation, die sowohl die grafische als auch textbasierte Notation der Modelle ermöglicht. Mit UML [8] hat die OMG eine Modellierungssprache entwickelt, welche diese Anforderungen soweit erfüllt und die sich inzwischen in vielen Anwendungsgebieten als Modellierungsnotation etabliert hat [9; 10].

Unsere Vision beschreiben wir im nächsten Abschnitt durch die Formulierung eines Vorgehensmodells für die Anwendungs- und Systemintegration. Anschließend gehen wir auf den UML-basierten Modellierungsansatz ein.

5.1 RADES-Entwicklungsphasen

Angelehnt an den MDA-Prozess (Model Driven Architecture) der OMG [6] wurden für das RADES-Vorgehensmodell drei Entwicklungsphasen definiert, bei denen analog zu MDA als Ergebnis der ersten beiden Phasen ein plattformunabhängiges Modell (PIM) und ein plattformspezifisches Modell (PSM) entwickelt wird (Bild 2) [7].

Beim RADES-Vorgehensmodell wird allerdings der MDA-Ansatz um eine Ebene erweitert. Als Ergebnis der dritten Entwicklungsphase liegt hier kein generierter Code vor, sondern eine weitere Detaillierung des PSM, zugeschnitten auf das Zielprodukt – das so genannte Plattformmodell (PM). Das Plattformmodell soll die Realisierung des Integrationsszenarios auf der Zielplattform exakt beschreiben. Erst nach dem Abschluss der Entwicklungsphasen wird in der Konfigurationsphase aus dem Plattformmodell Code für die Zielplattform generiert und anschließend konfiguriert, um ihn durch Deployment auf die Laufzeitumgebung übertragen zu können.

Als weiterer Unterschied zu MDA ist das RADES-Vorgehensmodell auf die Domäne der Anwendungsintegration spezialisiert. Für diese Domäne wird ein Regelwerk definiert, wie Modelle der einzelnen Phasen aufgebaut sind und welche Bedingungen erfüllt sein müssen, damit ein Modell am Ende einer Phase als syntaktisch vollständig definiert angesehen werden kann. Verbunden mit den Notationsvorgaben, die in Abschnitt 5.2 vorgestellt werden, wird dies im Sinne von [17] als domänenspezifische Modellierungssprache bezeichnet.

Die Modelle aller Phasen werden in dem bereits erwähnten zentralen Repository nach jeder Entwicklungsphase abgelegt. Nachfolgend beschreiben wir, welche Informationen die Modelle der jeweiligen Entwicklungsphasen analog zu Bild 2 nach deren Abschluss beinhalten und wie sie in den darauf folgenden Phasen weiterverwendet werden, bis dann schließlich in der Konfigurationsphase produktspezifischer Code generiert wird und zur Laufzeitumgebung übertragen werden kann.

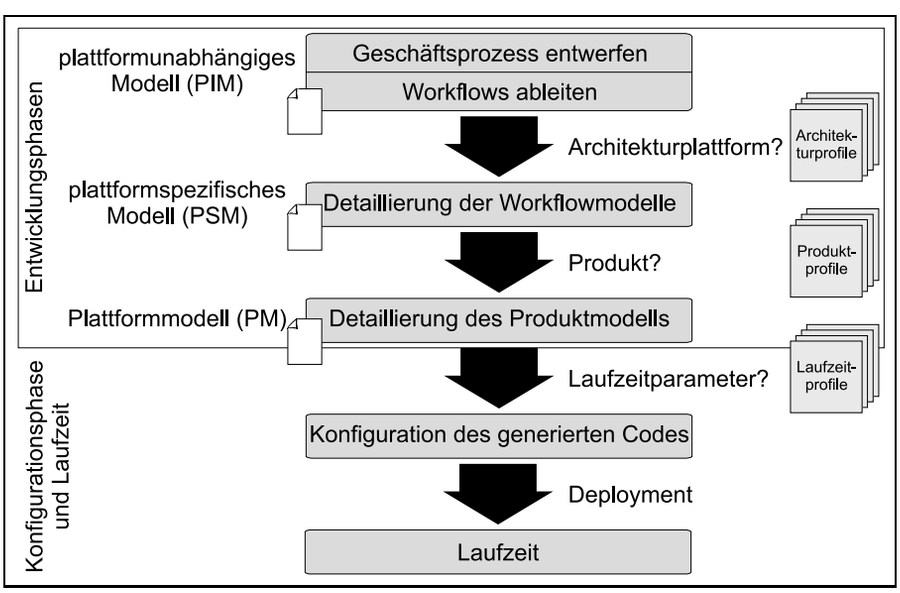

Bild 2 Dreiphasiger Entwicklungsprozess.

5.1.1 Geschäftsprozess entwerfen und daraus Workflows ableiten Grundvoraussetzung für die Durchführung eines Integrationsprojektes ist die Formulierung eines Geschäftsprozesses, der den Integrationsbedarf aus Anwendersicht abstrakt darstellt. In der Praxis zeigt sich leider, dass dieser Geschäftsprozess den an einem Integrationsprojekt beteiligten Personen oftmals nicht klar ist. Aus unserer Sicht ist es aber notwendige Voraussetzung dafür, dass der neu zu schaffende Systemverbund später auch genau die Funktionalität bietet, die Ziel des Integrationsprojektes sind.

Aus der Sicht eines Systemverbundentwicklers muss sich dieser Geschäftsprozess in Form eines oder mehrerer Workflows auf Anwendungen und Ressourcen abbilden lassen. Oft sind für einen Teil der Systeme bereits Schnittstellen geschaffen worden, welche die Systeme untereinander verbinden, andere Systeme sind womöglich noch völlig isoliert. Häufig möchte man auch trotz bereits vorhandener Schnittstellen diese durch eine neue Integrationstechnologie ablösen, weil die existierende Lösung veraltet oder problematisch ist.

Workflows können somit auch als Entscheidungsgrundlage für die Integrationsstrategie herangezogen werden. In manchen Fällen kann es schließlich sinnvoller sein, bestehende Systemverbünde mit Mitteln der vorhandenen Technologie zu erweitern, anstatt sie durch eine komplette Neuentwicklung abzulösen.

Das RADES-Vorgehensmodell sieht deswegen innerhalb der ersten Entwicklungsphase die Formulierung von plattformunabhängigen Workflows vor. Plattformunabhängig heißt, dass in dieser ersten Phase nur Anwendungen, Systeme und Ressourcen, sowie deren logische Abhängigkeiten untereinander betrachtet werden, deren Funktionalität und Daten für die Ausführung des Workflows aus fachlicher Sicht benötigt werden. Folglich spielen hier EAI-Architekturen und Plattformen, Middleware-Systeme oder präzise Beschreibungen von Schnittstellen und Daten noch keine Rolle. Das Ergebnis dieser Phase ist somit eine präzise Übersicht über die zu integrierenden Systeme und deren Beziehungen zueinander in Form von mehreren UML-Diagrammen, die zusammengefasst das plattformunabhängige Modell (PIM) bilden.

5.1.2 Detaillierung der Workflows aus Architektursicht Insbesondere zur Unterstützung von Wartungsaufgaben im späteren Betrieb, wie beispielsweise die Migration auf ein EAI-Produkt eines anderen Herstellers mit der gleichen Architektur, ist es nun notwendig, in einem nächsten Entwicklungsschritt das PIM zu einer plattformabhängigen Formulierung zu detaillieren, dem plattformspezifischen Modell (PSM). Unter einer plattformabhängigen Formulierung verstehen wir dabei eine Formulierung unter Berücksichtigung der angestrebten Integrationsarchitektur, ohne dabei auf spezifische Merkmale eines bestimmten Integrationsproduktes einzugehen. Ein Beispiel dafür ist die von vielen EAI-Produkten unterstützte „Hub-and-Spoke“ Architektur, bei der ein zentraler Broker die Transformation und Zustellung von Nachrichten an die über Anwendungs- und Ressourceadapter angebundenen Anwendungen und Ressourcen übernimmt. Der Übergang von PIM zu PSM soll dabei durch automatische Modelltransformation unterstützt werden, welche durch vordefinierte Architekturprofile möglich wird Abschnitt 5.1.6. Diese Profile beinhalten Informationen über diejenige Integrationsarchitektur, welche die im PIM formulierten Beziehungen zwischen den Systemen technisch realisieren soll. Das so erstellte Grundgerüst des PSM muss nun vom Entwickler zu einem vollständigen PSM erweitert werden.

5.1.3 Detaillierung des Produktmodells Erst in der dritten Entwicklungsphase müssen auch produktspezifische Eigenschaften berücksichtigt werden, wie beispielsweise Namenskonventionen von Warteschlangen (Queues). Dazu soll das in der zweiten Phase erstellte PSM wiederum mit Hilfe eines für das einzusetzende Integrationsprodukt entwickelten Produktprofils zu einem Plattformmodell-Gerüst transformiert und anschließend vom Entwickler zum vollständigen Plattformmodell (PM) vervollständigt werden.

Das Plattformmodell soll so detailliert ausgestaltet sein, dass sich daraus der notwendige Code für die Zielplattform ableiten lässt. Hier müssen deswegen viele produktspezifische Merkmale, wie beispielsweise Strukturen von Transformationsregeln, in das Modell eingebracht werden.

5.1.4 Konfiguration und Laufzeit Nach dem Abschluss aller Entwicklungsphasen kann aus dem im Repository abgelegten Plattformmodell Code generiert und für das Deployment auf die Laufzeitumgebung konfiguriert werden. Dazu sind wiederum Informationen über das Laufzeitsystem notwendig, beispielsweise müssen die zuvor in UML grafisch notierten Nachrichtenformate in eine für die Zielplattform verständliche Form gebracht werden. Auch auf dieser Ebene sollen Änderungen an den generierten Ergebnissen prinzipiell möglich sein. Ziel muss es natürlich dennoch sein, die Modelle der dritten Entwicklungsebene derart präzise auszugestalten, dass möglichst gar keine Änderungen an dem daraus generierten Code notwendig sind.

5.1.5 Deployment Das Deployment ist im Sinne des vorgestellten Entwicklungsprozesses nicht mehr Teil der Entwicklung, sondern Teil des Betriebs. Es hängt sehr stark von den Werkzeugen und Besonderheiten der EAI-Plattform ab, mit der das Integrationsprojekt realisiert wird. Beim Deployment kann es natürlich passieren, dass sich Fehler bemerkbar machen, die einen

Rücksprung in den Entwicklungsprozess erforderlich machen. Beispielsweise kann sich schlimmstenfalls erst hier eine Nachrichtenformatdefinition als fehlerhaft erweisen, weswegen man dann das Plattformmodell der dritten Entwicklungsphase oder möglicherweise sogar das plattformspezifische Modell (Architekturmodell) der zweiten Entwicklungsphase überarbeiten muss. Dabei besteht die Gefahr, oder in manchen Fällen vielleicht sogar die Notwendigkeit, dass die notwendigen Überarbeitungen nicht an den Modellen der jeweiligen Entwicklungsphase vorgenommen werden, sondern direkt an den für das Deployment generierten Daten. Hier ist deswegen ein Mechanismus zur Änderungspropagation analog zu Abschnitt 3.2 notwendig.

5.1.6 Unterstützung der Phasenübergänge durch Modelltransformation und Codegenerierung

Wie bereits angesprochen werden in der ersten Phase, der plattformunabhängigen Entwicklung, zunächst die logischen Aspekte des zu erstellenden Systemverbundes beantwortet, im wesentlichen *wer kommuniziert wann mit wem*.

Vor dem Übergang in die nächste Phase, in der das plattformspezifische Modell erstellt wird, muss zunächst die Zielarchitektur ausgewählt werden. Sehr viele Hersteller von EAI-Produkten setzen bei ihren Produkten auf eine Sternarchitektur mit einem zentralen Broker. Dieser bindet alle zu integrierenden Anwendungen über spezielle Anwendungsadapter an sich und steuert über eine regelbasierte Nachrichtenzustellung oder einen Publikations- und Subskriptionsmechanismus (eng.: *Publish/Subscribe*) den Nachrichtenverkehr zwischen den angebundenen Systemen. Häufig werden auch Busarchitekturen implementiert. Hier wird in der Regel der Publikations- und Subskriptionsmechanismus bevorzugt eingesetzt, eine regelbasierte Nachrichtenzustellung kann damit aber auch realisiert werden.

Aufgrund der gemeinsamen möglichen Kommunikationsmuster können diese Systeme trotz ihrer architektonischen Unterschiede auf dieser Ebene zunächst gemeinsam behandelt werden. Sie lassen sich deswegen so mit den zwei Architekturschablonen „Stern- oder Busarchitektur mit regelbasierter Nachrichtenzustellung“ und „Stern- oder Busarchitektur mit Publikations- und Subskriptionsmechanismus“ beschreiben, woraus sich ein Grundgerüst eines plattformspezifischen Modells für diese Architekturen aus dem plattformunabhängigen Modell generieren lässt. Auch andere EAI-Architekturen lassen sich so abbilden. Beispielsweise kann die Architektur eines Integrationsproduktes, das die Web Services-Technologie implementiert, von den oben genannten typischen EAI-Architekturen abweichen und dennoch alle wichtigen Konzepte von EAI realisieren.

Nach der manuellen Verfeinerung dieses Grundgerüsts zu einem vollständig definierten PSM kann der Übergang zu einem Plattformmodell vollzogen werden. Erst hier kommen auch technische Spezialitäten der jeweiligen Implementierungen zum Tragen. Vor dem Übergang zum Plattformmodell muss deswegen die Entscheidung für ein konkretes Produkt zur Realisierung des Integrationsszenarios gefallen sein. Um das plattformspezifische Modell zu einem Grundgerüst eines Plattformmodells transformieren zu können, sind deswegen auch hier Transformationsprofile in Form von Produktprofilen notwendig, die Informationen über die speziellen Ausprägungen des Produkts beinhalten.

Nachdem das generierte Grundgerüst des Plattformmodells zu einem vollständig definierten Plattformmodell manuell verfeinert wurde, steht als letzter Schritt des Entwicklungsprozesses die Generierung von Konfigurationsdaten, Code, oder anderen vergleichbaren Information über die Zielplattform an. Deswegen werden hier neben dem Plattformmodell weitergehende technische Informationen über die Zielplattform zur Generierung benötigt, beispielsweise wie Nachrichtenformate technisch repräsentiert werden, oder in welcher Art und Weise Transformationsregeln technisch ausgedrückt werden.

Durch die starken Abhängigkeiten, die insbesondere zwischen zwei benachbarten Phasen existieren, bieten sich Mechanismen an, die einen zumindest semi-automatischen Übergang zwischen benachbarten Phasen ermöglichen. Die hier vorgestellte profilunterstützte Modelltransformation soll dies für RADES ermöglichen. Offensichtlich ist, dass die jeweiligen Modellinterpreter, welche die Transformation eines Modells zu einem Modell der nächsten Phase durchführen, dazu sehr viel Wissen über die Beschaffenheit von Modellen der Ausgangs- und Zielphase benötigen müssen. Das semantische Wissen über die Modelle gelangt folglich erst durch die Modellinterpreter in unser Vorgehensmodell. Obwohl diese Modellinterpreter für RADES bisher noch nicht existieren, zeigen Ansätze wie MIC [17], dass dieser Ansatz grundsätzlich realisierbar ist.

5.2 Modellierung bei RADES

Für die Darstellung der Informationen in den jeweiligen Ebenen sieht die OMG für MDA eine Modellierung mit UML (*Unified Modeling Language*) vor. Speziell für EAI hat die OMG inzwischen auch eine UML-Profilspezifikation entwickelt [12]. Auf der Notation dieses Profils basiert auch die Modellierung bei RADES, die wir in diesem Abschnitt beispielhaft vorstellen wollen. Aufbauend auf dem letzten veröffentlichten Dokument der OMG zu „UML Profile for EAI“ werden bei RADES zwei gängige Modellierungsansätze unterstützt, nämlich *Kollaborationen* und *Aktivitäten*.

1. *Kollaborationen*: mit ihnen modelliert man die Art und Weise

der Zusammenarbeit zwischen den zu integrierenden Systemen, beispielsweise in Form von Abhängigkeiten zwischen den Systemen. Außerdem werden die Nachrichtenflüsse definiert. Als Diagrammtypen kommen *Klassen-* und *Kollaborationsdiagramme* zum Einsatz,

2. *Aktivitäten:* sie formulieren das Modell des Geschäftsprozesses bezüglich der zu implementierenden Integrationspunkte auf verschiedenen Abstraktionsebenen durch die Modellierung von Kontroll- und Nachrichtenflüssen. Als Diagrammtypen kommen hier *Aktivitäts-* und *Sequenzdiagramme* zum Einsatz.

Die UML-Elemente der jeweiligen Diagrammtypen werden um Stereotypen ergänzt, um deren Rolle im Diagramm klar kennzeichnen zu können. Dadurch wird der Einsatz von Modellinterpretern zum Übergang in die nachfolgenden Phasen möglich, analog zu Abschnitt 5.1.6.

Zur Modellierung werden in der „UML Profile for EAI"-Spezifikation Stereotypen für Anschlüsse (*Terminal*), Operatoren (*Operator*) und Operatoren-Verbünde (*Compound Operator*), Ressourcen (*Resource*) und Nachrichtenformate (*Message Format*) definiert (Bild 3). Einige dieser Stereotypen besitzen auch Unterstereotypen. Der Stereotyp *Resource* beispielsweise besitzt die Unterstereotypen *Database*, *Queue* und *Subscription Table*, um Datenbanken, Warteschlangen von Messaging-Middleware, sowie Subskriptionstabellen zu beschreiben. Modelle für Subskriptionstabellen werden für EAI-Systeme benötigt, die Publikations- und Subskriptionsmechanismen zur Nachrichtenverteilung anbieten.

Diese Stereotypen helfen nun, die verschiedenen Komponenten eines Integrationsszenarios zu kennzeichnen. Legacy-Systeme beispielsweise werden bei einem noninvasiven Integrationsansatz[1] als Operatoren betrachtet, deren innerer Aufbau für die Einbindung in die Integrationslandschaft nicht relevant ist. Sie erhalten deswegen den Stereotyp *PrimitiveOperator*. In der ersten Entwicklungsphase wird die Art und Weise, wie die Systeme untereinander technisch angebunden sind, nicht formuliert. Hier werden nur Assoziationen zwischen Operatoren, Operatoren-Verbünden und Ressourcen in Klassendiagrammen modelliert, sowie die zeitliche Abfolge von Interaktionen zwischen diesen Komponenten in Aktivitäts- oder Sequenzdiagrammen.

Erst in der zweiten Entwicklungsphase werden Eingangs- und Ausgangsschnittstellen (als Terminal-Stereotyp), Nachrichten- und Datentypen, sowie die Art und Weise des Informationsaustauschs modelliert, beispielsweise ob dieser über Nachrichten oder Prozeduraufrufe stattfindet. Bei einer plattformspezifischen Modellierung einer „hub-and-spoke" EAI-Architektur zum Beispiel werden Ein- und Ausgangsschnittstellen oft über Adapter mit dem Integrationsserver verbunden, wofür die Stereotypen *SourceAdapter* bzw. *TargetAdapter* vorgesehen sind.

In der letzten Entwicklungsphase werden dann produktspezifische Merkmale modelliert, beispielsweise produktspezifische Felder in Nachrichtenköpfen. Je nach Produkt kann es hier notwendig sein, eigene Stereotypen zu definieren, es stehen aber mindestens alle Konstrukte der PSM-Ebene zur Verfügung.

[1] Integration über bereits bestehende Schnittstellen des Anwendungssystems. Im Gegensatz dazu wird bei einem invasiven Integrationsansatz das Anwendungssystem so verändert, dass optimierte Schnittstellen zum Anwendungssystem für die Integration zur Verfügung stehen.

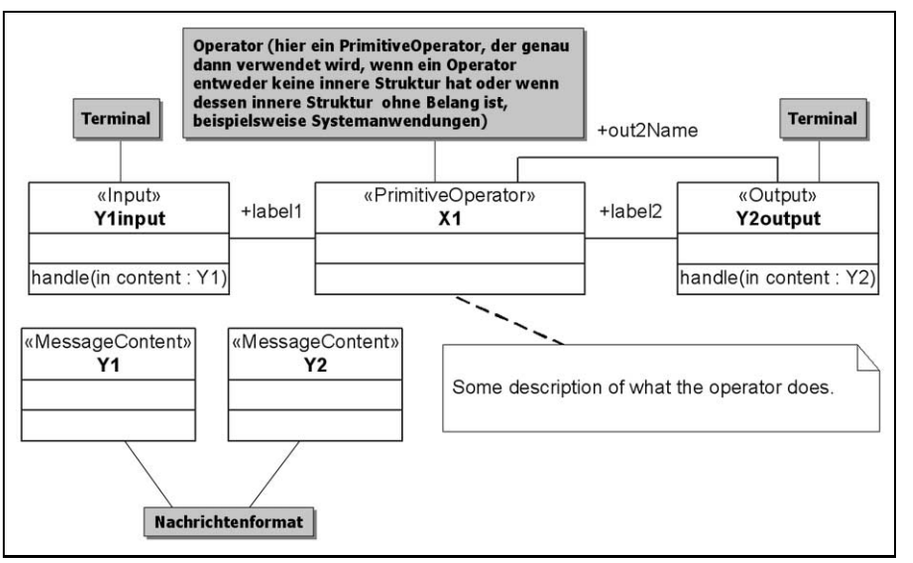

Bild 3 Terminals, Operator und Nachrichtenformate.

5.3 Unterstützung der Szenarien

In Abschnitt 3 haben wir Szenarien formuliert, in denen unserer Meinung nach die zentralen Aufgaben während des Lebenszyklus einer Integrationslandschaft bearbeitet werden müssen. Wir wollen diese Szenarien an dieser Stelle noch einmal aufgreifen und darlegen, warum sich RADES unserer Meinung nach besonders zur Unterstützung dieser Szenarien eignet.

Szenario 3.1, in dem wir eine zentrale, einheitliche und automatisierte Dokumentation fordern, wird durch RADES durch zwei Konzepte unterstützt. Erstens sieht RADES über ein Repository eine zentrale Datenhaltung der Modelle und der darin implizit und explizit enthaltenen Dokumentation vor. Zweitens sorgt die einheitliche Notation der Modelle in allen mit RADES durchgeführten Projekten

dafür, dass diese besser als bisherige Ansätze für die Dokumentation werkzeuggestützt aufbereitet werden können. Szenario 3.2 wird durch das RADES-Vorgehensmodell unterstützt, das einen „roten Faden" durch den Entwicklungsprozess bildet und durch die unterschiedlichen Abstraktionsebenen verschiedene Sichten auf die Integrationslösung bietet. Durch die drei Abstraktionsebenen bietet RADES somit auch Aufsetzpunkte für die Szenarien 3.3 und 3.4, je nachdem, ob sich Änderungen an den Workflows, an der Architektur, oder am Produktmodell widerspiegeln sollen.

5.4 Verwandte Arbeiten

Es gibt Forschungsprojekte, die in verschiedener Hinsicht einen ähnlichen Ansatz wie das RADES-Projekt verfolgen. Bezüglich der Entwicklungskonzepte sind hier besonders die Projekte rund um Model-Integrated Computing hervorzuheben [17; 18]. Bei MIC werden in einer domänenspezifischen Modellierungssprache alle relevanten Informationen über ein zu entwickelndes System modelliert und mit Hilfe von Modellinterpretern in andere Modelle transformiert oder zu Code generiert. Da MIC aber generell für beliebige Domänen verwendbar sein soll, legt es im Gegensatz zu RADES zum einen kein Vorgehensmodell fest, und gibt zum anderen auch keine domänenspezifische Modellierungssprache vor, beispielsweise für die Anwendungsintegration.

Das Projekt „Integration Engineering" [19] bearbeitet eine dem RADES-Projekt ähnliche Aufgabenstellung. Ziel dieses Projektes ist die Entwicklung einer Methodik zur Abbildung kooperativer Geschäftsprozesse auf eine internetbasierte IT-Struktur. Da es sich noch um ein recht junges Projekt handelt, sind bisher leider allerdings nur wenige Informationen verfügbar.

6 Zusammenfassung

Ein immer wieder unterschätzter Faktor bei der Systemintegration ist die notwendige Agilität und Flexibilität, da immer wieder neue IT-Konzepte und Technologien entstehen und auch eingesetzt werden. Weiterhin gibt es immer neuen Bedarf für die Integration von Systemen – sei es zur Optimierung interner Abläufe, die Auswirkung einer Fusion oder die zeitlich begrenzte Kooperation mehrerer Partner.

Bestehende kommerzielle Produkte erzielen meistens auf Basis der ihnen zugrunde liegenden Technologien und Konzepte eine geeignete Integration entsprechend dem aktuellen (Projekt-)Bedarf. Über die Zeit entstehen so aber eine Reihe isolierter Integrationsinseln, die sich nur selten miteinander integrieren lassen. Langfristig betrachtet wird deren Integration mit anderen Systemen oder anderen Integrationsinseln aber nötig werden.

Die OMG hat andererseits mit ihren Arbeiten zur *Model Driven Architecture* (MDA) ein wichtigen Schritt zur Einführung homogenisierter Beschreibungsmodelle begonnen. Die Anwendung dieser Konzepte erfordert jedoch eine weitere Konkretisierung entsprechend der aktuellen Domäne. Weiterhin fehlt es natürlich noch an einer angemessenen Tool-Unterstützung.

Mit unserem Ansatz versuchen wir das Beste beider Welten zu verbinden. Dabei fokussieren wir uns auf die Domäne der Anwendungsintegration (EAI). Die Basis bilden MDA und UML. In Abschnitt 5 haben wir notwendige Erweiterungen des Vorgehensmodells sowie die Auswahl der für die Modellierung benötigten Stereotypen beschrieben. Unsere Erweiterungen sollen insbesondere sicherstellen, dass eine Abbildung auf konkrete Technologien und Produkte möglich ist. Beim Übergang zum Betrieb rücken dann die kommerziellen Tools in den Vordergrund, ohne dass wir unser durchgängiges Modell durchbrechen.

Obwohl im Detail sowohl manche konzeptionellen als auch technische Fragen noch offen sind, sind wir sicher, dass unser Ansatz eine wesentliche Verbesserung der bisher gängigen Praxis bei der Systemintegration darstellt, die bisher zu sehr technologiegetrieben durchgeführt wird und sich unserer Meinung nach zu wenig mit den fachlichen Anforderungen beschäftigt.

Literatur

[1] *H. Häuschen*: EAI – Enterprise Application Integration, URL: http://www.ifi.unizh.ch/ikm/Vorlesungen/ebusiness/ws03/material/FolienEAI_F.pdf [Zugriff am 30.01.2004].

[2] *S. Möckel*: EAI-Überblick und Basistechnologien des EAI, URL: http://ais.informatik.uni-leipzig.de/download/ 2002s_s_ieb/SilvioMoeckel_EAI.pdf [Zugriff am 30.01.2004].

[3] *D. Roedner*: The Next Wave of Integration Platforms. – In: eai Journal, September 2002, S. 12–15.

[4] *D. Draxler, C. Ungerböck*: Enterprise Application Integration. Seminararbeit, Universität Wien 2002, S. 37.

[5] *W. van den Heuvel, W. Hasselbring, M. Papazoglou*: Top-Down Enterprise Application Integration with Reference Models. – Third International Workshop on Engineering Federated Information Systems, Dublin, Ireland, Juni 2000.

[6] *Object Management Group (OMG)*: Model Driven Architecture (MDA), URL: http://www.omg.org/mda [Zugriff am 26.01.2004].

[7] *W. Emmerich:* Distributed Component Technologies and their Software Engineering Implications. – In: Proceedings of the 24th International Conference on Software Engineering, Mai 2002, S. 537–546.

[8] *Object Management Group (OMG)*: Unified Modeling Language (UML), URL: http://www.omg.org/uml [Zugriff am 26.01.2004].

[9] *J. Miller*: What UML should be. – In: COMMUNICATIONS OF THE ACM, November 2002/Vol. 45, No. 11, S. 67–69.

[10] *B. Selic, G. Ramackers, C. Kobryn*: Evolution, not Revolution. – In: COMMUNICATIONS OF THE ACM, November 2002/Vol. 45, No. 11, S. 70–72.

[11] *G. Fairbanks:* Why Can't They Create Architecture Models Like „Developer X"? An Experience Report. – In: Proceedings of the 25th International Conference on Software Engineering, Mai 2003, S. 548–552.

[12] *Object Management Group (OMG)*: UML Profile for Enterprise Application Integration (EAI), URL: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML_for_EAI [Zugriff am 26.01.2004].

[13] *J.H. Hausmann, S. Kent*: Visualizing Model Mappings in UML. In: Proceedings of the 2003 ACM symposium on Software visualization, Juni 2003, S. 169–178.

[14] *SAP AG*: SAP Exchange Infrastructure, URL: http://help.sap.com/content/documentation/netweaver/cont_netw_xi.htm [Zugriff am 26.01.2004].

[15] *IBM*: WebSphere InterChange Server and WebSphere Business Integration Toolset documentation, URL: ftp://ftp.software.ibm.com/software/websphere/integration/ wicserver/library/doc/wics422/wics422core.zip [Zugriff am 30.01.2004].

[16] *P. Bernstein et al.:* A Vision for Management of Complex Models, URL: ftp://ftp.research.microsoft.com/pub/tr/tr-2000-53.pdf [Zugriff am 05.04.2004].

[17] *J. Sztipanovits, G. Karsai:* Model-Integrated Computing, URL: http://csdl.computer.org/dl/mags/co/1997/04/r4110.pdf [Zugriff am 05.04.2004].

[18] *A. Ledeczi et al.:* The Generic Modeling Environment, URL: http://www.isis.vanderbilt.edu/Projects/gme/GME2000Overview.pdf [Zugriff am 05.04.2004].

[19] *Fraunhofer IAO:* Projekthomepage „Integration Engineering", URL: http://www.sw-management.iao.fraunhofer.de/projekte/sm_05.html#IE [Zugriff am 05.04.2004].

1

2

3

1 Clemens Dorda hat an der Universität Stuttgart Informatik studiert und sein Studium 2002 mit dem Diplom abgeschlossen. Er ist seitdem wissenschaftlicher Mitarbeiter von Prof. Dr.-Ing. habil. Bernhard Mitschang am Institut für Parallele und Verteilte Systeme (IPVS) der Universität Stuttgart. Adresse: Universität Stuttgart, Institut für Parallele und Verteilte Systeme (IPVS) Abteilung Anwendersoftware, Universitätsstr. 38, 70569 Stuttgart, E-Mail: Clemens.Dorda@informatik.uni-stuttgart.de

2 Dr. Hans-Peter Steiert hat Informatik an der Universität Kaiserslautern studiert und 2001 in der Arbeitsgruppe von Prof. Härder an der Universität Kaiserslautern promoviert. Seit 2001 ist er als wissenschaftlicher Mitarbeiter bei DaimlerChrysler im Forschungsbereich *IT for Engineering* tätig. Adresse: DaimlerChrysler AG, Research and Technology, RIC/ED, Postfach 2360, 89013 Um, E-Mail: Hans-Peter.Steiert@DaimlerChrysler.com

3 Dr. Jürgen Sellentin hat Informatik an der Universität Kaiserslautern studiert und 1999 in der Arbeitsgruppe von Prof. Mitschang an der Universität Stuttgart promoviert. Seit 1996 ist er Mitarbeiter von DaimlerChrysler, wo er bis 2003 das Team *Integrationsarchitekturen* innerhalb des Forschungsbereiches *IT for Engineering* geleitet hat. Seit Sommer 2003 ist er verantwortlich für das *Test & Integration Lab* im Bereich *Technology Integration & Platform Development* des zentralen DaimlerChrysler IT Managements.
Adresse: DaimlerChrysler AG, Technology Integration (ITI/TP), HPC Z354, 70546 Stuttgart, E-Mail: juergen.sellentin@daimlerchrysler.com