Architekturbasierter Entwurf komplexer Softwaresysteme

Peter Boße, Markus Iwig
Technische Universität Ilmenau
Fakultät für Informatik und Automatisierungstechnik
peter.bosse@rz.tu-ilmenau.de
markus.iwig@informatik.tu-ilmenau.de

Zusammenfassung

Der vorliegende Beitrag befaßt sich mit der Problematik eines architekturbasierten Softwareentwicklungsprozesses. Kernziel ist hierbei eine Verbesserung der Wiederverwendung und Wartbarkeit des Systems und seiner Bestandteile. Der Beitrag schildert Prinzipien der Architektur- und Komponentenspezifikation. Darauf aufbauend wird ein möglicher Prozeß zur Entwicklung solcher Architekturen aufgezeigt. Abschließend werden die Vorteile dieser Vorgehensweise, aber auch ihre Risiken und Probleme geschildert und ein Ausblick auf künftige Arbeiten des Gebietes gegeben.

1 Einleitung

Softwarearchitektur wird als eine Menge von Sichten auf ein Softwaresystem verstanden, in deren Vordergrund die Betrachtung der strukturellen Aspekte des Laufzeitsystems (z.B. Komposition von Komponenten, Protokolle für Kommunikation, Synchronisation und Datenzugriff, physische Verteilung) steht [1].

Jede View legt den Fokus auf einen ganz bestimmten Aspekt des zu entwickelnden Systems. In der Regel erfolgt die Betrachtung dieser Aspekte auf einem höheren Abstraktionsniveau als bei der ‘herkömmlichen’ Entwicklung, z.B. mit objektorientierten Analyse- und Designmethoden, deren Fokus vor allem auf der Modellierung der Klassenstruktur liegt [2]. Ziel der Softwarearchitekturbeschreibung ist es, Systeme als Kombination miteinander kommunizierender Komponenten auszudrücken. Dabei besteht heute der Anspruch, daß diese Systeme in heterogenen Umgebungen verteilt arbeiten, wobei die Implementationssprachen der Systemteile verschieden sein können.

Der Begriff der Softwarekomponente wird derzeit stark strapaziert, wobei keine einheitliche Semantik zugrunde liegt. Übereinstimmend ist, daß es sich bei Komponenten um Softwareeinheiten handelt, die bestimmte Funktionalitäten über eine definierte Schnittstelle bereitstellen und sich durch einfache Wiederverwendbarkeit auszeichnen. Im Kontext von Softwarearchitekturen bezeichnen wir mit Komponenten die funktionalen Einheiten des entsprechenden Laufzeitsystems. Jede dieser Softwarekomponenten stellt eine Instanz eines Komponententyps dar, die über definierte Verbindungen mit Instanzen anderer Typen kommuniziert. Man betrachtet dabei meist die oberen Granularitätsstufen; eine hierarchische Verfeinerung ermöglicht die Erschließung feingranularer Ebenen.

Aufgrund der immer weiter wachsenden Komplexität bedarf es neuer Mittel, die innerhalb einer Architekturbeschreibung den Systemaufbau explizit zum Ausdruck bringen [3]. Im folgenden werden Prinzipien der Spezifikation von Softwarekomponenten und der Architekturbeschreibung näher vorgestellt.

2 Mechanismen der Architekturbeschreibung

2.1 Spezifikation von Softwarekomponenten

Die Spezifikation von Komponententypen umfaßt zum einen die Definition der Schnittstelle einer Komponente zu ihrer Umgebung (Interface), zum anderen muß das Verhalten der Komponente beschrieben werden, welches bei Benutzung dieses Interfaces durch andere Komponenten ausgelöst werden soll. Für die Interfacebeschreibung können verschiedene Prinzipien angewendet werden. Basierend auf den Ausführungen in [4] sollen zwei davon im folgenden näher betrachtet werden.

In den Interfaces der sogenannten *Object-Connection-Architecture* (OCA) werden nur die durch eine Komponente bereitgestellten (exportierten) Dienste spezifiziert. Aus der Sicht der potentiellen Klienten ist diese Art der Schnittstellenbeschreibung völlig ausreichend. Benötigt eine Komponente Dienste aus ihrer Umgebung, so werden diese unter Verwendung von Referenzen auf die Diensterbringer bereitgestellt. Eine Beschreibung dieser Referenzen erfolgt erst auf Codeniveau (Realisierung). Die Verbindungen zwischen den Komponenten werden also ausschließlich durch die entsprechende Implementierung bestimmt. Das Prinzip der Object-Connection-Architecture ist in den OO-Sprachen zu finden und dient als Grundlage sowohl der CORBA- als auch der COM-*Interface Definition Language* (IDL). Wie im folgenden gezeigt wird, mangelt es diesen Sprachen im Kontext von Systembeschreibungen an den nötigen Modellierungselementen und der Ausdruckskraft, Architekturen explizit zu charakterisieren.

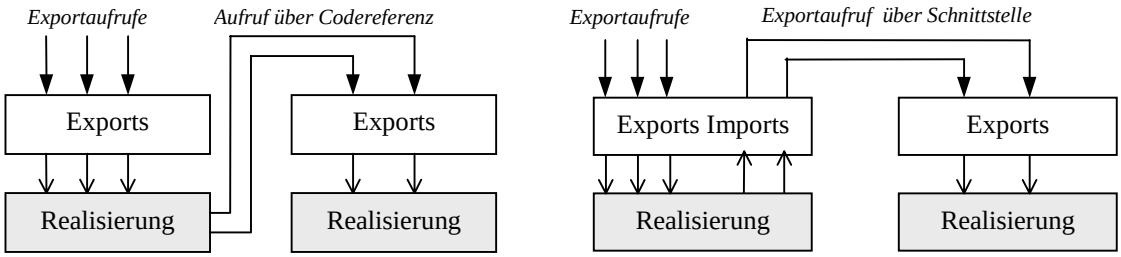


Abbildung 2-1: OCA vs. ICA

Eine Erweiterung der Object-Connection-Architecture stellt die *Interface-Connection-Architecture* (ICA) dar. Hier werden neben den exportierten Diensten auch die benötigten (importierten) Dienste im Interface spezifiziert. Die Beschreibung der Verbindungen zwischen den Systemkomponenten findet jetzt auf der Ebene der Schnittstellen durch eine explizite Definition der Systemkonfiguration statt. Dies bewirkt die Trennung von Architektur und Implementierung und hat den Vorteil, daß bereits vor Beginn der Implementierung von Komponenten die Struktur des Systems modelliert und in einem ersten Schritt getestet und bewertet werden kann. Eine Umsetzung der Interface-Connection-Architecture ist in den meisten *Architecture Description Languages* (ADL's) zu finden [5].

Aus Sicht der Implementation wird durch die Anwendung ICA-konformer Strukturen ein hoher Grad an Unabhängigkeit zwischen Komponenten erreicht. Die Realisierungen 'hinter' den Interfaces haben keine Referenzen aufeinander und sind somit vollständig gekapselt. Die Implementation einer solchen Komponente kann leichter durch interfacekonforme

Realisierungen ersetzt bzw. an entsprechender Stelle in anderen Systemen wiederverwendet werden.

Entsprechend der vorgestellten Prinzipien beinhaltet die Komponententypdefinition die Spezifikation der angebotenen Dienste (Exports), der benötigten Dienste (Imports) sowie die Beschreibung des Verhaltens, das bei Aufruf eines Exports ausgelöst werden soll. Ausgangspunkt für die Verhaltensbeschreibung ist das Ziel, sich ausschließlich auf die von außen sichtbaren, im Interface definierten Merkmale zu beschränken. Bei der Spezifikation von Zuständen und Zustandsübergängen stehen demnach keine komponenteninternen Konstrukte (z.B. Variablen) zur Verfügung. Die Verwendung von *Black-Box*-Spezifikationen erweist sich in diesem Zusammenhang als vorteilhaft [6]. Black-Boxes haben genau den Anspruch, daß nur äußere Größen zur Beschreibung ihres Verhaltens, d.h. der Reaktion auf einen bestimmten Stimulus, benutzt werden dürfen. Ein Stimulus kann das Eintreten eines bestimmten Zustandes im umgebenden System (Ereignis) oder der Aufruf eines Exports sein. Zur Charakterisierung des Zustandes wird die Historie, d.h. die Vorgeschichte der aufgetretenen Stimuli verwendet.

2.2 Architekturbeschreibung

Die Architektur eines Softwaresystems läßt sich unter verschiedenen Gesichtspunkten betrachten. Die zum Verständnis des Systems wichtigste Sicht ist unserer Meinung nach die Struktur der Komponenten zur Laufzeit und ihre Kommunikation untereinander. Aufgrund der hohen Komplexität solcher Strukturen ist es notwendig, Abstraktionsstufen unterschiedlicher Komponentengranularität einzuführen. Die Elemente der verschiedenen Schichten sind Entsprechungen (im objektorientierten Sinne Instanziierungen) der oben beschriebenen Komponententypen. Auf der obersten, abstraktesten Ebene befinden sich die Hauptkomponenten des zu spezifizierenden Systems. Diese können in der nächsten Ebene durch die Verschaltung von Subkomponenten zu einer Architektur verfeinert werden.

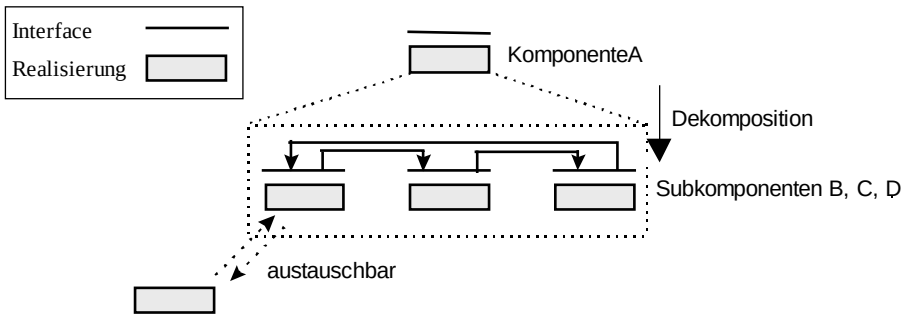


Abbildung 2-2: hierarchische Verfeinerung

Dabei erfolgt ein Mapping der Verantwortlichkeiten der Oberkomponente auf die Teilfunktionen der entsprechenden Unterkomponenten. Die eigentliche Struktur wird erst durch die Definition der Konnektionen sichtbar. Konnektionen sind gerichtete binäre Verbindungen zwischen Komponenten, wobei die Exports der Komponente, die die Rolle des Servers spielt, mit den Imports der die Clientrolle einnehmenden Komponente verbunden werden. Dabei können sich die Namen der Operationen, die durch Export bzw. Import repräsentiert werden, unterscheiden, die Typen von Parametern und Rückgabewerten müssen dagegen kompatibel sein. Dieser

hierarchische Entwurf kann über beliebig viele Stufen fortgesetzt werden. Auf der untersten Ebene befinden sich die Komponenten, für die keine sinnvolle Aufteilung in Subkomponenten mehr gefunden werden kann. Für diese wird die Realisierung in Form einer Klassenbeschreibung dokumentiert.

Im Ergebnis dieser Entwurfsschritte entsteht eine Spezifikation/Beschreibung, die schon vor der Implementierung eine umfangreiche Analyse und Bewertung des Systems erlaubt. Mit entsprechenden Tools ist die Spezifikation als erster Prototyp der zu erstellenden Software ausführbar.

Der für die Beschreibung der Architektur notwendige Aufwand ist nicht zu unterschätzen und muß unter wirtschaftlichen Gesichtspunkten kritisch betrachtet werden. Die Informationen, die in den Spezifikationen enthalten sind, lassen sich allerdings sehr gut für die Generierung von Softwarecode benutzen. Zur automatischen Codeerzeugung bedarf es programmiersprachenspezifischer Abbildungsregeln (Mapping). Durch diese Automatismen relativiert sich der zusätzliche Spezifikationsaufwand erheblich.

Neben der Sicht auf die logische Laufzeitstruktur des Systems können in einer Architekturbeschreibung weitere wichtige Aspekte dokumentiert werden. In einer Verteilungssicht können die Art einer Konnektion (local oder remote) zwischen zwei Komponenten sowie eine Zuordnung von Komponenten zu den Knoten einer Hardwarekonfiguration (Rechnernetzwerk) festgeschrieben werden. Entsprechend dieser Verteilung ist unter Benutzung der Informationen aus der Interfacespezifikation die automatische Generierung von IDL-Interfaces (CORBA) und unter Verwendung einer konkreten ORB-Implementation die Generierung von 'ORB-Anbindungs'-Code möglich (z.B. Nutzung des Naming-Service).

Alle Entwurfsentscheidungen können mit Hilfe von Rationals dokumentiert werden. Hierzu gehören die Begründungen für die getroffenen Entscheidungen und eine Angabe von Alternativen. Daneben erlauben Rationals die Dokumentation der Systemevolution und sogenannten architectural drift. Dieser Aspekt einer Softwaredokumentation ist nicht zu unterschätzen, da in der heutigen Praxis oftmals das Entwurfswissen nur in den Köpfen der Entwickler steckt und somit unter Umständen für eine spätere Wartung oder Systemmodifikation verloren gehen kann.

Anforderungen und Besonderheiten, die bei der Entwicklung und späteren Implementierung der Architektur beachtet werden müssen, werden mittels Constraints festgeschrieben. Eine Formalisierung dieser Restriktionen ermöglicht eine automatische Kontrolle und Anzeige von Constraint-Verletzungen.

Weitere wichtige Sichten einer Softwarearchitektur sind die Pattern-View und eine Development-View. In der Pattern-Sicht können charakteristische Architekturstile [1] bzw. Architekturmuster [7] definiert und die Komponenten den entsprechenden Rollen zugeordnet werden. Die Development-View zeigt Aspekte der Architektur aus Sicht der Entwicklungsumgebung auf. Hier können Elemente der Architektur- und Komponentenbeschreibung auf Objekte der Implementations- und Codemanagementebene (Klassen, Applikationen, Konfigurationen etc.) abgebildet werden.

Ausgehend von der Erkenntnis, daß die gebräuchlichen Analyse- und Designmethoden auf dieser Ebene nicht ausreichend sind, wurden in den letzten Jahren speziell einige Sprachen entwickelt, die Architekturen von Softwaresystemen explizit unter Verwendung der oben beschriebenen Interfaces spezifizieren [5]. Weitere Quellen für diese Architekturbeschreibungssprachen sind modulbasierten Ansätze sowie die Megaprogrammierung [8].

3 Architekturbasierter Entwicklungsprozeß

Dieses Kapitel dient der Einführung eines Ansatzes für einen architekturbasierten Softwareentwicklungsprozeß. Es werden die einzelnen Schritte kurz diskutiert und der Zusammenhang mit benachbarten Phasen hergestellt.

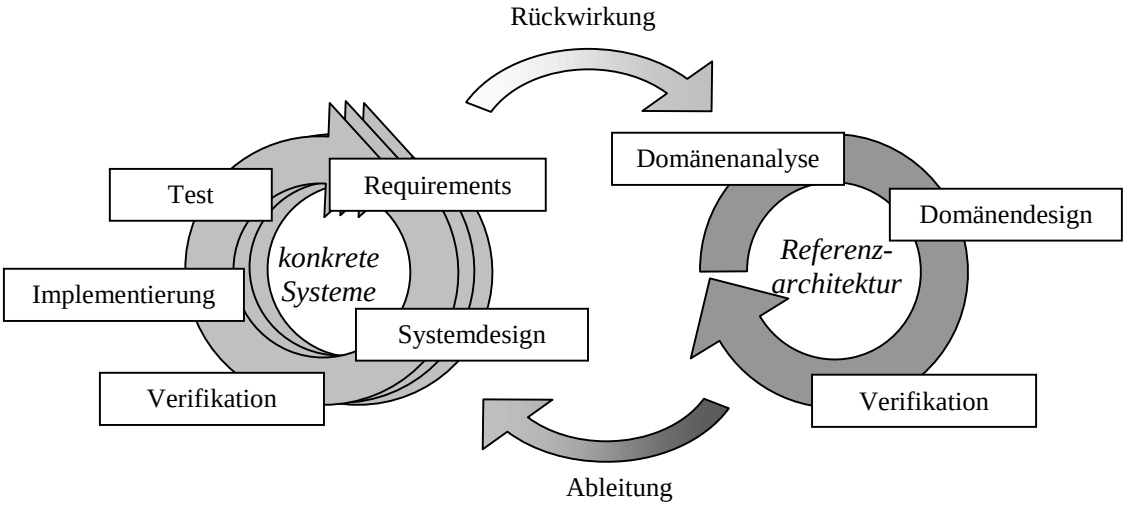


Abbildung 3-1: Prozeß

In Abbildung 3-1 sind zwei gekoppelte Prozesse dargestellt: einer für die evolutionäre Entwicklung einer sogenannten Referenzarchitektur und ein zweiter für die Ableitung konkreter Anwendungen.

Die Domänenanalyse beschäftigt sich mit der Identifizierung verschiedener Domänen, dem Sammeln domänenbezogener Daten sowie deren Organisation und Repräsentation. Unter *Domäne* wird ein fachlicher Bereich mit abgrenzbarer Funktionalität verstanden. Während der Analysephase wird die Zieldomäne genau eingegrenzt. Weiterhin werden die Gemeinsamkeiten und Unterschiede verschiedener Applikationen des gewählten Gebietes exakt herausgearbeitet und in einer angemessenen Art und Weise dargestellt. Hauptanliegen aller Domänenanalysetechniken ist die Verbesserung des Verständnisses einer Domäne durch das Erfassen aller relevanten Informationen und deren Darstellung in verschiedenen Modellen.

In der Phase des Domänen-Designs steht der Entwurf von Softwaresystemen in Form von Architekturen im Mittelpunkt. Oftmals kann für eine Familie von fachlich verwandten Systemen eine Referenzarchitektur extrahiert werden. Sie bildet die Ergebnisse aus der Domänenanalyse auf abstrakte Softwarekomponenten ab. Dabei muß diese Abbildung nicht 1:1 erfolgen, das heißt einer Komponente können mehrere Funktionen zugeordnet werden bzw. mehrere Komponenten sind für die Abdeckung einer komplexen Funktionalität verantwortlich. Eine Referenzarchitektur manifestiert auf abstrakte Weise das Entwurfswissen für einen fachlich abgegrenzten Bereich (Domäne) in Form allgemeingültiger Komponenten einschließlich ihrer Interaktionen. Sie kann als Wurzel einer Architekturfamilie verwandter Systeme bezeichnet werden. Es entsteht eine Schablone, auf deren Basis an konkrete Anforderungen angepaßte Architekturen abgeleitet

werden können. Die Entwicklung und Beschreibung domänenspezifischer Referenzarchitekturen adressiert schwerpunktmäßig Softwarewiederverwendung auf der Entwurfsebene.

Die Verifikation weist die Korrektheit der Entwurfsergebnisse auf Architekturniveau nach. Im Ergebnis stehen für die gewählte Domäne allgemeingültige und überprüfte Komponenten-Spezifikationen zur Verfügung.

Für konkrete Softwaresysteme müssen zunächst deren Anforderungen in einer Requirementsanalyse ermittelt werden. Diese können in verschiedenster Form, z.B. Use-Case-Model, notiert werden. Unter Verwendung der allgemeingültigen Komponentenspezifikationen der Referenzarchitetktur erfolgt die Ableitung einer konkreten Softwarearchitektur. Diese kann unter Beachtung der Requirements um zusätzliche Komponenten erweitert werden.

Eine anschließende Verifikation erbringt den Nachweis für einen korrekten Systementwurf. Für die wiederverwendeten Bestandteile der Referenzarchitektur erübrigt sich eine erneute Überprüfung.

In der nachfolgenden Phase der Systemimplementation wird die in der Architektur festgelegte Struktur auf eine gewählte Programmiersprache abgebildet. Eine architekturkonforme Programmierung verbessert die Chancen, wiederverwendbare Komponenten zu schaffen. Die technische Grundlage für die Realisierung komponentenbasierter Systeme können z.B. die durch CORBA und (D)COM definierten Objekt-/Komponentenmodelle bilden. Sie bieten die Möglichkeit, unter Benutzung einer entsprechenden Infrastruktur binäre lauffähige Softwareeinheiten über ein oder mehrere Interfaces unabhängig von Implementationsaspekten anzusprechen.

Im Ergebnis dieser Schritte entstehen ein oder mehrere konkrete Softwaresysteme der Domäne. Diese können ihrerseits auf die Referenzarchitektur Einfluß nehmen, z.B. indem weitere allgemeingültige Komponenten identifiziert und in die Referenzarchitektur integriert werden. Alle diese Architekturen können und werden in der Regel evolutionären Veränderungen unterliegen. Eine sorgfältige Wartung sowohl der Spezifikationen als auch der Systeme ist deshalb von essentieller Bedeutung (Traceability).

4 Nutzen, Probleme und Ausblick

Die in die Objektorientierung gesetzten Hoffnungen in Bezug auf Wiederverwendung und Wartbarkeit sind nicht in dem Maße erfüllt worden, wie dies anfänglich propagiert wurde. Ein wesentlicher Grund hierfür ist in der mangelnden Qualität entsprechender Software-beschreibungen zu sehen [9]. Dies gilt nicht nur für interne Entwicklungen, sondern in erster Linie auch für den Kauf von wiederverwendbaren Komponenten auf dem Markt. Um wiederzuverwenden, muß man wiederverwendbare Komponenten suchen und verstehen. Insbesondere die Abhängigkeiten zu anderen Komponenten sind hierbei häufig unklar. Die Spezifikation der von einer Komponente benötigten Dienste (Imports) ist in diesem Zusammenhang daher unerläßlich. Erst durch eine genaue Dokumentation der Rahmenbedingungen und des Verhaltens einer Komponente wird eine gute Basis für Wiederverwendung von Software geschaffen. Neben der Unterstützung der Codewiederverwendung ermöglicht eine solche Vorgehensweise Wiederverwendung auf der Ebene des Systemdesigns. Referenzarchitekturen auf Entwurfsebene können in diesem Kontext mit Frameworks auf Quelltextebene verglichen werden.

Der zusätzliche Aufwand für die Spezifikation sollte sich durch die verbesserten Möglichkeiten der Wiederverwendung und Wartung der Softwarebausteine schnell auszahlen. So kann die Softwareerstellung auf Dauer schneller werden, weil passende (und getestete!)

Komponenten schneller gefunden und eingesetzt werden können. Die Verwendung solcher ausgereiften Komponenten führt zu einer höheren Qualität der Softwareprodukte und einem entsprechenden Ansehen am Markt. Daneben sinken die Wartungskosten, da Fehler leichter gefunden und getrennt vom restlichen System behoben werden können. Aufgrund des komponentenorientierten Aufbaus kann auch flexibler auf fachliche Änderungen und Kundenwünsche reagiert werden. Diese hohe Flexibilität schlägt sich auch in kürzeren Entwicklungs- und Markteinführungszeiten der Produkte nieder. Die Möglichkeit der Generierung von Code bzw. IDL-Interfaces wurde bereits diskutiert.

Die Beschreibung von Architekturen und Komponenten dient des weiteren als Grundlage für die Kommunikation der verschiedenen Projektmitglieder (Softwareentwickler, Manager, Kunden etc.). Durch eine gemeinsame 'Sprache' können Anforderungen klar formuliert, verschiedene Alternativen aufgezeigt, Änderungen und Lösungen verständlich dargelegt werden.

Im obigen Abschnitt wurden eine Reihe von Vorteilen beim Einsatz eines architekturbasierten bzw. komponentenorientierten Entwicklungsprozesses vor allem in Bezug auf Wiederverwendung und Wartung vorgestellt. So offenkundig die Notwendigkeit einer solchen Vorgehensweise auch erscheinen mag, wichtig ist die volle Unterstützung seitens des Managements. Wiederverwendung im großen Stil kann nur funktionieren, wenn entsprechende Voraussetzungen seitens der Unternehmensleitung geschaffen werden. Wiederverwendung muß zu einem Bestandteil der Organisation, Unternehmenskultur und des Entwicklungsprozesses werden [9].

Die Akzeptanz und der Einsatz derartiger softwaretechnischer Mittel seitens der Praxis wird in großem Maße durch das Verhältnis von Aufwand und Nutzen bestimmt. Abgesehen von der Komplexität der zusätzlich zu erlernenden Architekturbeschreibungssprache spielt vor allem die Bereitstellung von ausgereiften Umgebungen/Tools für die Anwender eine wesentliche Rolle. Neben der Spezifikation der Systeme und ihrer Komponenten muß ebenfalls eine systematische Verwaltung der Fülle an Dokumentationen und Informationen gewährleistet werden, um eine sinnvolle Handhabung zu ermöglichen.

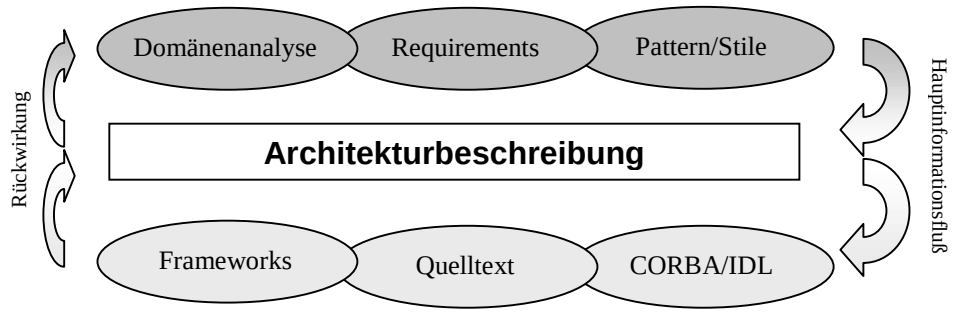


Abbildung 4-1: Stellung der Architekturbeschreibung

Im Projekt OSVA (Objektorientierte Softwarewiederverwendung in Verteilten Architekturen) wurden Werkzeuge realisiert, die die Anwendung wesentlicher Aspekte der vorgestellten Architekturkonzepte unterstützen. Grundlage hierfür ist ein in der objektorientierten Sprache Smalltalk umgesetztes Metamodell, welches die Elemente einer formalen Architektur- und Komponentenbeschreibung bereitstellt. Diese Art der Repräsentation erlaubt eine

- hohe Flexibilität bezüglich Erweiterbarkeit und Modifizierbarkeit des Beschreibungsmittels,
- vollständige Integration in die vertraute Entwicklungsumgebung,
- Nutzung der Vorteile der Objektorientierung,
- gute Möglichkeit der Simulation und
- einfachere Überführung in Quellcode der Programmiersprache.

Es existieren Tools für die Definition von Komponententypen (Schnittstelle und Verhalten), die Spezifikation des zugrundeliegenden Datenmodells sowie für die Beschreibung der Systemstruktur als Architekturen. Neben der Strukturbeschreibung werden eine Patternsicht und Verteilungssicht bereitgestellt. Bestehende Klassen-Modelle aus dem Werkzeug Rational Rose können über eine Schnittstelle in die Architekturbeschreibung eingelesen werden. Anhand der Informationen aus dem Metamodell hat der Nutzer weiterhin die Möglichkeit, CORBA-konforme IDL-Interfaces zu generieren. Innerhalb der Smalltalk-Entwicklungsumgebung existieren verschiedene Werkzeuge zum Reengineering bestehender Applikationen, die Architekturinformationen aus der anwendungsspezifischen Klassenhierarchie extrahiert.

Wichtig für die Akzeptanz seitens der Anwender ist die Berücksichtigung existierender Standards. Vorangegangene Untersuchungen der *Unified Modeling Language* (UML) haben gezeigt, daß derzeit noch beträchtliche Lücken bei der Unterstützung von Architekturbeschreibungen bestehen [10]. Hier sind entsprechende Erweiterungen zu erwarten. Auch die Bemühungen der *Object Management Group* (OMG) zur Standardisierung eines Komponentenmodells sowie die Entwicklung der Enterprise JavaBeans zeigen, daß dieses Thema in der industriellen Softwareentwicklung an Bedeutung zunimmt.

5 Literatur

[1] M. Shaw, D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996

[2] Ahmed Abd-el-Shafy Abd-Allah. Composing Heterogeneous Software Architectures. PhD, University of Southern California. August 1996

[3] L. Bass, P. Clements, R. Kazman. *Software Architecture In Practice*. Addison-Wesley, 1998

[4] D. Luckham. Three Concepts of System Architecture. http://pavg.stanford.edu/rapide/rapide-pubs.html, 1995

[5] P. Clements. A Survey of Architecture Description Languages. *SoftwareArchitecture Technology Initiative.* http://www.sei.cmu.edu/architecture/projects.html, 1996

[6] D. L. Parnas, Y. Wang. The Trace Assertion Method of Module Interface Specification. *CRL Report No.244*, McMaster University, Hamilton, Ontario, Canada, 1992

[7] F. Buschmann, R. Meunier, H. Rhonert, P. Sommerlad, M. Stahl. *Pattern-Oriented Software Architecture: A System of Patterns.* West Sussex, England, Wiley, 1996

[8] M. Nagl. *Softwaretechnik: Methodisches Programmieren im Großen*. Springer Verlag, 1990

[9] M. Holl, C. Mitterbauer. Wiederverwendung: Visionen und Illusionen. *OBJEKTSpektrum,* 98(2): 62-69

[10] J. E. Robbins, N. Medvidovic, D. Redmiles, D. Rosenblum. Integrating Architecture Description Languages with a Standard Design Method. University of California, 1997