Von der Prozedur zum Web Service: Retrospektive und Perspektive

Johannes Sametinger, Alois Stritzinger

Institut für Wirtschaftsinformatik
Johannes Kepler Universität Linz,
A-4040 Linz
{johannes.sametinger|alois.stritzinger}@jku.at
http://www.se.jku.at/{sametinger|stritzinger}

Abstract. Die Entwicklungsgeschichte der Sprachmittel zur Beschreibung und Strukturierung von Software war in der vergangenen 25 Jahren von einer hohen Dynamik geprägt. Dieser Artikel legt die Gründe für das Entstehen immer mächtigerer Konzepte dar und behandelt Web Services als vorläufigen Endpunkt dieser Entwicklung. Dabei wird deutlich, dass jedes Konzept nach wie vor eine eigenständige Daseinsberechtigung besitzt, und es der Verantwortung des Software-Ingenieurs obliegt, die jeweils besten Mittel zu wählen.

1 Einleitung

Softwaresysteme sind so komplex, dass sie nur selten von einzelnen Programmierern überschaut werden können. Die Beherrschung dieser Komplexität ist das Hauptproblem bei der Entwicklung großer Softwaresysteme. Zur Bewältigung der Komplexität werden Systeme in kleinere Einheiten zerlegt. Die Komplexität des Entwicklungsprozesses führt zu einer Zerlegung in Teilaufgaben, Phasen oder Iterationsschritte.

Gegen Ende der 70er Jahre, also in der Bronzezeit der Informatik, stellte die Prozedur bzw. Funktion *das* Strukturierungsmittel in der Softwareentwicklung dar. Dazu passende Entwurfsmethoden wie die funktionale Zerlegung und die schrittweise Verfeinerung wurden damals auch in der Arbeitsgruppe Rechenberg studiert und angewandt. Vorläufer des Prozedurkonzepts waren Makros und Subroutinen. Diese hatten aber bereits in den 60er Jahren mit Programmiersprachen wie Algol und PL/I an Bedeutung verloren. Die Anfang der 80er Jahre aufkommenden Mikrocomputer brachten den Durchbruch für die prozeduralen Sprachen Pascal und C.

Die Wurzeln der objektorientierten Programmierung reichen bis in die 60er Jahre zurück. Durchsetzen konnten sich die Konzepte aber erst in den 80er Jahren. Sprachen wie Smalltalk und C++ sollten ihre prozeduralen Vorgänger Pascal und C bald verdrängen. Heute sind auch objektorientierte Sprachen bereits ein alter Hut. Das Internet hat sich etabliert und auch als Software-Plattform bewährt. Viele Informationssysteme werden heute als Web-Anwendungen realisiert. Derzeit sind Web Services in Mode. In diesem Artikel werden wir die geschichtliche Entwicklung von den Prozeduren bis zu den Web Services beschreiben und einen Blick in die nächste Zukunft wagen.

2 Prozeduren / Strukturierte Programmierung

Bei der strukturierten Programmierung wird ein komplexes Gesamtsystem in kleine Teilfunktionen gegliedert. Dabei wird das Ziel verfolgt, ein übersichtliches und aus möglichst unabhängigen Funktionen zusammengesetztes, also gut strukturiertes Programm zu erhalten. Solche Einheiten können mit Prozeduren und Funktionen realisiert werden. Im Entwurfsprozess werden nach dem Top Down-Prinzip klar abgegrenzte Teilaufgaben identifiziert und jeweils in einer abgeschlossenen Prozedur realisiert. Ein gesamtes Programm kann so aus den einzelnen Prozeduren Schritt für Schritt aufgebaut werden. Die Prozeduren selbst nutzen sich gegenseitig, indem sie sich aufrufen und Daten austauschen.

Der wesentliche Fortschritt von Prozeduren gegenüber Makros und Subroutinen ist die Trennung von Schnittstelle und Implementierung und die daraus resultierende Abstraktionsmöglichkeit. Ein weiterer Vorteil von Prozeduren ist die kellerartige Verwaltung der lokalen Daten mit Aktivierungssätzen. Dies ermöglicht die Realisierung von rekursiven Prozeduren, die für viele Probleme einfache Lösungen erlauben. Prozeduren sind aber auch mit Schwächen behaftet:

- Zustände, Seiteneffekte
  Prozeduren sind idealerweise zustandslos und seiteneffektfrei. Dies ist allerdings oft nicht möglich; die Zustände können nur ungeschützt global gespeichert werden. Schwer durchschaubare Abhängigkeiten sind meist die Folge.
- Sprachabhängigkeit
  Die Aufruf- und Parameterübergabemechanismen von Prozeduren sind sprach- und/oder systemabhängig. Softwareteile in unterschiedlichen Sprachen lassen sich schwer kombinieren und müssen an jedes Rechnersystem angepasst werden.
- Nebenläufige Prozesse
  Bei der Verwendung von nebenläufigen Prozessen können schwerwiegende Fehler auftreten, wenn Prozeduren globale Daten manipulieren, d.h. Prozeduren sind ohne zusätzliche Mechanismen nicht mehrprozessfähig (reentrant).
- Verteilung
  Prozeduren können nicht ohne spezielle Unterstützung über Maschinengrenzen hinweg (remote) aufgerufen werden. Mit Prozeduren alleine lassen sich daher keine verteilten Systeme realisieren.

Pascal [9] ist die klassische Programmiersprache für prozedural strukturierte Programmierung.

3 Module / Modulare Programmierung

Prozedurale Programme sind häufig monolithisch. Sind sie aus mehreren Programmteilen zusammengesetzt, dann wird die Korrektheit von Prozeduraufrufen und Datenzugriffen eines Programmteils auf einen anderen vom Compiler nicht überprüft. Dies hat vielfach zur Folge, dass globale Zustände und Prozeduren stark miteinander gekoppelt sind und Fehlerkorrekturen, Änderungen und Erweiterungen in einem Pro-

grammteil sich oft auch auf andere Programmteile auswirken können. Die Grundidee der modularen Programmierung besteht darin, Programmteile zu definieren, die Daten und Funktionen bzw. Prozeduren zu abgeschlossenen Programmeinheiten zusammenfassen. Ein wichtiger Leitgedanke ist dabei das Geheimnisprinzip (Information Hiding). Anwender eines Moduls benutzen nur dessen Schnittstelle und interessieren sich nicht für die Details seiner Implementierung.

Das Geheimnisprinzip lässt sich mittels Datenkapselung umsetzen. Die Datenkapselung in Form des Moduls mit exakt definierten Schnittstellen und kontrollierten Zugriffsrechten verbessert die Wartbarkeit und die Wiederverwendbarkeit von Softwaresystemen beträchtlich. Modulare Softwaresysteme bestehen somit aus einer mehr oder weniger großen Zahl von Modulen mit präzise festgeschriebenen Schnittstellen und Abhängigkeiten.

Durch das Modulkonzept werden einige, aber bei weitem nicht alle von der strukturierten Programmierung bekannten Probleme gelöst. Zwar können Zustände geschützt, und Seiteneffekte eingedämmt werden. Die Sprachabhängigkeit bleibt aber weiterhin bestehen. Ebenso gibt es noch keine klare Lösung für nebenläufige Prozesse und verteilte Systeme. Typische Vertreter für modulare Programmiersprachen sind Modula-2 [15] und Ada [1].

4 Klassen / Objektorientierte Programmierung

Bei der objektorientierte Programmierung spielen Klassen von Objekten eine entscheidende Rolle. Klassen beschreiben Objekte in denen Zustände (Daten) und Verhalten (Prozeduren bzw. Methoden) zusammengefasst sind. Im Unterschied zu Modulen lassen sich Klassen als Typen verwenden. Zusätzlich können Klassen von anderen Klassen erben, wodurch Typhierarchien möglich werden. Zur Laufzeit besteht ein Softwaresystem aus vielen solchen Objekten, die miteinander über das Versenden von Nachrichten kommunizieren und kooperieren. Mittels dynamischer Bindung führt das Versenden einer Nachricht zum Aufruf der jeweils passenden Methode. Klassen stellen typischerweise Abstraktionen von realen oder fiktiven Gegenständen dar. Im Gegensatz dazu sind bei der prozedural strukturierten Programmierung die Abstraktionen Aufgaben oder Funktionen.

Die objektorientierte Programmierung kam in den 80er Jahren durch Sprachen wie Smalltalk [10], ObjectPascal [24] und etwas später C++ [23] in Mode. Die Ursprünge der objektorientierten Programmierung liegen aber schon bei Simula [2] in den 60er Jahren. Methoden brachten zwei neuartige Eigenschaften:

- Zustände
  Methoden sind vielfach zustandsbehaftet. Sie hängen sozusagen an einem wohldefinierten und gekapselten Zustand.
- Polymorphie
  Die konkrete Funktionalität eines Aufrufs ist vom Typ, d.h. der Klasse des Empfängerobjektes abhängig.

Damit lassen sich Softwaresysteme flexibler gestalten. Die Wiederverwendbarkeit von Software-Bausteinen wurde im großen Stil machbar. Klassenbibliotheken mit tausenden Klassen fanden Einzug in den Entwickleralltag. Zeitgemäße objektorientierte Sprachen gestatten es ferner, Anweisungsfolgen als kritische Regionen zu markieren, um gleichzeitiges Betreten durch mehrere Prozesse zu verhindern. Es bleiben trotz dieser Vorzüge wesentliche Schwächen erhalten:

- Sprachabhängigkeit
  Die Aufruf- und Parameterübergabemechanismen sind wie bei Prozeduren sprach- und/oder systemabhängig.
- Verteilungsunfähigkeit
  Methoden können nicht über Maschinengrenzen hinweg (remote) aufgerufen werden. Damit lassen sich auch mit objektorientierten Konzepten alleine keine verteilten Systeme realisieren.

Vertreter objektorientierter Sprachen sind Smalltalk [10], C++ [16], Java [12], C# [6].

5 Komponenten / Komponentenbasierte Programmierung

Parallel zur Verbreitung der objektorientierten Programmierung entstand das Konzept von Softwarekomponenten, wobei in den Anfängen die Ziele dieser Entwicklung diffus waren. Komponentenmodelle stellten entweder Programmiersprachenabhängigkeit in den Vordergrund, z.B. Microsoft COM [4], oder visuelle Kombinierbarkeit, z.B. ActiveX Controls [3], Java Beans [13].

Im Moment etablieren sich Softwarekomponenten eher serverseitig, d.h. sie werden hauptsächlich zur Realisierung der Geschäftslogik in Client/Server-Systemen eingesetzt. Die Unabhängigkeitsbestrebungen der Komponenten lassen sich allerdings nur teilweise verwirklichen. So sind etwa Enterprise Java Beans (EJBs) [11] auf einen sogenannten EJB-Applikation Server mit virtueller Java-Maschine angewiesen und die nächste Microsoft Serverkomponenten-Technologie wird auf der DotNet-Umgebung [5] basieren. Trotz einer gewissen Schwerfälligkeit die den Serverkomponenten anhaftet, bieten sie Vorteile, die insbesondere bei komplexen Transaktionssystemen zu Tragen kommen.

Komponenten sind mehrprozessfähig, weil sie entweder zustandslos sind oder die Synchronisierung über eine Datenbank erfolgt. Weiters sind sie auch von ferne ausführbar. Damit gestatten komponentenbasierte Systeme die Entwicklung großer, verteilter Mehrbenutzersysteme mit garantiertem, d.h. sicherem Transaktionsverhalten der persistenten Daten. Dazu kommen noch zusätzliche Dienste wie automatische Lastverteilung, Wiederanlauf nach Systemausfällen u.a.m. Serverseitige Komponentenplattformen genügen den Anforderungen zeitgemäßer betrieblicher Anwendungen schon in hohem Maße. Es verbleibt die Plattformabhängigkeit und damit auch die Schwierigkeit der Integration mit informationstechnischen Altlasten (legacy systems) als erhebliches Manko der Komponententechnologie.

Typische Vertreter für Komponententechnologien sind Enterprise JavaBeans [11], COM/ActiveX [4] und Corba [8].

6 Web Services / Serviceorientierte Programmierung

Das Bedürfnis, heterogene Systeme über große Entfernungen zu koppeln, führte schon in den 80er Jahren zur Bildung der Object Management Group (OMG) [7], mit dem primären Ziel, einen Standard zur plattformübergreifenden Kommunikation zu entwickeln und zu definieren. Schwierigkeiten durch verschiedene unternehmenspolitische Ausrichtungen, unklare und überambitionierte Zielsetzungen und aufwändige oder ungenügende Implementierungen ließen aber die Verbreitung von Corba Ende der 90er Jahre stagnieren.

Die stürmische Entwicklung des Internets und insbesondere des World Wide Webs prägten die 90er Jahre. In den Anfängen war das Web ein ausschließlich passives, multimediales Hypertext-Informationssystem, das primär statische Dokumente an Webbrowser auf Anforderung verteilte. Allgegenwärtigkeit, Vertrautheit und einfache Bedienbarkeit der Webbrowser führten zu immer anspruchsvolleren Web-Anwendungen, die konventionellen Client/Server-Systemen in ihrer Funktionalität beinahe ebenbürtig waren. Serverseitige Software war erforderlich, um Benutzer zu identifizieren, mit Eingabedaten behaftete Anfragen abzuarbeiten und personalisierte Antwortseiten zu retournieren. Es entstanden Konzepte wie Serverside-Skripting, programmierbare Server Pages und Servlets [14], die alle das Hypertext Transfer Protokoll (HTTP) [21] als Transportformat und Universal Resource Locators (URLs) [22] als Adressierungsmechanismus verwendeten.

Von URL-adressierbaren Services war es nur mehr ein kleiner Schritt zu XML-basierten, entfernten Aufrufen (Remote Procedure Call), die ausschließlich mit standardisierten Formaten und Protokollen abgewickelt und beschrieben werden. Dieses Konzept ist unter dem Namen Web Service [19] bekannt. Web Services sind Softwareeinheiten, die mit anderen Systemen über das Internet interagieren. Dabei werden ausschließlich offene Standards wie eXtensible Markup Language (XML) [17], Simple Object Access Protocol (SOAP) [18] und Web Service Description Language (WSDL) [20] verwendet.

Web Services gestatten den Aufbau einfacher verteilter Systeme, deren Kommunikation und deren Schnittstellenbeschreibungen völlig plattformunabhängig auf strukturiertem Text allein beruhen. Sie bieten das Potenzial, aus dem verteilten Informationssystem WWW ein verteiltes System, also globale, verteilte, heterogene Rechnersysteme zu entwickeln. Dies ist analog zu modularen Programmen auf einem einzelnen Computersystem mit privaten Daten und öffentlichen Operationen darauf. Künftig sind mit Web Services neuartige Anwendungen denk- und machbar: So lassen sich damit etwa verteilte Client/Server-Systeme für technische oder betriebliche Anwendungen realisieren. Elektronische Marktplätze, die mit Anbietern und Nachfragern über Web Services kommunizieren sind ebenfalls machbar. Es ist vorstellbar, dass viele herkömmliche Web-Sites künftig auch service-orientierte Informationszugänge anbieten, die eine programmierte, sprich automatische Abfrage und Auswertung ermöglichen. Diese Szenarien sind aber noch Visionen.

Konkret können Web Services schon heute gewinnbringend zur Integration heterogener Systeme innerhalb von Organisationen oder zwischen wenigen Teilnehmern, z.B. Business2Business, zum Einsatz kommen. Web Services sind allerdings derzeit ebenfalls noch mit Mängeln und Schwächen behaftet:

- Kommunikationssicherheit, Authentizität
  Das Internet ist ein öffentliches Netz und daher inhärent unsicher. Öffentliche Web Services bedürfen öffentlicher Verschlüsselungsstandards, um Übertragungssicherheit, sowie digitale Signaturen, um Senderauthentizität zu gewährleisten. Erst der jüngste SOAP-Standard legt diesbezügliche Standards fest.
- Leistungsfähigkeit
  Nachrichten von Web Services sind in XML-Texten codiert. Alle transportierten Nachrichteninhalte wie Parameter, Ergebnisse und Fehlermeldungen müssen unmittelbar vor der Übertragung in das Textformat transformiert und nach dem Empfang wiederum analysiert und in interne Daten rücktransformiert werden. Zusätzlich ist XML-Text eine relativ ineffiziente, „geschwätzige" Codierung. Mit einer Standardisierung von dynamischen Kompressions- und Dekompressionsverfahren könnte Bandbreite gespart werden.
- Höhere Datentypen und –strukturen
  Web Service Daten liegen entweder in atomarer Form oder als einfache Verbundstrukturen oder Felder vor. Anspruchsvollere Strukturen sind derzeit nicht definiert. Die Abbildung der externen Daten auf interne Strukturen und vice versa obliegt allein den Implementierungen von Service und Client.
- Transaktionale Integrität
  Der Web Service Standard kennt derzeit keine Unterstützung für Transaktionen, d.h. Änderungen, die garantiert vollständig oder gar nicht wirksam sind. Man kann bei der Implementierung der Services Transaktionsunterstützung nutzen bzw. vorsehen. Es ist aber zu erwarten, dass auch auf der Kommunikationsebene Mechanismen für Transaktionen, Fehlerverhalten, Wiederanlauf, etc. benötigt werden.
- Lastverteilung, Serviceverteilung, Verteiltes Speichermanagement
  In SOAP und WSDL sind noch viele nützliche Dienste, die zum Beispiel aus Corba und anderen verteilten Systemen bekannt sind, nicht berücksichtigt.
- Versions- und Konfigurationsmanagement, u.a.m.

Ob und wann entfernte, XML-basierte Aufrufe mit komplexen Datenstrukturen eine softwaretechnische Selbstverständlichkeit werden, ist schwer vorhersehbar. Die Entwicklung wird längst nicht so explosiv erfolgen, wie die Verbreitung des World Wide Web, da viele anwendungsspezifische Datenformate und Verhaltenssemantiken mühsam ausverhandelt und implementiert werden müssen. Trotz dieser Schwierigkeiten stellen Web Services einen vielversprechenden vorläufigen Endpunkt in der Entwicklung der Software-Strukturierungskonzepte dar.

7 Resümee

Die jeweils späteren, mächtigeren Strukturierungskonzepte können und sollen frühere nicht ersetzen, da zum Teil enorme Laufzeiteinbußen in Kauf genommen werden müssen. Die beschriebenen Konzepte sollen einander ergänzen nicht verdrängen. Für einfache Aufgaben kann eine Prozedur nach wie vor ein perfektes Strukturierungsmittel sein. Das Können des Software Ingenieurs drückt sich unter anderem darin aus, das er für eine gegebene Aufgabe, das am besten geeignete Mittel auszuwählen versteht.

Es hat Jahrzehnte gedauert, bis sich die Erkenntnis allgemein durchgesetzt hat, dass universelle Kommunikation der Austausch von strukturiertem Text ist – nichts sonst. XML, SOAP und WSDL sind nur moderne Varianten von strukturierten Datenströmen wie sie auch durch Grammatiken beschrieben werden können.

Prof. Rechenberg hat schon vor 25 Jahren formale Sprachen sowie die Analyse und Synthese von strukturierten Datenströmen als wichtiges Gebiet der Informatik erkannt und interessante Forschungsbeiträge dazu geleistet. Sein Wissen hat er in didaktisch hervorragender Weise an seine Studenten weitergegeben. Wir danken ihm dafür.

Verweise

1. Ada Home: The Web Site for Ada . http://www.adahome.com/
2. Holmevik Jan Rune. The History of Simula. http://java.sun.com/people/jag/SimulaHistory.html
3. Microsoft Corporation. ActiveX Controls. http://msdn.microsoft.com/library/default.asp?url=/workshop/components/activex/activex_node_entry.asp
4. Microsoft Corporation. COM – Component Object Model. http://www.microsoft.com/com/
5. Microsoft Corporation. .NET Framework. http://msdn.microsoft.com/netframework/
6. Microsoft Corporation. Visual C# .NET. http://msdn.microsoft.com/vcsharp/
7. OMG – Object Management Group. http://www.omg.org/
8. OMG – Object Management Group: Corba – Common Object Request Broker Architecture. http://www.omg.org/corba/
9. Pascal Central. http://www.pascal-central.com/
10. Smalltalk. http://www.smalltalk.org/
11. Sun Microsystems, Inc. Enterprise JavaBeans. http://java.sun.com/products/ejb/
12. Sun Microsystems, Inc. Java. http://java.sun.com/
13. Sun Microsystems, Inc. JavaBeans. http://java.sun.com/products/javabeans/
14. Sun Microsystems, Inc. Servlets. http://java.sun.com/products/servlet/
15. The Modula-2 Webring. http://www.modulaware.com/m2wr/
16. The C++ Resources Network. http://www.cplusplus.com/
17. W3C – World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/XML/
18. W3C – World Wide Web Consortium. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/
19. W3C – World Wide Web Consortium. Web Services Activity. http://www.w3.org/2002/ws/
20. W3C – World Wide Web Consortium. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl
21. W3C – World Wide Web Consortium. HTTP – Hypertext Transfer Protocol. http://www.w3.org/Protocols/
22. W3C – World Wide Web Consortium. Naming and Addressing: URIs, URLs, ... http://www.w3.org/Addressing/
23. Wikipedia. C++. http://de.wikipedia.org/wiki/cplusplus
24. Wikipedia. ObjectPascal. http://de.wikipedia.org/wiki/ObjectPascal