Implementierung einer anwendungsspezifischen formalen Beschreibungssprache durch einen Application-Framework

Dipl.-Ing. Peter Wendorff, Global-Finanz Unternehmensgruppe, Bonn.

Im vorliegenden Artikel wird ein Software-Projekt (CFG-Projekt=Computer-Finanz-Gutachten) bei einem Allfinanz-Dienstleister behandelt, bei dem das Hauptproblem in der Spezifizierung von Algorithmen lag. Es wird die Konstruktion einer anwendungsspezifischen formalen Beschreibungssprache gezeigt, die mittels einer Framework-Architektur in der Programmiersprache C++ realisiert wurde. Bei der Konstruktion dieser Beschreibungssprache wurde die semi-formelle Notation der Fachabteilung aufgenommen und systematisiert sowie formalisiert. Das Ergebnis war eine formale Spezifikationssprache auf dem Abstraktionsniveau des Anwendungsgebiets, die eine wesentliche Hilfe im Rahmen des Requirements-Engineerings darstellte.

1 Nutzen formaler Spezifikationen

"I believe the hard part of building software to be the specification, design, and testing of [...] [the] conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared with the conceptual errors in most systems." [Bro87, S. 11].

Aus zwei Gründen ist der Einsatz formaler Methoden im Requirements-Engineering besonders vorteilhaft [Hall90, S. 19]:

a) "They work largely by making you think very hard about the system you propose to build."
b) "They can help clients understand what they are buying."

Der weiteren Verbreitung formaler Methoden im Requirements-Engineering in der Praxis steht jedoch die kryptische Syntax und komplexe Semantik der gängigen formalen Spezifikationssprachen (VDM, Z) im Wege. Eine Muß-Anforderung an eine formale Spezifikationssprache für ein kommerzielles Projekt ist dagegen die Unterstützung der Begriffswelt des Anwendungsgebiets. Eine weitere ist die Unterstützung von Prototyping im realen Systemumfeld (bspw. Integration realer Dialogmasken, Daten, Reports usw. bei der Anforderungsspezifikation und -definition).

2 Problemstellung und Lösungsansatz

Das CFG-Projekt dient der computergestützten Analyse der Versorgungs- und Vermögenssituation von Kunden. Es umfaßt u.a. ein in C++ programmiertes Auswertungsprogramm, das alle erforderlichen fachlichen Algorithmen des relevanten Anwendungsgebiets Versicherungswirtschaft / Sozialversicherungsrecht / Steuerrecht enthält.

Das wesentliche Problem in diesem Projekt war die Spezifikation und Verfeinerung der fachlichen Algorithmen, denn das Anwendungsgebiet ist extrem komplex und enthält zahllose Spezialfälle. Ein weiteres Problem stellte die Forderung nach sofort einsetzbaren Teilergebnissen dar, um einen in Betrieb befindlichen, völlig unwartbaren und instabilen Minimal-Prototyp abzulösen. Es wurde folgende Lösungsstrategie entworfen:

a) Inkrementelles Vorgehensmodell, zur schnellen Realisierung der Kernfunktionen und späteren (planmäßigen) Integration von Spezialfällen.
b) Prototyping, um die Auswirkungen von Änderungen und Erweiterungen des Fachkonzepts sofort (unter Echtbedingungen) validieren zu können.
c) Application-Framework-Architektur, für Code-Reuse auf der Ebene einer einheitlichen Software-Architektur [BeDe92].
d) Konstruktion einer applikationsspezifischen formalen Beschreibungssprache auf dem Abstraktionsniveau des Fachkonzepts [Dav82].

Application-Framework-Technologien gewinnen vor dem Hintergrund zunehmenden Kostendrucks steigende Aufmerksamkeit [BeDe92]. Das grundlegende Konzept wird in [WeLi95] wie folgt erläutert:

"Ein Framework besteht aus einer Menge vorgefertigter Bausteine. Es unterstützt nicht nur die Wiederverwendung von Code, sondern vor allem auch die Wiederverwendung des Designs. Die Klassen, aus denen ein Framework besteht, wurden bereits von den Entwicklern miteinander 'verdrahtet'. Das fachspezifische Wissen über Zusammenhänge und Abläufe der Klassen wurde also bereits im Framework hinterlegt und braucht nur noch aufgerufen zu werden."

Ein Framework stellt die softwaretechnologischen Grundlagen für eine Applikations-Domäne, d.h. für eine Klasse von ähnlichen Programmen eines Anwendungsgebiets dar. Diese Applikations-Domäne ist im vorliegenden Fall durch folgende Charakteristika gekennzeichnet:

a) Der zentrale Ausgangspunkt einer Analyse ist stets ein Kunde, der als wirtschaftliche Interessengemeinschaft beliebig vieler Personen definiert wird.

b) Die Analyse der Daten erfolgt im Rahmen von Kapiteln, also abgeschlossenen Einheiten. Beispiele für Kapitel sind: "Altersversorgung", "Krankentagegeld", "Liquidität", "Steuerliche Situation", "Fremdgenutzte Immobilien" usw.
c) Es werden zugekaufte Fremdmodule verwendet (Rentenberechnung, Steuerberechnung, Tarifberechnung usw.), die wegen des großen Datenvolumens über breite, komplexe Schnittstellen verfügen und umfassende Fehlerdiagnose-Routinen erfordern.
d) Die Analyse von Daten innerhalb eines Kapitels kann kundenbezogen, personenbezogen oder beides sein.
e) Die Analyseergebnisse werden über Datenbanktabellen (Drucktabellen) an einen Berichtsgenerator übergeben. Dabei sind drei Betriebsmodi möglich (personenbezogen, kundenbezogen oder auswertungsbezogen).

Das Programm-Skelett des Frameworks enthält "Slots", in denen die applikationsspezifischen Softwareanteile quasi als "Einschübe" realisiert werden. Ein Kapitel der Applikations-Domäne verkörpert dabei genau einen Slot. Im Rahmen des CFG-Projekts sollte der Slot zu einem Kapitel die "essentiellen Algorithmen dieses Anwendungsgebiets" repräsentieren, und zwar so, daß sie von der Fachabteilung (mit minimaler Schulung) lesbar sind.

Ebenso war vorgesehen, jeden applikationsspezifischen Fremdmodul über einen Slot zu integrieren.

3 Vorgaben der Fachabteilung

Die Vorgaben der Fachabteilung bestanden aus Data-Dictionary, Algorithmen, Vorlagen für die vom System zu erstellenden Reports sowie diversen Schlüssel-Systematiken für das Konfigurations-Management. Im weiteren wird exemplarisch das Kapitel KT ("Krankentagegeld") für pflichtversicherte Arbeitnehmer betrachtet. Zunächst ist im folgenden ein Ausschnitt des entsprechenden Reports angegeben (Die Bezeichner in eckigen Klammern werden beim Ausdruck durch die berechneten Werte in den Drucktabellen ersetzt):

Durchschnittliches Monatsnettoeinkommen:	[vGehNetto]
Krankengeld nach Fortfall der Lohnfortzahlung:	[vMtlGesetzlKrankengeld]
abzüglich des Arbeitnehmeranteils zur Rentenversicherung:	[vBeitragGrvMtl]
abzüglich des Arbeitnehmeranteils zur Arbeitslosenversicherung:	[vBeitragAlvMtl]
zuzüglich mtl. Leistung aus privaten Zusatzversicherungen:	[vMtlPkvVersKrankentagegeld]
verbleibt eine Auszahlung des Krankengeldes von:	[vMtlNettoKrankengeld]

Im weiteren werden nun die Algorithmen betrachtet. Für die Absicherung eines GKV-pflichtversicherten Arbeitnehmers stellten sie sich in der ursprünglichen Version wie folgt dar:

```
vekst/stkl/emtlsteuerbrutto=Berechnung der mtl. Lohnsteuer an Hand des mtl. steuerpflichtigen
  Bruttoeinkommens, der Steuerklassenwahl und eines evt. vorliegenden Steuerfreibetrages
  (Datensatzübergabe an Steuer-Modul und Rücksendung dieses Wertes)

vkst(vekst/stkl/emtlsteuerbrutto)=
  Berechnung der mtl. Kirchensteuer auf Basis des vekst/stkl/emtlsteuerbrutto

vgehnetto=Berechnung des durchschnittlichen monatlichen Nettogehalts=
  (((emtlsteuerbrutto-vekst/stkl/emtlsteuerbrutto- vkst(vekst/stkl/emtlsteuerbrutto)) X eanzgeh
  : 12) - [...]) + [...]

Schlüsseltyp 1 - 1 - 1   [GKV-pflichtversicherter Arbeitnehmer]
Version 13

10 c Eingabewerte zur Berufschlüsselwahl
[...]
1101 vbeitragalvmtl = vbeitragalvjahr : 12
1102 vbeitraggrvmtl = vbeitraggrvjahr : 12
1103 vmtlgesetzlkrankengeld = emtlsozversbrutto X 80 : 100
1104 IF vmtlgesetzlkrankengeld > vmtlnetto THEN vmtlgesetzlkrankengeld = vmtlnetto
[...]
1111 vmtlpkvverskrankentagegeld = epkvverskrankentagegeld X 30
1112 vmtlnettokrankengeld = vmtlgesetzlkrankengeld - vbeiraggrvmtl - vbeiragalvmtl
1113 vmtlfehlbetragzumnetto = vgehnetto - vmtlnettokrankengeld
1114 IF vmtlfehlbetragzumnetto = 0 Print 06 - 1 - 3
1115 IF vmtlfehlbetragzumnetto < 0 THEN [...] Print 06 - 1 - 2
1116 Print 06 - 1 - 1
```

4 Analyse der Vorgaben der Fachabteilung

Eine wichtige Erfahrung des Projekts bestand darin, daß gerade die Fehler in den Vorgaben eine wichtige Enscheidungshilfe bei der Definition der Aufgabenteilung von Framework und der zugehörigen Slots war. Vor diesem Hintergrund werden nun einige Details bzw. Fehler der Vorgaben der Fachabteilung analysiert (Eine allgemeine Klassifizierung von Fehlerarten in Spezifikationen findet sich z.B. in [May85]).

Unvollständigkeit 1: Grundlage des Projekts ist ein relationales Datenmodell in dritter Normalform. Die fachlichen Vorgaben enthalten jedoch keine SQL-Statements oder sonstige Spezifikation der Datenbankzugriffe. Dies zeigt, wie nicht anders zu

erwarten, daß die Ebene des logischen, relationalen Datenmodells für die Spezifikation der Algorithmen inadäquat ist. Der Framework hat daher die Aufgabe, eine transparente, objektorientierte Zugriffsschicht für das Datenmodell, auf der Abstraktionsebene des Fachkonzepts zu schaffen.

Unvollständigkeit 2: Es wird nicht deutlich, daß u.U. mehrere Personen zu einem Kunden gehören können, deren Situation analysiert werden muß, wobei diese beispielsweise zusammen oder getrennt veranlagt sein können. Diese Problematik betrifft alle Kapitel bzw. Fremdmodule und muß daher im Framework realisiert werden.

Unvollständigkeit 3: Der Wert *vbeitraggrvjahr* wird aufgrund der gesetzlichen Sozialversicherungs-Beitragssätze errechnet. In den Vorgaben (Zeile 1102) wird keine Rücksicht auf Sonderfälle (z.B. pflichtversicherte, angestellte Ärzte, die einem Versorgungswerk angehören) genommen. Dies bestätigt die Erfahrungstatsache, daß die Fachabteilung implizit Plausibilisierungen voraussetzt, weil dieses Wissen für Fachleute Allgemeingut darstellt. Der Framework sollte dieser Erkenntnis Rechnung tragen. Insbesondere ist hierbei die Unterstützung einer konsistenten und vollständigen Fehlerbehandlung in allen Applikationen, die auf dem Framework basieren, wünschenswert.

Unvollständigkeit 4: *vekst/stkl/emtlsteuerbrutto* bezieht sich auf einen im Projekt eingesetzten Steuer-Berechnungsmodul. Die Steuerklassenwahl ist in der Spezifikation nicht weiter ausgeführt. Dies ist ein Hinweis darauf, daß diese Problematik innerhalb eines Analysekapitels (wie z.B. hier "KT") aus fachlicher Sicht ein irrelevantes Detail darstellt. Deshalb ist durch den Framework eine transparente Schnittstelle zu den Ergebnissen von Fremdmodulen zur Verfügung zu stellen.

Unvollständigkeit 5: Es existiert kein Hinweis darauf, was der Befehl *Print* bedeutet. Anscheinend soll er den Ausdruck einer kompletten Analyse bewirken, also einer Seite mit etlichen berechneten Werten. Welche das sind, ist jedoch nicht ersichtlich. Hier wird deutlich, daß die Kommunikation mit dem verwendeten Berichtsgenerator ein vom Framework bereitzustellender Service ist.

Im vorliegenden Projekt war es sehr hilfreich, Unvollständigkeiten in Vorgaben der Fachabteilung nicht als frustrierende Arbeits-Hemmnisse auffassen, sondern als Ansatzpunkt, um die Abstraktionsebene des Fachkonzepts zu identifizieren. Dies war die Voraussetzung für eine projektspezifische formale Spezifikationssprache auf dieser Abstraktionsebene.

5 Formale Spezifizierung der Algorithmen

Zunächst wird ein Teil des Programmcodes gezeigt, der mit den oben gezeigten Vorgaben korrespondiert, jedoch an mehreren Stellen korrigiert bzw. erweitert wurde.

```
BOOL CKapKt::BearbeitungEinerPerson(
  CPerson& personP, CPerson& personPartnerVonP)
BEGIN_FUNC
[...]
  IF (personP.IstGkvPflichtig() ODER personP.IstGkvFreiwillig())
  THEN
      vMaxMtlGesetzlKrankengeld =MIN(personP.SozVersPflMtlBrutto(), personP.KvBbgMtl())*0.8;
      vMtlGesetzlKrankengeld =MIN(vMaxMtlGesetzlKrankengeld, personP.vGehNetto());
      PRINT("vMtlGesetzlKrankengeld", vMtlGesetzlKrankengeld);

      // Bei Krankheit müssen die GRV- und ALV-Beiträge
      // auf vMtlGesetzlKrankengeld bezogen werden
      vBeitragGrvMtlBeiKrankheit=vMtlGesetzlKrankengeld*personP.GrvBeitrSatz()/2;
      vBeitragAlvMtlBeiKrankheit =vMtlGesetzlKrankengeld*personP.AlvBeitrSatz()/2;
      PRINT("vBeitragGrvMtl", vBeitragGrvMtlBeiKrankheit);
      PRINT("vBeitregAlvMtl", vBeitragAlvMtlBeiKrankheit);
     [...]
  END_IF
[...]
END_FUNC
```

Im weiteren werden die Konsequenzen aus der Analyse der in Kapitel 4 genannten "Unvollständigkeiten" dargestellt.

Konsequenz aus Unvollständigkeit 1: Das relationale Datenmodell wird vom Framework gekapselt, so daß ein konzeptuelles Datenmodell auf der Abstraktionsebene des Anwendungsgebiets zur Verfügung steht. Beispielsweise hat sich im vorliegenden Projekt das Konzept bewährt, alle personenbezogenen Daten zu einer Person in je einem Objekt einer Klasse "CPerson" zu speichern. Diese Erfahrung aus einem konkreten Projekt wurde auf die gesamte Applikations-Domäne verallgemeinert. Hierbei läßt sich die Framework-Technologie besonders gut erläutern: Das Konzept der Klasse "CPerson" ist konstituierender Bestandteil des Frameworks. Die Daten jedoch, die in dieser Klasse gespeichert werden, sind applikationsspezifisch. Der Anwendungsentwickler, der eine spezielle Applikation auf Basis des Frameworks programmiert, erstellt durch Definition der Attribute der Klasse "CPerson" eine applikationsspezifische, formale Beschreibungssprache [Dav82]. Hierzu werden vordefinierte Methoden der Klasse "CPerson" des Frameworks applikationsspezifisch überschrieben.

Konsequenz aus Unvollständigkeit 2: Der Framework verfügt über eine Klasse CKapitel, die eine abstrakte Basisklasse darstellt. Jedes Kapitel der Analyse wird als Slot des Frameworks, d.h. als eine von CKapitel abgeleitete Klasse (z.B. "CKapKt" für das Kapitel "Krankentagegeld") dargestellt. Die abgeleiteten Klassen ererben dabei 5 virtuelle Methoden, die bei Bedarf überladen werden. Eine dieser 5 Methoden ist die hier gezeigte Funktion *BearbeitungEinerPerson(...)*. Sie wird vom Framework pro Person einmal aufgerufen. Mittels der beiden übergebenen Objektreferenzen stellt der Framework die Personendaten aus der Datenbank kontextabhängig und plausibilisiert zur Verfügung.

Konsequenz aus Unvollständigkeit 3: Der Framework stellt vordefinierte Methoden zur Plausibilisierung von Daten bereit, die ebenfalls bei Bedarf überschrieben werden können. Attribute wie z.B. *personP.GrvBeitrSatz()* werden dadurch im folgenden vom Framework plausibilisiert bereitgestellt (also z.B. personP.GrvBeitrSatz()=0, falls personP ein Arzt mit Versorgungswerk-Mitgliedschaft ist).

Konsequenz aus Unvollständigkeit 4: Die Steuerberechnung wird mit all ihren Einflußfaktoren in einem, auf die Integration von Fremdmodulen spezialisierten, Slot des Frameworks durchgeführt. Innerhalb eines Kapitels sind lediglich die Endergebnisse von Fremdmodulen interessant, die als Attribute der Klasse "CPerson" realisiert sind.

Konsequenz aus Unvollständigkeit 5: Interessant ist noch die Realisierung des Befehls "PRINT". Er macht das Zusammenspiel von C++-Applikation und Report-Generator mit einer dazwischenliegenden Datenbankschnittstelle für den Betrachter weitgehend transparent. Dem Leser des Programmcodes werden nur zwei Informationen gegeben: Zum einen der Feldbezeichner im Report und zum anderen die im Programm verwendetete Variable. Die dahinterliegende, relativ aufwendige Schnittstelle und deren Steuerlogik bleibt völlig verborgen.

6 Erfahrungen und Ergebnisse

a) Der Programmcode in den Kapitel-Slots des Application-Frameworks ist für alle Projektbeteiligten problemlos lesbar und wird als Arbeitsgrundlage voll akzeptiert. Mißverständnisse können damit sicher vermieden werden. Gleichzeitig sind alle Vorteile einer formalen Spezifikation gegeben. Inkonsistenzen und Unvollständigkeiten werden bereits in der Analysephase entdeckt. Prototypen können i.d.R. im Gespräch mit den Fachverantwortlichen am Rechner erstellt werden, um auf beiden Seiten das Verständnis für das Problem zu vertiefen und fachliche Vorgaben sofort formal zu spezifizieren. Beim Prototyping werden lediglich Bausteine des Frameworks mit Grundkonstrukten von C++, wie z.B. Variablen, algebraischen Operatoren, Kontrollstrukturen usw. kombiniert. Die so erstellten Prototypen können unmittelbar mit Echtdaten und den tatsächlichen Reports verwendet werden. Die in [Dav1982] geschilderten positiven Erfahrungen bezüglich der "Lesbarkeit" von Programmcode, bei einer derartigen Vorgehensweise, wurden im vorliegenden Projekt uneingeschränkt bestätigt.

b) Die objektorientierte Zugriffsschicht für das relationale Datenmodell hat sich in der Praxis voll bewährt. Allerdings ist sie z.Zt. nur eingeschränkt und behelfsweise implementiert, da hier die im wissenschaftlichen Schrifttum beschriebenen Ergebnisse verwertet werden sollen und deren Evaluierung noch nicht abgeschlossen ist.

c) Es ergeben sich Arbeitserleichterungen dadurch, daß Routinetätigkeiten, wie z.B. Pflege von Algorithmen und Data-Dictionary toolgestützt, auf Grundlage des aktuellen Programmcodes durchgeführt werden können, und Inkonsistenzen folglich unmöglich sind.

d) Die entwickelte Framework-Architektur hat sich als sehr geeignet erwiesen, Kernfunktionen schnell zu realisieren. Viele Details können im Framework zunächst kursorisch realisiert werden, ohne daß die u.U. sehr komplexen Sachverhalte in den Slots in Erscheinung treten. Dadurch kann der Programmcode in den Slots auf das Wesentliche konzentriert werden.

e) Eine Methode ist nur so praxistauglich wie die sie unterstützenden Werkzeuge. Im beschriebenen Projekt hat bisher die Toolunterstützung einer modernen Entwicklungsumgebung (GUI, Versionsverwaltung, Debugger, datenbankgestützter Browser, Profiler...) ausgereicht. Klar ist jedoch, daß insbesondere bei der Wiederverwendung der Framework-Architektur spezialisierte Tools verwendet werden müssen. In diesem Rahmen ist ein konsequentes Konfigurationsmanagement erforderlich, um die Konsistenz des Frameworks über alle Applikationen sicherzustellen.

Literaturhinweise

[BeDe92]: Beck, R.P.; Desai, S.R. et al.; "Architectures for Large-Scale Reuse"; AT & T Technical Journal (Nov./Dec. 1992), pp. 34-45.

[Bro87]: Brooks, F. P.; "No Silver Bullet - Essence and Accidents of Software Engineering"; IEEE Computer, (Apr. 1987), pp. 10-19.

[Dav82]: Davis, A.M.; "The Design of a Family of Application-Oriented Requirements Languages"; IEEE Computer (May 1982), pp. 21-28.

[Hall90]: Hall, A.; "Seven Myths on Formal Methods"; IEEE Software (Sep. 1990), pp. 11-19.

[Mey85]: Meyer, B.; "On Formalism in Specifications"; IEEE Software (Jan. 1985), pp. 6-26.

[WeLi95]: Weyerhäuser, M; Lipps, P.; "Anwendungsumgebungen sind flexibler als Betriebssysteme"; Computerwoche (Nr. 35, Sep. 1995), S. 15-18.