Software - Entwicklung
mit
SDL und Ada

Gerhard Beck
Bosch Telecom GmbH, Backnang
gerhard_beck@bk.bosch.de

Zusammenfassung

Die Integration ‚standardisierter Sprachen‘ für Projektorganisation, SW-Entwurf und Implementierung wird als praxiserprobter methodischer Ansatz zur SW-Entwicklung beschrieben. Der gezielte Einsatz phasenspezifischer Ausdrucksmittel, die Überwindung der Sprachgrenzen und zukünftige Entwicklungspotentiale werden aufgezeigt

Stichworte:
Methodik SW-Entwicklung, V-Modell, SDL, Ada

1 Einleitung

1.1 Motivation

Der vorliegende Artikel entstand anläßlich eines Workshops ’Erfahrungen mit Ada’ der GI-Fachgruppe 2.1.5 Ada am 17.April 97 im Forschungszentrum Karlsruhe -Technik und Umwelt.

Der Artikel berichtet über langjährige praktische Projekterfahrungen mit der Programmiersprache Ada bei Bosch Telecom. Insbesondere wird über eine im Telekommunikationsumfeld entwickelte Methodik berichtet, die durch Verknüpfung der Spezifikationssprache SDL (Specification and Description Language) und der Programmiersprache Ada geprägt ist. Das V-Modell liefert hierbei den organisatorischen Rahmen und das Basis-Vokabular zur Projektabwicklung.

1.2 Technisches Umfeld

Folgende Kenngrößen charakterisieren das technische Umfeld:

- Applikation: Server für Key Distribution (online)
- Embedded Computer System (ECS)
- Entwicklungsumgebung:
  - SPARC/Solaris (native)
  - VMEbus, 68xxx, pSOS^{+m} (cross)
- ca. 150.000 SLOC

1.3 Begriffsbestimmung

V-Modell[1]

Entwicklungsstandard, regelt die Vorgehensweise bei SW-Entwicklungen in den Bereichen Software-Erstellung, Qualitätssicherung, Konfigurations- und Projektmanagement. Das V-Modell stellt den einheitlichen SW-Entwicklungsstandard der deutschen Bundesbehörden dar, der zunehmend im industriellen Bereich als Firmenstandard übernommen wird.

SDL88[2], **SDL92**[3]

Standardisierte (ISO, CCITT) Spezifikationssprache zur Beschreibung der Struktur und des Verhaltens von Realzeit-, verteilten - und/oder interaktiven Systemen. SDL basiert auf asynchron kommunizierenden Prozessen, die als erweiterte endliche Automaten (extended finite state machine, EFSM) modelliert werden. SDL verfügt neben textueller Notation über Sprachelemente zur graphischen Verarbeitung des Systementwurfs.

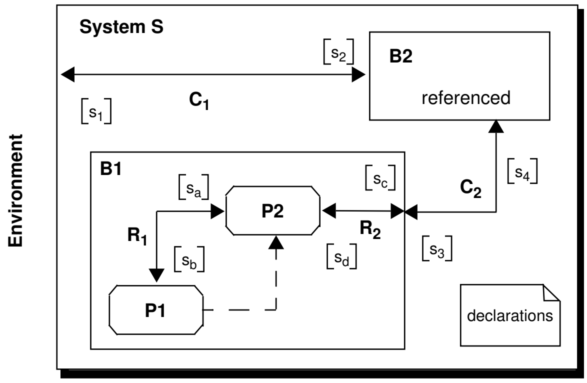

Bild 1: SDL Systemarchitektur (static structure)

SDL92 stellt eine evolutionäre Weiterentwicklung von SDL88 mit objektorientierten Inhalten dar. Bild 1 illustriert beispielhaft die Darstellung eines Systemdiagramms zur Beschreibung der *Systemarchitektur*. Das System (System S) wird in Blöcke (Block B) strukturiert, die mittels Kanäle (Channel C) untereinander und mit der Umgebung (Environment) verbunden sein können. Auf der untersten Strukturie-

1. Vorgehensmodell, Planung und Durchführung von IT-Vorhaben, Der Bundesminister des Innern, August 92
2. Recommendation Z100, CCITT, Specification and Description Language (SDL), 1988
3. Revised Recommendation Z.100, CCITT, Specification and Description Language (SDL), 1992

rungsebene enthalten Blöcke mindestens einen Prozeß (Process P), der über Signalverbindungen (SignalRoute R) mit anderen Prozessen oder Kanälen verbunden ist. Signalverbindungen und Kanäle transportieren Signale (Signal s_i) einschließlich optionaler Signalparameter. Die zum Signal-Transport erforderliche Zeit ist unbestimmt, die Signalreihenfolge bleibt erhalten. Bild 2 zeigt beispielhaft Elemente eines Prozeßdiagramms zur Beschreibung des *Systemverhaltens*.

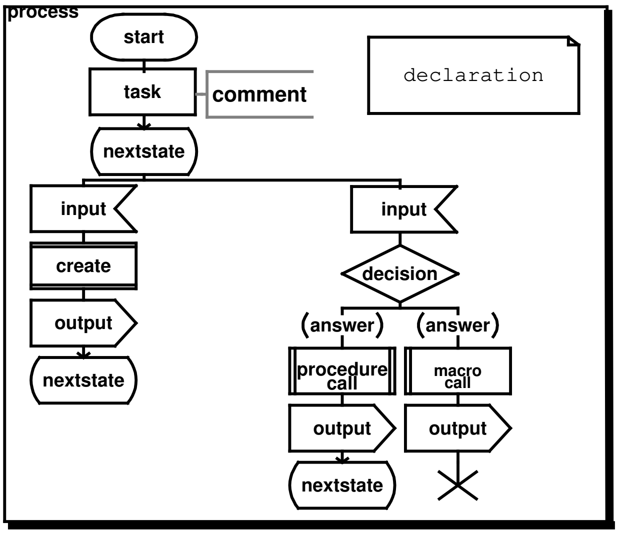

Bild 2: SDL Prozeßbeschreibung (behaviour)

Das Datentypkonzept von SDL basiert auf abstrakten Datentypen (abstract data type, ADT). Abstrakte Datentypen beschreiben das funktionale Verhalten von Objekten durch Spezifikation von Datentypen (sorts) und Operatoren (operators). Durch zusätzliche Angabe von Gleichungen (equations) können die Eigenschaften von Operatoren spezifiziert werden.

newtype Bool
literals true, false;
operators
not: Bool -> Bool;
axioms
not (true) == false;
not (false) == true;
endnewtype Bool;

Bild 3: Beispiel einer einfachen ADT-Deklaration

Ada83[1], Ada95[2]

Standardisierte (ANSI, ISO) Programmiersprache, die neben modernen Konzepten der SW-Erstellung die Programmierung großer, eingebetter Systeme in Realzeitumgebungen unterstützt.

Spracheigenschaften Ada83

- strenges Typkonzept
- Datenabstraktion
- Nebenläufigkeit
- getrennte Übersetzbarkeit
- Progammschablonen
- Ausnahmebehandlung

Spracherweiterungen Ada95

- Objektorientierung
- hierarchische Bibliotheken
- Realzeitprogrammierung

2 Vorstellung der Methodik *SDL/Ada*

2.1 Software-Entwicklungsprozeß

Jeder SW-Entwicklungsprozeß bedarf zur Koordinierung einzelner Tätigkeiten eines organisatorischen Rahmens, der Entwurf und Implementierung des Produkts ‚Software' ermöglicht.

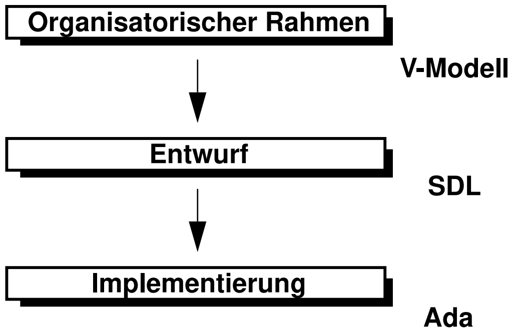

Bild 4: Basisstruktur SW-Entwicklung, Abbildung

Das V-Modell stellt einen solchen Rahmen zur Verfügung, indem mittels eines projektspezifischen Tailorings sämtliche zu erstellenden Entwicklungsergebnisse auf informaler Ebene definiert werden. Die Bedeutung eines solchen Rahmens erwächst aus der Bereitstellung eines standardisierten Vokabulars, auf dessen Basis die Kommunikation zwischen Auftraggeber und Auftragnehmer aber auch zwischen den beteiligten Entwicklern erfolgt.

2.1.1 Beschreibungsebenen

Die Beschreibungsebenen innerhalb des V-Modell Submodells ‚SW-Entwicklung' können in drei Kategorien unterschieden werden:
Beschreibungen in natürlicher Sprache stellen eine *informale Beschreibungsebene* dar, die zur Kommunikation der beteiligten Personen untereinander genutzt wird. Entwicklungsergebnisse in Form ausführbarer Software stellen eine *formale Beschreibungsebene* dar, in der sämtliche Details der Implementierung in einer vom Rechner verständli-

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983
2. Information technology-Programming languages-Ada, International Standard ISO/IEC-8652:1995(E)

chen Programmiersprache formuliert vorliegen. Zwischen beiden Ebenen wird eine *semiformale Beschreibungsebene* vorgesehen, innerhalb derer der Entwurf der gewählten SW-Architektur beschrieben wird. Der Architekturentwurf sollte gut kommunizierbar sein und gleichzeitig als Basis einer automatischen Codegenerierung genutzt werden können. Die zuletzt genannte Anforderung macht eine formale Definition der Spezifikationssprache selbst notwendig.

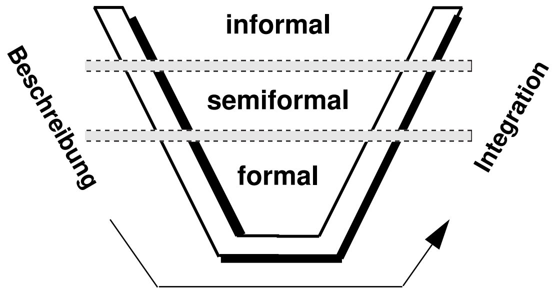

Bild 5: V-Modell, Beschreibungs- / Integrationsebenen

Die Lage der beiden Grenzlinien kann variiert werden. Hierbei ist wegen des Interpretationsspielraums eine Minimierung des informalen Anteils anzustreben. Die Lage der unteren Grenzlinie wird über den gewählten Detaillierungsgrad der Spezifikation bestimmt.

2.1.1.1 Informale Ebene

Zu Beginn jeder SW-Entwicklung stehen Beschreibungen wie Konzeptpapier, Lastenheft und Anforderungspapiere. Kennzeichnend für diese Beschreibungebene sind natürlichsprachliche Texte, oft ergänzt um beliebige graphische Elemente, die das 'WAS' der Applikation beschreiben. Die Differenzierung zwischen *System*, *Segmenten* und *SW-Komponenten* innerhalb des V-Modells impliziert eine erste Strukturierung der Aufgabenstellung nach definierten Gesichtspunkten.

2.1.1.2 Semiformale Ebene

Auf Basis der informalen Beschreibungen entstehen zur Beschreibung einer geeigneten SW-Architektur Entwurfsunterlagen, die das 'WIE' zur Erfüllung der gestellten Anforderungen beschreiben (global design). Typisch für diese Beschreibungsebene ist der Bedarf nach möglichst graphischen Ausdrucksmitteln mit formal definierter Semantik, die den Entwurf der SW-Architektur anschaulich machen und eine Diskussion derselben innerhalb der Entwicklergruppe ermöglichen. Die Zulassung einer gewissen Unschärfe in der Beschreibung ist wegen der zu diesem Zeitpunkt unklaren Details oder zur Abstraktion eine essentielle Forderung an die Beschreibungssprache. Im angesprochenen Telekommunikationsumfeld hat sich die Spezifikationssprache SDL zur Beschreibung von Architektur und Verhalten des zu entwerfenden Systems verbreitet durchgesetzt.

2.1.1.3 Formale Ebene

Innerhalb der formalen Beschreibungsebene findet die Implementierung der mit SDL beschriebenen Architektur statt. Es liegt nahe, in SDL formulierte Entwürfe automatisch in einen Coderahmen zu transformieren. Die manuelle Vervollständigung der nicht in SDL formulierten Details (detailed design) entspricht der Ergänzung von Algorithmen und Datentypen (ADTs). Der Umfang der direkt zu erstellenden Software kann in einem gewissen Rahmen durch angemessene Entwurfsentscheidungen gesteuert werden. Die Programmiersprache Ada bietet durch die konzeptionelle Nähe zu SDL (gemeinsames Anwendungsumfeld) ideale Voraussetzungen sowohl zur automatischen Codegenerierung (Beispiel tasking-Konzept) als auch zur manuellen Programmierung der Details (Beispiel Paket-Konzept).

2.1.2 Integrationsebenen

Den drei Beschreibungsebenen des Submodells stehen drei Integrationsebenen (Modul, Prozeß und Prozessorsystem(e)) gegenüber, deren Existenz den Bedarf an unterschiedlichen Testumgebungen motiviert.

2.1.2.1 Modul

Auf Modul-Ebene werden die üblichen Test- und Integrationswerkzeuge eingesetzt, die in der Regel als Bestandteil einer Programmierumgebung vorzufinden sind. Mittels High Level Debugger können auch in Cross-Umgebungen Kontrollfluß und Speicherinhalte in allen Details (Bitebene) beobachtet werden.

2.1.2.1.1 Prozeß

Auf Prozeß-Ebene gestaltet sich wegen der parallelen Kontrollflüsse die Beobachtung der Abläufe mit herkömmlichen Mitteln aufwendig. An dieser Stelle hat sich die Konzentration auf Prozeßzustand und Nachrichtenfluß zwischen den Prozessen bei gleichzeitiger Verbergung von Details bewährt. Da herkömmliche Debugger eine solche Abstraktion nur bedingt unterstützen, ist ein maßgeschneidertes Testwerkzeug korrespondierend zur Spezifikation vorteilhaft. Darüber hinaus besteht während der Entwicklung oft der Bedarf, fehlende Systemteile auf Spezifikationsniveau zu simulieren. Auf Basis der SDL-Spezifikation kann geeignete Software zur Test- und Simulationsunterstützung automatisch generiert werden.

2.1.2.2 Prozessorsystem(e)

Die Integration eines Prozessorsystems erfolgt üblicherweise durch Stimulierung und Beobachtung der realen Schnittstellen des Integrationsobjektes. In diesem Bereich ist oft der Einsatz von Protokollanalysatoren oder speziellen Werkzeugen (Beispiel Lastsimulation) erforderlich. Die Vielfalt von Schnittstellentypen, Koordinierungsprobleme verteilter Architekturen und zulässige Ablaufvarianz stehen Automatisierungsanforderungen im Sinne von Regressionstests entgegen. In der Praxis sind deshalb aufwendige manuelle ‚Abnahmen' auf Basis von Lastenheft und Anforderungspapieren üblich, die das korrekte Verhalten des Integrationsobjektes in typischen und Grenzbereichen überprüft.

2.2 Software-Architekturen

2.2.1 Entwicklungsumgebung

Der graphische Entwurf der SDL-Spezifikation erfolgt mittels eines kommerziellen Werkzeugs. Als Ergebnis liegt ein dokumentiertes Systemmodell vor, das die *Architektur* und das *Verhalten* des zu entwerfenden Systems auf Spezifikationsniveau beschreibt. Das Werkzeug überprüft hierbei die Konsistenz der Spezifikation und gestattet unter Vernachlässigung realzeitlicher Gesichtspunkte die Simulation des Entwurfs auf dem Entwicklungsrechner.

Mittels eines eigenentwickelten SDL/Ada-Frameworks wird der SDL-Entwurf in Ada-Code transformiert, der um die nicht in SDL spezifizierten Details (Datentypen und Funktionen) durch manuelle Programmierung zu ergänzen ist. Das vollständige Programm stellt eine Ada-Implementierung der SDL-Spezifikation dar, die auf Basis eines SDL/Ada-Laufzeitsystems ausführbar ist.

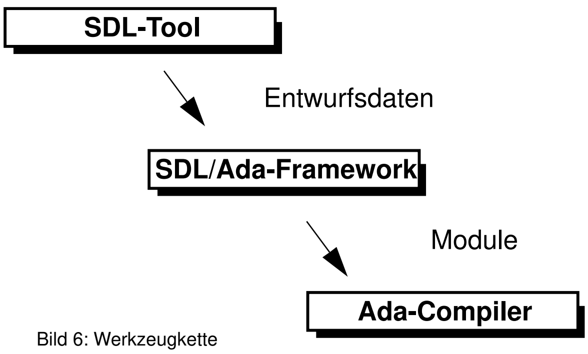

Bild 6: Werkzeugkette

Die in SDL formulierte Spezifikation erlaubt darüberhinaus Codegenerierung zur Testunterstützung indem auf das standardisierte SDL-Kommunikationsmodell aufgesetzt wird. Die Datentypdeklarationen der Signalparameter sind hierzu in Ada-Paketen bereitzustellen.

Spezielle Testwerkzeuge erlauben interaktiv oder programmgesteuert Kommunikationsverhalten und Zustände der Prozesse selektiv zu protokollieren. Durch Eingabe von Signalen über die Testschnittstelle können Prozesse stimuliert werden. Die Test-Ein-/Ausgaben erfolgen in Klartext über einen ausgezeichneten Metaprozeß auf dem Entwicklungs- oder Zielrechner.

Komponenten SDL/Ada-Framework

- **SDL/Ada-Tool**
  - Konsistenzüberprüfung
  - Dokumentation
  - Generierung
    - Schnittstellenpakete
    - interne Systemdarstellung
  - Testcode
- **SDL/Ada-Runtime**
  - **SDL/Ada-Interface**
    - starten / beenden von Prozessen
    - Prozeßidentifikation
    - Nachrichtenverarbeitung
    - Timer
    - Testfunktionen
  - **SDL/Ada-Kernel**
    - Abbildung der statischen Systemstruktur
    - Verwaltung von Prozeßinstanzen
    - Fehlerbehandlung
    - Monitorfunktionen
    - Testfunktionen
- **SDL/Ada-Test**
  - interaktive Testschnittstelle
  - Test - Interpreter

Bild 7 faßt den methodischen Ansatz auf die Realisierungsphasen bezogen noch einmal zusammen. Die SDL-Spezifikation stellt als Ausgangs- und Endpunkt den stabilsierenden Rahmen der Ada-Implementierung dar.

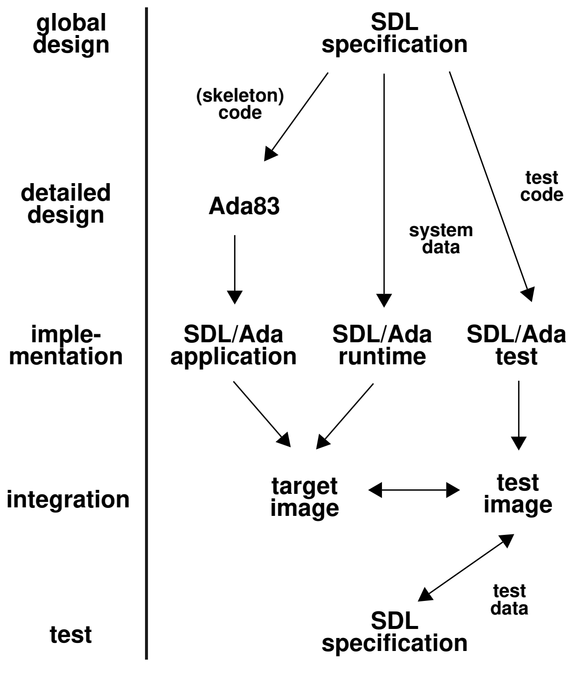

Bild 7: Entwicklungsprozeß

2.2.2 Zielsystem

Bild 8 zeigt ideale und praktische Zielarchitekturen SDL/Ada-basierter Implementierungen. Zwischen Betriebssystem und SDL-Applikation sind die beiden Laufzeitsysteme für Ada und SDL/Ada angesiedelt. Die geschichtete Architektur macht die SDL-Applikation prinzipiell auf jeder Ada-Plattform lauffähig. In der Praxis ist jedoch häufig die Portabilität der idealen SW-Architektur durch direkte Sichtbarkeit der tiefer liegenden Schichten eingeschränkt.

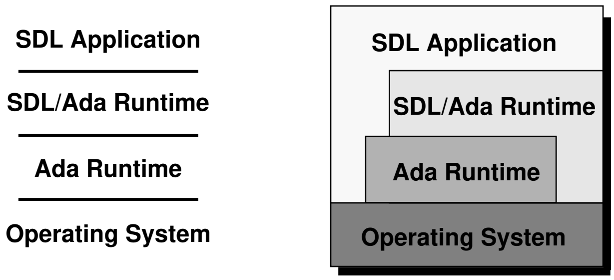

Bild 8: ideale / praktische SW-Architektur

Ein wesentlicher Grund hierfür liegt in der Bereitstellung effizienter asynchroner Kommunikationsmechanismen, die i.d.R. vom Betriebssystem, nicht jedoch von Ada83 bereitgestellt werden. Spracherweiterungen von Ada95 wie z.B. 'protected types' bieten interessante Ansätze, die die ideale Architektur in greifbare Nähe rücken lassen.

3 Anmerkungen zur Methodik *SDL/Ada*

3.1 Charakterisierung

Der im vorliegenden Artikel bechriebene methodische Ansatz ist durch die Verknüpfung von Standards geprägt. Die Integration der drei genannten Standards kann problemlos vollzogen werden. Die Verwendung einer Spezifikationssprache in der mittleren Ebene des V-Modells liefert eine scharfe Abgrenzung von Anforderungsbeschreibung, Architektur-Entwurf und Implementierung. Die unterschiedlichen Sprachebenen erzwingen eine saubere Differenzierung der durchzuführenden Entwicklungsarbeiten wobei der Übergang von der Grob- zur Detailspezifikation bzw. Implementierung rechnergestützt und somit konsistent vollzogen wird. Eine wesentliche Eigenschaft der vorgestellten Methodik ist die Skalierbarkeit, die durch angemessene Abstraktion auf Spezifikationsebene erreicht wird.

3.2 Spezifikationssprache vs. Implementierungssprache

Die Verwendung einer formal definierten Spezifikationssprache dient der präzisen Beschreibung eines geforderten Systemverhaltens bzw. dessen Struktur, die weit über informale und oft mißverständliche Papiere hinausgeht. Die ‚high-level'-Sprachmittel der Spezifikationssprache liefern einen anwendungsunabhängigen jedoch Domainen-spezifischen Baukasten erprobter SW-Komponenten. Der Freiheitsgrad einer direkten Programmierung wird auf Architekturebene gezielt eingeschränkt, wodurch eine Transparenz der Software auf Architekturebene sichergestellt und standardisierte Testschnittstellen ermöglicht werden. Die Spezifikation stellt ein simulierbares Systemmodell dar, das als Analyse- und Implementierungsbasis genutzt werden kann. Durch Einsatz eines Codegenerators kann das Programmgerüst der Implementierung direkt aus der Spezifikation abgeleitet werden. Die Überwindung der Sprachgrenzen (semantic gap) bereitet wegen der konzeptionellen Verwandschaft beider Sprachen keine Probleme. Bild 9 zeigt die realisierte Ada-Implementierung elementarer SDL-Konstrukte.

SDL	Ada83
architecture	data structure
behaviour	program structure
system	program(s)
block	package
process	task
channel	[routing mechanism]
signalroute (in)	[port] task
signalroute (out)	[transport] function
signal	enumeration type
signal data	data type
timer	[timer] task
abstract data type	package

Bild 9: Abbildung SDL -> Ada83

3.3 Ergebnisse

3.3.1 Projekt-Erfahrungen

Der auf die drei Standards abgestützte SW-Entwicklungsprozeß wird durch die vorgestellte Methodik klar strukturiert. Darüberhinaus wird durch die eingesetzten Sprachen eine globale Projektkultur gefördert, die die Grundlage nachvollziehbarer Entwicklungsergebnisse bildet. Die von kommerziellen Werkzeugen bereitgestellte Simulationsfähigkeit des SDL-Entwurfs auf dem Entwicklungsrechner verliert mit der Verfügbarkeit des SDL/Ada-Lauzeitsystems auf der gleichen Rechnerplattform an Bedeutung. In der Wartungsphase hat sich die Verfügbarkeit der Testmittel auf Spezifikationsniveau im Zielsystem für Fehleranalysen als hilfreich erwiesen.

Defizite bzgl. Anwendung und Leistungsfähigkeit der drei Standards behindern singulär und in verstärktem Maße in der Kombination die initiale Akzeptanz der Vorgehensweise.

3.3.2 Produkt-Qualität

Die homogene, visualisierbare Modellierung der Software im Sinne von SDL macht die Implementierung in weiten Teilen nachvollziehbar (Programmgerüst). Der 'Umweg' über die Spezifikationssprache zahlt sich durch die graphische Dokumentation der SW-Architektur und Verfügbarkeit adäquater Testmittel spätestens in der Wartungsphase aus. Die Kombination mit den bekannten qualitätsfördernden Eigenschaften der Programmiersprache Ada macht die Methodik auch gegenüber kommerziellen 'C'-Code generierenden SDL-Werkzeugen überlegen. Die Standardisierung schützt die eigene Wertschöpfung durch Portierbarkeit und Unabhängigkeit von bestimmten Werkzeugen, Plattformen und Herstellern. Diesem Aspekt wird insbesonders bei langlebigen Produkten größte Bedeutung zugemessen.

3.4 Entwicklungspotentiale

3.4.1 Standards

Die aktuell verfügbare SDL/Ada-Entwicklungsumgebung entstand im Rahmen eines Projektes und ist aus nachvollziehbaren Gründen bzgl. ihrer Funktionalität auf Basismechanismen (SDL-Subset) beschränkt.Die Einschränkungen beziehen sich auf verzichtbare SDL-Sprachmittel und Coderahmengenerierung für die Prozeßimplementierung, deren Vervollständigung wünschenswert erscheint. Portierungen des SDL/Ada-Frameworks auf weitere Plattformen (z.B. POSIX, Ada95 bare) werden durch die geschichtete Architektur begünstigt. Die Protokollierung der Prozesskommunikation mit Hilfe der SDL/Ada-Testwerkzeuge liefert derzeit Testdaten, die in einer speziellen Textform vorliegen. Eine Weiterentwicklung in Richtung standardisierter Nachrichtendarstellung (message sequence charts[1], MSC) würde die Integration graphischer Komponente kommerzieller SDL-Werkzeuge ermöglichen. Die SDL/Ada-Testwerkzeuge bieten eine interaktive und eine programmierbare textorientierte Benutzerschnittstelle. In diesem Bereich wäre eine Annäherung an bestehende Standards (Tree and Tabular Combined Notation[2], TTCN) im Sinne des Artikels vorteilhaft.

3.4.2 Architekturen

Die im Kapitel Zielsysteme angesprochenen SW-Architekturen basieren auf realen Implementierungen. Stichworte wie ‚Multiprogramming', ‚Multiprocessing', ‚distributed Processing', ‚distributed SDL' und ‚Ada Partitions' bieten vielfältige Aspekte über weitere sinnvolle Architekturen nachzudenken.

3.4.3 Objektorientierung

Nach [3] werden *Systemstruktur* und *Systemverhalten*, die Basiselemente jeder SDL-Spezifikation, als zwei unabhängige Dimensionen zur Beschreibung der statischen und dynamischen Aspekte eines Systemmodells betrachtet. *Vererbung* als kennzeichnende Größe objektorientierter Ansätze zur Abstraktion und Wiederverwendung von Software wird, wie in Bild 10 dargestellt, als dritte, unabhängige Dimension beschrieben.

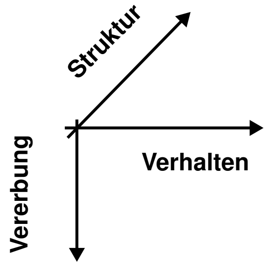

Bild 10: Dimensionen System-Modell

Objektorientierte Spracherweiterungen von SDL92 und Ada95 fordern Klarheit bzgl. einer evolutionären Weiterentwicklung der beschriebenen Methodik SDL/Ada.

Unter der in Bild 11 stilisierten *horizontale*n Integration von OO und SDL sei ein zunächst objektorientierter Ansatz verstanden, der in der Implementierung mit SDL weitergeführt wird. Semantische Unterschiede beider Welten lassen diese Integrationsform fragwürdig erscheinen.

Die *vertikale* Integration bearbeitet getrennte SW-Bereiche mit beiden Ansätzen parallel. Die Ergebnisse werden erst zur Bindezeit zusammengeführt und stellen somit keine echte Integration dar.

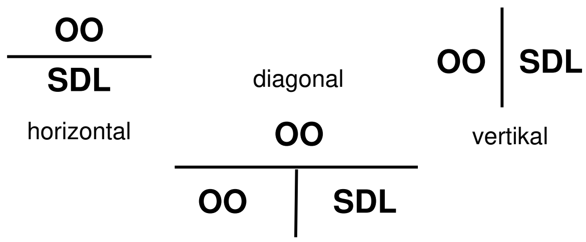

Bild 11: OO / SDL Integrationsstrategien

Bei der vorgeschlagenen *diagonale*n Integration wird Objektorientierung zunächst als übergeordnete Methodik nur im Bereich der Analyse (OOA) einsetzt. Nach Identifikation aktiver und passiver Objekte werden im Grobentwurf die aktiven Objekte wie bisher als SDL-System modelliert. Die passiven Objekte entsprechen aus SDL-Sicht den abstrakten

1. New Recommendation Z.120, CCITT, Message Sequence Charts (MSC), 1992
2. Draft Recommendation X.292, CCITT, OSI Conformance Testing Methodology and Framework for Protocol Recommendations for CCITT Applications - The Tree and Tabular Combined Notation (TTCN), 1992

Datentypen (ADTs), die auch im Feinentwurf objektorientiert modelliert werden können.

Bild 12 zeigt den ‘diagonalen‘ Migrationspfad zur Integration von OO und SDL im Ada-Kontext. Die Sprachmittel von Ada95 gestatten die Methoden-Integration im Feinentwurf.

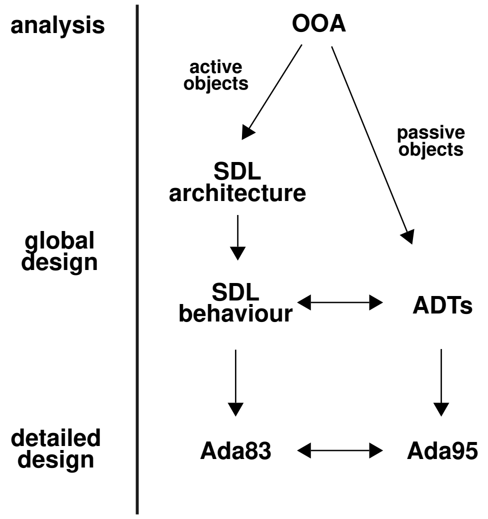

Bild 12: OO / SDL / Ada Integration

4 Schlußbemerkungen

Die Verknüpfung einer anwendungsspezifischen Spezifikationssprache mit einer leistungsfähigen Programmiersprache hat sich in einem realen Projekt als methodischer Ansatz voll bewährt. Die Standardisierung beiden Sprachen und des praktizierten organisatorischen Rahmens dient der langfristigen Sicherung der eigenen Wertschöpfung. Die Aussagen sind im Wesentlichen auf folgende Erfahrungen gestützt:

Sprachen

- Die praktizierte SW-Entwicklungskultur wird weitgehend durch die zur Verfügung stehende(n) Sprache(n) geprägt und somit positv beeinflußt.
- Die Sprachen SDL und Ada haben durch ihren gemeinsamen Enstehungscharakter (Reviews) Ähnlichkeiten bzgl. Qualität und Leistungsfähigkeit.
- Die Abgrenzung von Grob- und Feinentwurf bzw. Implementierung wird ohne semantischen Bruch durch zwei Sprachebenen erzwungen.
- Zum Entwurf stehen formal definierte high-level Sprachelemente zur Verfügung, die ein nahezu intuitiv verständliches Prozeß- und Kommunikationsmodell verwenden.
- Die Abbildung des Entwurfs auf die Programmiersprache kann automatisiert werden.
- Integration und Test kann durch die homogene Kommunikationsinfrastruktur auf Spezifikationsniveau durchgeführt werden.
- Der Detaillierungsgrad des Grobentwurf‘s (Abstraktion) kann nach Bedarf individuell gesteuert werden.

Standards

- Die Standardisierung ist Grundlage der Verfügbarkeit kompatibler, kommerzieller Werkzeuge.
- Das projektweite Implementierungsmodell bleibt über die Lebendauer des Produkts konstant.
- Kommunikationsprobleme zwischen den beteiligten Personen werden durch das ‚standardisierte Kommunikationsprotokoll‘ weitgehend vermieden.
- Stabilität und Verbreitung der Standards vergrößern das in Frage kommende Mitarbeiterpotential, Reibungsverluste durch Mitarbeiterwechsel werden minimiert.
- Architektur und Prozeßverhalten können mittels kommerzieller Werkzeuge entworfen und visualisiert werden.
- Unabhängigkeit von Plattform und Hersteller kann für einen Systemwechsel z.B. aus wirtschaftlichen Überlegungen genutzt werden.
- Der durch den Entwicklungsstandard geprägte organisatorische Rahmen stellt das erforderliche informale Vokabular auch zur projektübergreifenden Kommunikation bereit.

5 Literatur

[1] F. Belina, D. Hogrefe, A. Sarma (1991)
„SDL with Applications from Protocol Specification“, Prentice Hall

[2] R. Braek, O. Haugen (1993)
„Engineering Real Time Systems,
An object-oriented methodology using SDL“,
Prentice Hall

[3] B. Selic, G. Gullekson, P. Ward (1994)
„Real-Time Object-Oriented Modeling“,
John Wiley & Sons, Inc.