Die Spezifikationssprachen MSC und SDL

Teil 2: Specification and Description Language (SDL)

Jens Grabowski, Ekkart Rudolph und Michael Schmitt

Im Telekommunikationsbereich haben sich *Message Sequence Chart* (MSC) und *Specification and Description Language* (SDL) als wichtigste Modellierungssprachen etabliert. Im ersten Teil dieses Artikels wurde die Anforderungsspezifikation mit MSC erläutert. Der vorliegende zweite Teil geht auf die Beschreibung von verteilten Systemen mit SDL ein und skizziert den Einsatz von modernen Software-Werkzeugen.

In the telecommunication area, *Message Sequence Chart* (MSC) and *Specification and Description Language* (SDL) are the dominating modeling languages. In the first part of the article, requirement specification with MSC has been illustrated. In this part, the description of distributed systems with SDL and the use of modern software tools is considered.

3 Specification and Description Language

Die *Specification and Description Language* (*SDL*) [2; 4] ist eine graphische Beschreibungssprache für die Modellierung verteilter Systeme. Sie ermöglicht es, sowohl das funktionale Verhalten als auch die Struktur eines Systems zu spezifizieren.

SDL ist eine formale Sprache mit einer klar definierten Semantik. Daher ist es möglich, SDL-Spezifikationen nicht nur als informativen Zusatz, sondern auch als normativen Bestandteil in der Standardisierung zu verwenden. Aufgrund der eindeutigen Interpretation der Sprachkonstrukte können SDL-Systeme zudem direkt in ausführbaren Code transformiert werden, so dass sich SDL auch als Implementationssprache anbietet.

Die Semantik von SDL basiert auf dem Konzept parallel ablaufender endlicher Zustandsmaschinen, das um lokale Daten und Uhren (*Timer*) erweitert wurde. Als Kommunikationsmechanismus zwischen den Zustandsmaschinen dient die asynchrone Übertragung von Signalen. Jede Signalübertragung erfolgt über Kanäle, die Sender und Empfänger miteinander verbinden. SDL bietet aber auch Sprachkonstrukte an, um synchrone Kommunikation in Form von *Remote Procedure Calls* beschreiben zu können.

3.1 Agenten und Strukturierung

Eine SDL-Spezifikation besteht aus einer Vielzahl von Diagrammen, die ein verteiltes System in seiner hierarchischen Struktur darstellen.

Das Grundgerüst einer jeden SDL-Spezifikation bilden *Agenten*. Jeder Agent wird separat durch ein oder mehrere Diagramme beschrieben. Ein Agent kann aus einer Zustandsmaschine, Prozeduren und Variablen bestehen. Ferner dürfen Agenten weitere Agenten enthalten. Es existieren zwei unterschiedliche Arten von Agenten, *Blöcke* und *Prozesse*, die sich im Grad der Parallelität und Atomizität von Anweisungen unterscheiden. Der äußerste Block einer SDL-Spezifikation wird als *System* bezeichnet.

Bild 1 zeigt ein SDL-Systemdiagramm für das Produktionsprozess-Beispiel [3]. Das System besteht aus den drei Blöcken `TeileC`, `TeileD` und `Roboterplatz`. Jeder Block stellt eine eigenständige Einheit dar, deren Ausführung *parallel* zu allen anderen Blöcken erfolgt. Sollen die Blöcke synchronisiert werden, so muss dies durch den Austausch von Signalen erfolgen (siehe Abschnitt 3.2 und 3.3).

Damit eine SDL-Spezifikation vollständig ist, muss jeder referenzierte Agent in einem weiteren Diagramm definiert sein. Die *USE*-Anweisung in Bild 1 referenziert auf ein Paket, in dem weitere Diagramme zusammengefasst sind. Für den `Roboterplatz` vom Typ `Produktionszelle` ist eine entsprechende Spezifikation in Bild 2 angegeben. Wie zu erkennen ist, gliedert sich die Produktionszelle in drei *Prozesse* auf, nämlich `Diskriminator`, `Ortsschalter` und `Roboter`.

Obwohl es prinzipiell möglich ist, Blöcken eigene Zustandsmaschinen zuzuordnen und Prozesse wieder in Blöcke zu unterteilen, ist es üblich, Blöcke zur hierarchischen Strukturierung des Systems nach logischen oder implementationstechnischen Gesichtspunkten zu verwenden und das

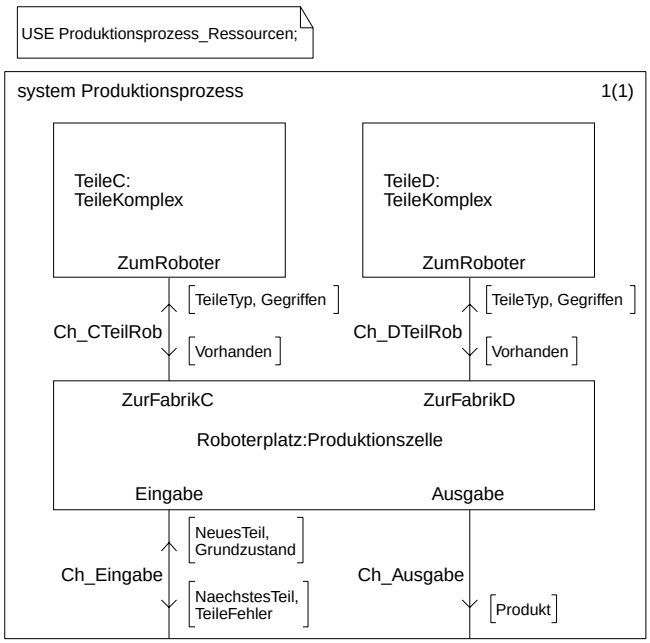

Bild 1: System `Produktionsprozess`.

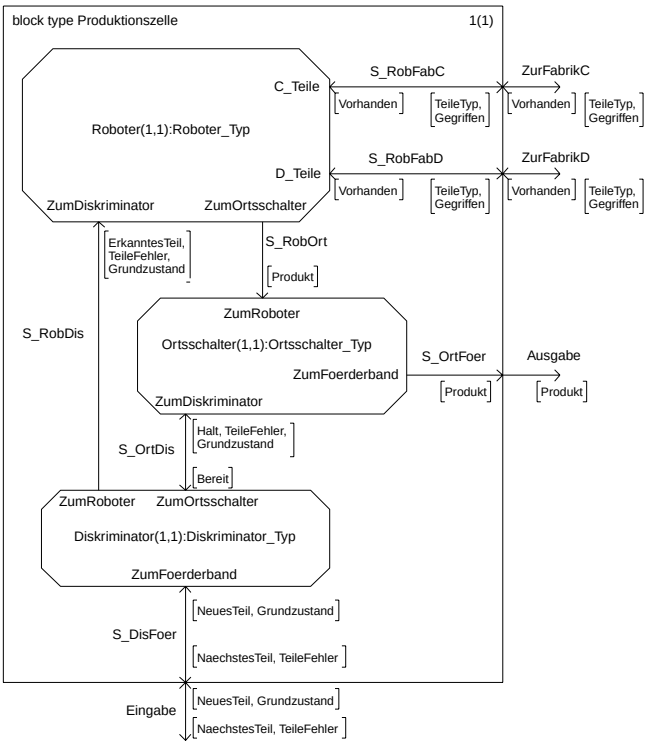

Bild 2: Blocktyp `Produktionszelle`.

funktionale Verhalten auf unterster Ebene in Prozessen zu beschreiben.

3.2 Kommunikation

Die Kommunikation zwischen Agenten erfolgt in SDL durch den Austausch von Signalen. Hierzu müssen der sendende und empfangende Agent durch einen *Kanal* miteinander verbunden sein.

Kanäle können in SDL wahlweise uni- oder bidirektional sein, wobei die Semantik eines bidirektionalen Kanals durch zwei unidirektionale Kanäle definiert ist. Für einen Kanal kann spezifiziert werden, ob er Signale mit oder ohne Verzögerung überträgt.

In Bild 1 sind die drei Blöcke über insgesamt vier Kanäle sowohl untereinander, als auch mit der Systemumgebung (dem Förderband) verbunden. Die Übertragung der Signale erfolgt dabei verzögert, wie anhand der mittig angeordneten Pfeilspitzen zu erkennen ist (vgl. mit den verzögerungsfreien Kanälen in Bild 2).

Jeder Kanal ist charakterisiert durch die Listen der Signale, die in die jeweilige Richtung übertragen werden können. Z. B. kann Block `RoboterPlatz` über Kanal `Ch_CTeilRob` die Signale `TeileTyp` und `Gegriffen` an Block `TeileC` verschicken und das Signal `Vorhanden` empfangen.

Ist ein Agent mit seiner Außenwelt verbunden, so ist pro Kanal ein Zugangspunkt (*gate*) anzugeben. Dieser Zugangspunkt muss sowohl in dem Diagramm gekennzeichnet werden, in dem der Agent referenziert wird, als auch bei der Definition des Agenten selbst.

In Bild 1 ist der Agent `Roboterplatz` über den Zugangspunkt `ZurFabrikC` mit `TeileC` verbunden. In Bild 2 wird dieser Zugangspunkt definiert und mit dem Kanal `S_RobFabC` verknüpft.

Zugangspunkte sind notwendig, um den Bezug zwischen den verschiedenen Diagrammen einer Spezifikation herzustellen. Über die Zugangspunkte kann auch herausgefunden werden, welche Zustandsmaschinen über welche Kanäle (bzw. Folgen von Kanälen) miteinander kommunizieren können, auch wenn die tatsächlichen Kommunikationswege – wie im vorliegenden Fall für den Prozess `Roboter` – nur indirekt über eine Hierarchie von Blockdiagrammen definiert sind.

3.3 Verhaltensbeschreibung

Das dynamische Verhalten eines SDL-Systems wird durch endliche Zustandsmaschinen beschrieben. Eine Zustandsmaschine ist charakterisiert durch eine endliche Menge von Zuständen sowie Zustandsübergängen (*Transitionen*). Jede Zustandsmaschine verfügt über genau eine Eingabewarteschlange, in der alle eintreffenden Signale zwischengespeichert werden.

Eine Zustandsmaschine wartet entweder auf eine Eingabe aus der Warteschlange oder vollzieht einen Zustandsübergang. Während einer Transition kann eine Zustandsmaschine verschiedenste Aktionen durchführen, z. B. Werte von lokalen Variablen verändern oder ihrerseits Signale versenden.

Eingabewarteschlange und Zustandsmaschine arbeiten unabhängig voneinander, so dass auch während der Ausführung eines Zustandsübergangs keine Signale verloren gehen. Zwei von unterschiedlichen Quellen/Kanälen gleichzeitig eintreffende Signale werden dabei in zufälliger Reihenfolge in der Warteschlange gepuffert.

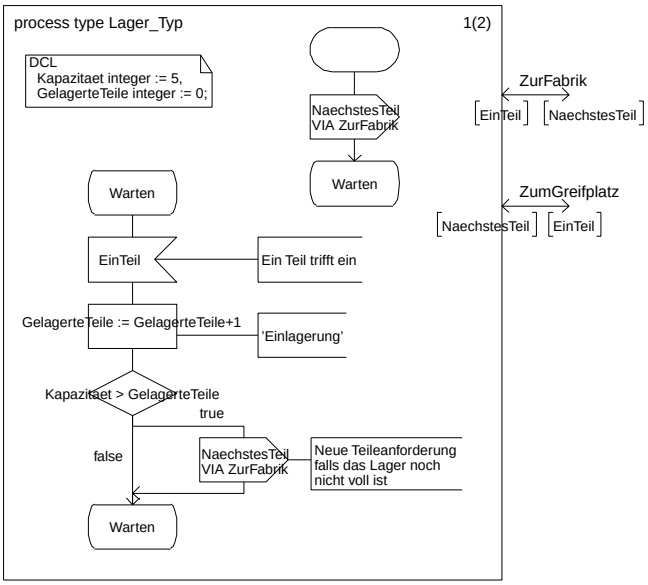

Bild 3: Prozesstyp `Lager_Typ`.

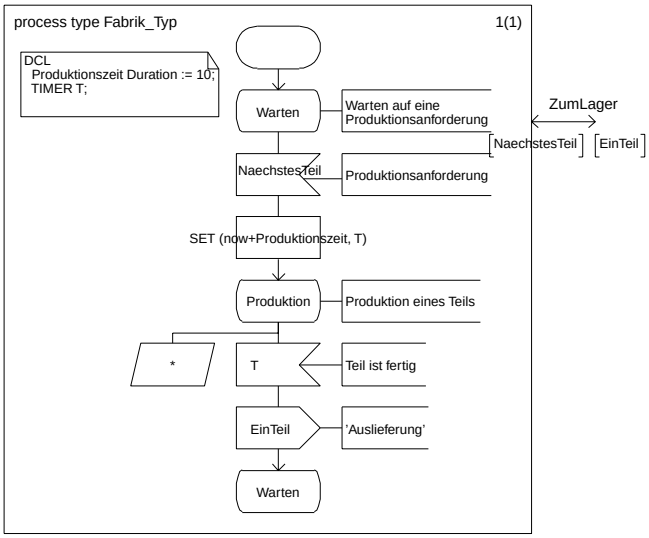

Bild 4: Prozesstyp `Fabrik_Typ`.

Bild 3 zeigt einen Ausschnitt einer SDL-Prozessdefinition, in der ein Lager mit einer Kapazität von 5 Teilen modelliert ist. Ausgehend vom Startzustand (in SDL dargestellt als leeres ovales Symbol) sendet die Zustandsmaschine zunächst das Signal `NaechstesTeil` an die Fabrik, um den Lagerbestand aufzufüllen. Danach wechselt sie in den Zustand `Warten` und wartet auf ein eingehendes Signal. Trifft in diesem Zustand anschließend das Signal `EinTeil` ein, inkrementiert die Zustandsmaschine zunächst die Anzahl der gelagerten Teile. In Abhängigkeit davon, ob die maximale Kapazität des Lagers bereits erreicht ist oder nicht, wird danach ein weiteres Teil über den Zugangspunkt `ZurFabrik` von einem entfernten Agenten angefordert, und in den Zustand `Warten` zurückgekehrt. `Lager_Typ` verfügt über zwei lokale Variablen (`Kapazitaet` und `GelagerteTeile`), die mittels `dcl`-Anweisungen (*declaration*) eingeführt werden.

Die Ausführung einer Zustandsmaschine wird jeweils durch das erste Signal in der Warteschlange gesteuert. Ist für dieses Signal im aktuellen Zustand keine Transition definiert, so wird es gemäß SDL-Semantik verworfen. In manchen Situationen ist es jedoch wünschenswert, unerwartete Signale solange in der Warteschlange zu verwahren, bis sie sinnvoll abgearbeitet werden können. Beim Prozesstyp `Fabrik_Typ` (Bild 4) wird das sogenannte *save*-Konstrukt (dargestellt als Raute) eingesetzt, um im Zustand `Produktion` alle Signale (`*`) in der Warteschlange zu belassen, da während der Produktion eines Bauteils keine Anfragen nach neuen Teilen bearbeitet werden können.

Timer

Das Verhalten einer Zustandsmaschine kann nicht nur durch eintreffende Signale, sondern auch mittels *Timer* (Uhren) gesteuert werden. Jede Zustandsmaschine kann eine beliebige Anzahl von lokalen Timern verwalten, auf die sie mit den Operationen *Set* und *Reset* zugreift. *Set* aktiviert einen Timer und hat als Argument den Zeitpunkt, zu dem der Timer ablaufen soll (in der Regel ein Ausdruck der Art `now + X`). *Reset* stoppt einen Timer. Läuft ein Timer ab, so wird ein Signal mit dem Namen des Timers in die Warteschlange der Zustandsmaschine eingefügt.

Der Einsatz von Timern ist sinnvoll, um das Blockieren von Agenten zu verhindern, die auf ein bestimmtes Signal warten. Trifft das Signal innerhalb einer vorgegebenen Zeitspanne nicht ein, so läuft ein vorher gesetzter Überwachungstimer ab und das Timer-Signal löst einen Zustandsübergang aus. Timer eignen sich darüber hinaus auch zur Beschreibung von Ereignissen, die sich in bestimmten zeitlichen Abständen wiederholen.

Im Prozess `Fabrik_Typ` (Bild 4) wird ein Timer `T` eingesetzt, um die Erstellung eines neuen Bauteils in der vorgegebenen Zeitdauer `Produktionszeit` zu beschreiben. Sobald im Zustand `Warten` eine Produktionsanforderung (Signal `NaechstesTeil`) eintrifft, wird `T` gesetzt und in den Zustand `Produktion` gewechselt. Läuft der Timer ab, wird das Signal `EinTeil` an das Lager gesendet.

Bei der Verwendung von Timern ist zu beachten, dass SDL keine Zeiteinheiten definiert. Es bleibt also dem Betrachter überlassen, in Bild 4 die `Produktionszeit` als 10 Sekunden, 10 Stunden, o. ä. zu interpretieren. SDL macht außerdem keine Annahmen über die Ausführungsdauer der einzelnen Aktionen einer Zustandsmaschine.

Dynamische Agentenerzeugung

SDL unterstützt die dynamische Erzeugung von Agenten. Damit ist es z. B. möglich, Serverarchitekturen zu beschreiben, bei denen die eingehenden Dienstanfragen jeweils von einem eigenen Prozess bearbeitet werden. In SDL kann kein Agent die Terminierung eines anderen Agenten erzwingen. Jeder Agent ist selbst für seine Beendigung verantwortlich. Es ist jedoch möglich, einen Agent so zu spezifizieren, dass er in jedem Zustand ein spezielles Signal akzeptiert, auf das er mit Terminierung reagiert.

3.4 Objektorientierung

Die Sprache SDL folgt dem objekt-orientierten Paradigma. Ebenso wie Datentypen dürfen Agententypen, Prozeduren und Signale durch Vererbung spezialisiert werden. Auf diese Weise ist es möglich, neue Eigenschaften hinzuzufügen oder bestehende Eigenschaften umzudefinieren. Bei Agenten ist es beispielsweise möglich, einzelne Transitionen ihrer Zustandsmaschine in einem abgeleiteten Agenten zu überschreiben. Um SDL-Typen in verschiedenen Kontexten verwenden zu können, lassen sich diese zudem parametrisieren.

3.5 Datenmodell

Im Gegensatz zu MSC [5], das über eine Art Meta-Datenkonzept beliebige externe Datenbeschreibungssprachen einbindet, verfügt SDL über eine eigene Datensprache.

Neben einer Reihe von vordefinierten einfachen Datentypen wie *boolean*, *character*, *integer* oder *real* stellt es einige spezielle Datentypen, etwa zur Kommunikation (*pid – process identifier*) oder für Timeroperationen (*duration*) bereit. Aus den vordefinierten und speziellen Datentypen können verschiedene strukturierte Datentypen gebildet werden (*struct*, *array*, *choice* etc.).

SDL unterscheidet zwischen Werten und Objekten. Letztere sind Referenzen auf Werte. Die Semantik von SDL-Objekten ist der Semantik für Referenzen in der Programmiersprache Java sehr ähnlich.

Als Besonderheit stellt SDL einen *any*-Ausdruck zur Verfügung, der einen unspezifizierten Wert eines vorgegebenen Datentyps zurückliefert. Mit Hilfe von *any* ist es möglich, Nichtdeterminismus im Verhalten eines Systems zu beschreiben.

3.6 Weitere Sprachmerkmale

SDL bietet eine Reihe weiterer Sprachkonstrukte an, deren Diskussion hier aufgrund des begrenzten Umfangs nicht möglich ist. So hat SDL unter anderem Kontrollstrukturen, mit denen auch komplexe Algorithmen sowohl graphisch als auch in textueller Form beschrieben werden können. Ein weiteres Merkmal von SDL ist das von C++ entliehene Konzept des Exception-Handlings. Nähere Informationen und eine vollständige Beschreibung von SDL bietet die SDL-Forum Society [6].

4 Entwicklungswerkzeuge

Der Erfolg einer Sprache ist oftmals an die Verfügbarkeit leistungsfähiger Entwicklungsumgebungen gekoppelt. Für MSC und SDL existieren eine Reihe von kommerziellen und akademischen Softwaretools [7]. Die Palette der Werkzeuge reicht von graphischen Editoren, die den Entwickler mit den passenden Symbolen und inkrementeller Syntaxprüfung unterstützen, bis hin zu Compilern zur automatischen Generierung von C++- oder Java-Code aus SDL-Spezifikationen.

Die „semantische Nähe“ von SDL und MSC erlaubt es, beide Sprachen kombiniert einzusetzen. So kann bei der schrittweisen Simulation von SDL-Spezifikationen der jeweilige Ablauf in Form eines MSCs dargestellt werden. Umgekehrt kann aber auch verifiziert werden, ob ein durch ein MSC beschriebener Ablauf für ein SDL-System möglich ist.

Als Ergänzung zu Simulatoren können Validationstools eingesetzt werden. Diese simulieren automatisch alle möglichen durch eine SDL-Spezifikation beschriebenen Abläufe und überprüfen die Spezifikation dabei auf potentielle Probleme wie z. B. Deadlocks oder unerreichbaren Code.

5 Zusammenfassung

Die Sprachen SDL und MSC haben in der Telekommunikation eine bedeutende Stellung erreicht. Nahezu alle großen Hersteller setzen MSC und SDL für ihre Produktentwicklung ein. Obwohl SDL zunächst als reine Spezifikationssprache konzipiert wurde, hat in den vergangenen Jahren ihre Bedeutung als Implementierungssprache stark zugenommen.

Aufgrund der auch im Telekommunikationsbereich zunehmenden Popularität von UML bemühen sich Anwender und Werkzeughersteller derzeit um eine Harmonisierung von MSC, SDL und UML. Dabei sollen die Vorzüge von MSC und SDL, nämlich ihre formale Semantik und die ausgereifte Entwicklungsmethodik, bewahrt bleiben.

Literatur

[1] G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modelling Language User Guide*. Addison-Wesley, 1999.

[2] J. Ellsberger, D. Hogrefe, A. Sarma. *SDL – Formal Object-oriented Language for Communicating Systems*. Prentice Hall, 1997.

[3] S. Kowalewski. *Modellierungsmethoden aus der Informatik*. In: at-Automatisierungstechnik 49, Heft 9, Sept. 2001.

[4] ITU-T SG 10. *Specification and Description Language* (SDL). Rec. Z.100, Geneva, 2000.

[5] ITU-T SG 10. *Message Sequence Chart* (MSC). Rec. Z.120, Geneva, 2000.

[6] SDL Forum Society. http://www.sdl-forum.org

[7] SDL-Tools. http://www.sdl-forum.org/Tools/index.htm

Manuskripteingang: 19. Februar 2001.

Vorstellung der Autoren im Teil 1, at Heft 12 (2001) Seite A19