UML - Unified Modeling Language im Einsatz[1]

Teil 3: UML-RT für echtzeitkritische und eingebettete Systeme

Bernhard Rumpe, Robert Sandner

Auf Basis der grundlegenden Konzepte der UML wird im dritten Teil das Sprachprofil UML-RT besprochen. UML-RT zielt auf die Entwicklung technischer, echtzeitkritischer Anwendungen.

Unified Modeling Language in Practice

Part 1 and 2 introduced the basic concepts of the standard UML. This article discusses the profile UML-RT which focuses on the development of technical, embedded systems.

5. Hintergrund der UML-RT

Die in Teil 1 und 2 dieser Reihe besprochenen Konzepte der UML eignen sich vor allem für die Entwicklung von Business-Software. Dort nimmt die UML eine dominierende Rolle ein. Für die Entwicklung technischer Anwendungen weist die Standard-UML dagegen wie die meisten objektorientierten Sprachen erhebliche Defizite auf. Dies betrifft vor allem die Darstellung von Systemarchitekturen, Kommunikationsaspekten und Echtzeiteigenschaften. Von verschiedenen Seiten werden deshalb Weiterentwicklungen der UML vorgeschlagen, um diese Defizite zu beheben [19,20,21]. Eine erhebliche Bedeutung hat die Integration von Elementen der ROOM-Methodik [19] in die UML erlangt [22], die durch einige Werkzeuge, darunter Rose-RT [23] implementiert wird.

In „Requests for Proposals“ hat die OMG 1999 begonnen, Erweiterungen der UML in einen einheitlichen Standard zu integrieren. Im Rahmen eines Redesigns der Sprache in der Version 2.0 werden unter anderem erweiterte Architekturbeschreibungstechniken Einzug halten. In einem weiteren Request für ein *UML Profile for Scheduling, Performance, and Time* [24] werden dabei auch Echtzeiterweiterungen aufgegriffen.

In diesem Teil der Serie werden Konzepte aus ROOM vorgestellt, die zu großen Teilen bereits als Erweiterungen in UML-Werkzeuge integriert sind und sehr wahrscheinlich in die Standardisierung der OMG einfließen werden. Dem üblichen Sprachgebrauch folgend werden diese Erweiterungen kurz als UML-RT bezeichnet. Die Darstellung ist am Werkzeug Rose-RT orientiert.

5.1 Konzepte der Erweiterung durch die OMG

Obwohl der UML 2.0 Standard noch nicht offiziell dokumentiert ist, zeichnet sich ab, dass die Architekturmodellierung einen erheblichen Stellenwert einnehmen wird. Im Vordergrund stehen dabei

- (hierarchisch geschachtelte) Komponenten und
- Kommunikation zwischen Komponenten

Die Architekturmodellierung betrachtet im Gegensatz zu Klassendiagrammen die Beziehungen von Objekten (Instanzen) des Systems. Im Vordergrund stehen dabei die hierarchische Gruppierung zu komplexeren Komponenten sowie die Schnittstellen, über die Objekte miteinander kommunizieren. Als Kommunikationsmittel wird hier zumeist nicht der Methodenaufruf, sondern die asynchrone Kommunikation mittels (gepufferter) Nachrichten benutzt.

Für die Echtzeiterweiterung der UML [24] liegt bereits ein Vorschlag von namhaften Werkzeugherstellern vor, die ihre derzeit unterstützten Dialekte in einem gemeinsamen Ansatz integrieren wollen [25]. Dieser Vorschlag lässt noch viele Freiheiten für die Unterstützung der Modellierung durch Werkzeuge, identifiziert jedoch drei wesentliche Konzepte, die in die UML einfließen sollen:

- Resourcen und Quality of Service Attribute
- Zeitkonzepte
- Scheduling

Wesentliche nichtfunktionale Anforderungen in technischen Anwendungen an Ressourcen wie Komponenten und Kommunikationskanäle werden mittels *Quality of Service* (QoS) Attributen definiert. Diese können als UML-Stereotypen den betreffenden Modellierungselementen zugeordnet werden. Solche Mechanismen bieten zunächst keine Garantie für die Einhaltung von Qualitätsmerkmalen, sondern stellen ein Mittel der Anforderungsdokumentation dar.
Für das wichtigste QoS-Attribut – Echtzeitfähigkeit – sind zwei Zeitmodelle vorgesehen: Kontinuierliche

[1] Erschienen in: at-Automatisierungstechnik, Heft 11/2001, S. A15-A18, Oldenbourg Verlag, München.

[RS01b] B. Rumpe, R. Sandner.
UML - Unified Modeling Language im Einsatz. Teil 3. UML-RT für echtzeitkritische und eingebettete Systeme.
In: at - Automatisierungstechnik, Reihe Theorie für den Anwender, Band 49, Heft 11. 2001.
www.se-rwth.de/publications

(beliebig genaue) Zeit und ein für Softwareanwendungen häufig gebrauchtes gröberes, diskretes Zeitmodell. Um Echtzeitbezüge in Modellen auszudrücken, stehen Uhren und Timer zur Verfügung. Timer sind rücksetzbare Uhren, die (wie ein Wecker) Signale absetzen können.
Für die Darstellung von Leistungsanforderungen wird in [25] über die explizite Modellierung von Scheduling-Strategien mit der UML diskutiert. Diese sind allerdings stark von der Implementierungsplattform abhängig. Ihre Einbeziehung in die UML steht deshalb im Widerspruch zum Ziel einer möglichst allgemeinen Modellierungssprache. Wir gehen deshalb auf Scheduling-Strategien nicht weiter ein.

5.2 Umsetzung der Konzepte in UML-RT

UML-RT bietet konkrete Erweiterungen der UML an, die sich unmittelbar in die oben diskutierten Konzepte einordnen lassen: Im Mittelpunkt stehen Elemente zur Beschreibung der Architektur eines Systems, insbesondere zur Darstellung hierarchischer Komponentenstrukturen und Kommunikationsbeziehungen. Beide Aspekte werden in Capsule Diagrammen dargestellt. Zur Modellierung von Zeit stehen Uhren und Timer zur Verfügung[2]. Scheduling-Strategien können mittels Prioritäten für Nachrichten beeinflusst werden, allerdings fehlt hier ein verbindliches mathematisches Modell.
UML-RT stellt in dieser Form einen Vorschlag dar, der die wesentlichen Ziele der Standardisierung durch die OMG umsetzt. Allerdings ist die Ausgestaltung des Standards ein von vielen, auch kommerziellen, Interessen abhängiger Prozess, der noch nicht abgeschlossen ist.

6. Architekturbeschreibung: Capsules

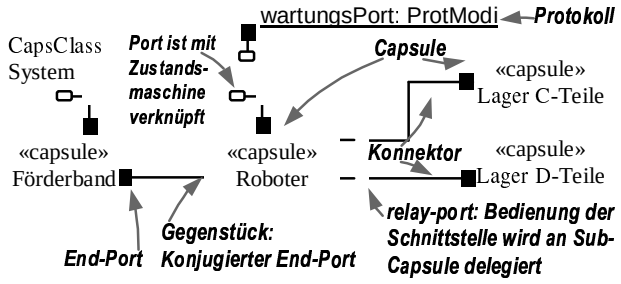

Abbildung 11: Capsule Diagramm des Gesamtsystems

Capsule Diagramme dienen der Darstellung der Komponenten (Capsules) und der logischen Kommunikationsbeziehungen im zu entwickelnden System[3]. Das Capsule Diagramm in Abb. 11 beschreibt die Architektur des als Anwendungsbeispiel betrachteten Produktionssystems. Das zu entwickelnde System wird in die beiden Komponenten Roboter und Förderband zerlegt. Jede dieser Komponenten besitzt mehrere *Ports,* die Kommunikationsschnittstellen mit der Umgebung definieren. Einem Port wird im Capsule Diagramm eine (von zumeist zweien) durch ein *Protokoll* definierte Rolle zugewiesen. Protokolle spezifizieren eine Schnittstelle für die Kommunikation und werden im folgenden Abschnitt besprochen. Das Verhalten einer Capsule wird mit einer Variante von Statecharts spezifiziert, die in Abschnitt 9 besprochen wird. *Konnektoren* zwischen den Ports beschreiben die „logischen" Kommunikationsbeziehungen. Diese werden normalerweise direkt in Kommunikationsbeziehungen der Implementierung umgesetzt, können aber auch durch Kommunikation über Dritte oder Busse realisiert werden.
Komponenten können hierarchisch aufgebaut werden: Jede Komponente kann ihrerseits in einem Capsule Diagramm in weitere Komponenten gegliedert sein. Eine hierarchisch zerlegte Komponente kann eine nach außen angebotene Schnittstelle entweder selbst bedienen oder die Bearbeitung an eine Sub-Komponente delegieren. Im ersten Fall wird ein End-Port mit dem Statechart der Komponente verknüpft. Im zweiten Fall wird ein Relay-Port mit Ports der Sub-Komponente verbunden. Diese hierarchische Zerlegung erlaubt eine beliebige tiefe Schachtelung von Komponenten.

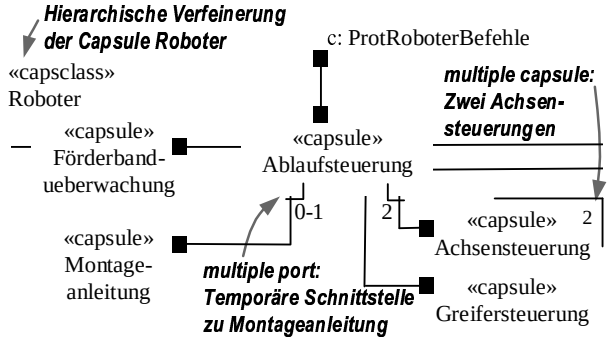

Abbildung 12: Aufbau eines Roboters

Abb. 12 zeigt den Aufbau der Roboter-Komponente. Dabei wird ein Entwurfsmuster genutzt, das die Steuerung des Betriebsablaufs von der Steuerung der einzelnen Mechanikkomponenten trennt. Die Ablaufsteuerung bedient sich zur Planung des Montageablaufs der Montageanleitung. Der Port der Ablaufsteuerung zur Montageanleitung ist mit der Multiplizität 0-1 versehen, da die verbundene Komponente erst bei Bedarf erzeugt wird.
Sowohl der Roboter als auch die Ablaufsteuerung in Abb. 12 sind Subklassen einer Oberklasse Automatisierungskomponente. Solche Spezialisierungsbeziehungen werden nicht in Capsule-Diagrammen sondern wie gewohnt in Klassendiagrammen ausgedrückt. Die Oberklasse definiert allgemeine Mechanismen für Wartung und Betrieb von Automatisierungskomponenten. Dieses Entwurfsmuster [26] stellt sicher, dass die spezialisierte Funktionalität jeder Komponente gekapselt wird und nicht durch nachträgliche Integration von Mechanismen für die Wartung ihre klaren Schnittstellen verliert. Im folgenden wird daher bei der Entwicklung der Protokolle und Statecharts des Systems die spezialisierte Funktionalität der Komponenten in die Wartungs-Mechanismen eingebettet.

[2] Weitere QoS-Attribute können Anwender wie im OMG-Profil vorgesehen als Stereotypen selbst definieren. Allerdings existiert dafür kein standardisiertes Vorgehen.

[3] Der Begriff „Capsule" wird im Artikel durch „Komponente" übersetzt, während „Capsule-Diagramm" erhalten bleibt.

7. Schnittstellenarchitektur: Protokolle

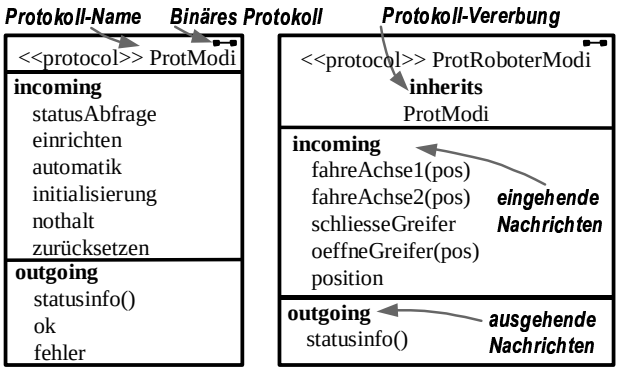

Abbildung 13: Protokolle für Wartungsmechanismen

Ein Protokoll wird als wiederverwendbare Einheit unabhängig von Komponenten spezifiziert. Es definiert Rollen, die versendbare und empfangbare Nachrichten festlegen und durch Ports implementiert werden. Zwei Ports können miteinander kommunizieren, wenn die zugeordneten Protokollrollen *kompatibel* sind, d.h. alle ausgehenden Nachrichten von der jeweils anderen Rolle empfangen werden können. In den bislang unterstützten binären Verbindungen wird das kompatible Gegenstück durch *Konjugation* – der Vertauschung von ein- und ausgehenden Nachrichten – modelliert. Im Capsule-Diagramm werden die entsprechenden Ports durch schwarze bzw. weiße Kästen modelliert.

Das Protokoll `ProtModi` in Abb. 13 legt die Wartungsdienste fest, die jede Automatisierungskomponente anzubieten hat, und definiert allgemeine Statusmeldungen. Eine Spezialisierung davon ist das Protokoll `ProtRoboterModi`, das die Schnittstelle für die Bewegungssteuerung von Robotern ergänzt und die Nachricht `Statusinfo` erweitert. Dieses Protokoll wird von der Komponente Ablaufsteuerung implementiert.

8. Layering

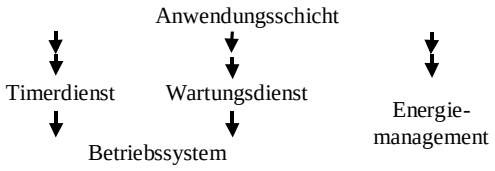

Abbildung 14: Schichtenbildung

Die Bildung expliziter Kommunikationsbeziehungen durch Konnektoren hat ihre Grenzen, wenn eine Komponente mit sehr vielen Partnern kommuniziert. Typischerweise bieten einige Komponenten Basisdienste wie etwa eine Meldungs-Konsole an, die von sehr vielen Partnern benötigt werden. Für eine übersichtliche Darstellung solcher Strukturen wird in der ROOM-Methodik die besonders in der Telekommunikation erfolgreich eingesetzte Bildung von hierarchischen *Schichten* verwendet.

Abb. 14 zeigt eine für die Beispielanwendung typische Schichtenbildung. Die bisher modellierten Komponenten sind in der Anwendungsschicht zusammengefasst, die sich auf eine – ihrerseits untergliederte – Serviceschicht stützt. Der Zugriff von Komponenten auf Dienste tieferliegender Schichten erfolgt über eine Sonderform der Ports, den *Service Access Points* (SAPs). In aktuell verfügbaren Werkzeugen wird die Schichtenbildung mittels SAPs und ihrer konjugierten Form, den *Service Provision Points* (SPPs), unterstützt. Eine diagrammatische Darstellung, z.B. mittels Capsule Diagrammen, wird jedoch nicht angeboten.

9. Verhalten: UML-RT Statecharts

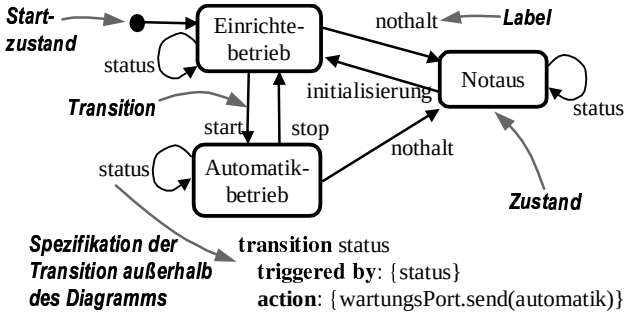

Abbildung 15: Statechart der Klasse Automatisierungskomp.

Das Verhalten von Komponenten wird in der UML-RT wie in der Standard-UML mit Statecharts beschrieben. Die benutzte Statechart-Variante hat zwar ein sehr ähnliches Aussehen, aber es ergeben sich wesentliche Unterschiede in der Modellierung. Komponenten kommunizieren in der UML-RT ausschließlich *asynchron* durch den gepufferten Austausch von Nachrichten. Der Empfänger kann die Bearbeitung einer Nachricht verzögern, und der Sender wird nicht bis zum Eintreffen einer Antwort – wie bei einem Methodenaufruf – blockiert.

Abb. 15 zeigt das UML-RT Statechart der Klasse Automatisierungskomponente, die als Ausgangspunkt der Entwicklung der Klasse Ablaufsteuerung dient. Es definiert mehrere Betriebs-Zustände. Die Transitionen sind mit Markierungen beschriftet, die dem Betrachter das intuitive Verständnis erleichtern sollen. Transitionen werden in der UML-RT sehr detailliert spezifiziert, zugunsten der Übersichtlichkeit jedoch nicht direkt im Diagramm. Die Transition `status` im Beispiel wird durch die gleichnamige Nachricht getriggert und umfasst als Aktion das Senden der Nachricht `automatik`.

UML-RT Statecharts können durch die Verfeinerung von Zuständen in Sub-Statecharts hierarchisch strukturiert werden. Die Komponente Ablaufsteuerung, die das Verhalten der Automatisierungskomponente erbt, verfeinert in Abb. 16 den Zustand `Automatikbetrieb`, um die Montageaufgabe zu realisieren.

Obwohl dieses Sub-Statechart ähnlich wie das Statechart in Abb. 8 strukturiert ist, weisen die Diagramme wesentliche Unterschiede auf: Während im normalen UML

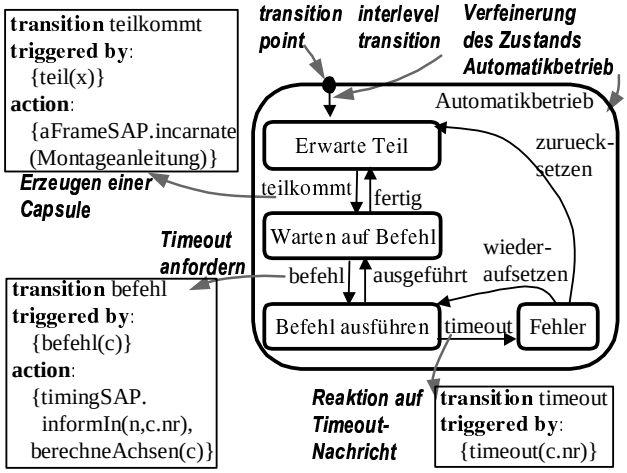

Abbildung 16: Sub-Statechart Automatik-Betrieb

Statechart die gesamte Ablauflogik in Methodenaufrufen wie `montiereC()` konzentriert ist, wird der Montageablauf hier durch einen wechselseitigen Nachrichtenaustausch realisiert. Dadurch ist die Ablaufsteuerung während der Montage nicht blockiert. Dies wird in Abb. 16 ausgenutzt, um auf Störungen der Mechanik zu reagieren: Für jede Aktion wird ein Timeout gesetzt, der nach Ablauf einer Wartezeit in einen Fehlerzustand führt, falls die Aktion noch nicht beendet ist. Dieses Beispiel macht den Einsatzbereich von Timern deutlich: Sie stellen keine Garantie für die Einhaltung von Echtzeitanforderungen dar, ermöglichen es aber, auf den Fortschritt von Zeit zu reagieren.

Das Beispiel zeigt einen typischen Einsatz von UML-RT Statecharts: Sie stellen eine abstrakte Modellierung des Komponentenverhaltens dar, während Details in einer Programmiersprache wie z.B. C realisiert werden. In Abb. 16 wird die Umsetzung von Befehlen der Montageanleitung in Positionsbefehle für die einzelnen Achsen in der Methode `BerechneAchsen` durchgeführt. Das Beispiel zeigt ein weiteres wichtiges Merkmal: *Transition points* definieren eine klare Schnittstelle zwischen Statecharts und Sub-Statecharts. Dadurch werden die Ebenen in Abb. 16 und 17 entkoppelt.

10. Zusammenfassung und Ausblick

Die UML findet immer mehr Verwendung in betrieblichen und technischen Entwicklungsprojekten. Das Sprachprofil UML-RT ermöglicht eine klare Beschreibung der Architektur technischer Anwendungen und bietet einfache Mittel zur Modellierung von Echtzeit an. Obwohl sie keine ausgeprägte Unterstützung für spezielle Anforderungen wie z.B. regelungstechnische Aufgaben bietet, reicht ihre Ausdrucksmächtigkeit für viele Anwendungen aus.

UML-RT zeigt, dass die UML durch ihre Erweiterbarkeit ausreichend Potential bietet, sich auch in technischen Anwendungsfeldern als standardisierte Modellierungssprache zu etablieren. Auf Basis der heutigen Form der UML-RT ist die Definition weiterer, spezifischer Anwendungsprofile denkbar. So wird derzeit intensiv an Anwendungsprofilen der UML für die Automobil-Industrie gearbeitet [27,28]. Anwendern bietet sich durch die dynamische Weiterentwicklung der UML die Chance, auf die Entwicklung im Sinne ihrer Bedürfnisse Einfluss zu nehmen. Weiterentwicklungen sind auch bei der Integration von Scheduling-Strategien in Modelle denkbar, etwa durch die Zuordnung von Komponenten zu Threads, wie sie in [21] zur Verfügung steht.

Es muss aber auch festgestellt werden, dass der UML wie auch der UML-RT in vielen Fällen elaborierte Analyse- und Validierungstechniken fehlen. Beispielsweise sind die Verhaltensmodelle der UML-RT und der Standard-UML bislang nicht durchgängig integriert, und es fehlen methodische Richtlinien für die Anwendung der Sprachmittel. Andererseits wird gerade in Europa an der Fundierung der UML intensiv geforscht. Gelingt eine präzise mathematische Fundierung zugleich mit der Integration weiterer Anwendungsprofile, kann die UML einen erheblichen Beitrag zur Interoperabilität in der Entwicklung, auch über Fachdisziplinen hinweg, leisten.

Danksagung:

Für ihre Mithilfe und Anregungen bei der Erstellung dieser Beiträge zur UML und UML-RT bedanken wir uns bei Prof. Dr. M. Broy, Dr. I. Krüger, Dr. S. Kowalewski, G. Popp und W. Prenninger.

Literatur:

[19] *B. Selic, G. Gullekson und P. T. Ward*. Real-Time Object-Oriented Modeling. John Wiley and Sons, Inc., 1994.

[20] *D. Harel et al*: STATEMATE. A Working Environment for the Development of Complex Reactive Systems. IEEE Transactions on Software Engineering, 16/4, 1990.

[21] *ARTiSAN Software:* Real-Time Studio®: The Rational Alternative, Verfügbar unter http://www.artisansw.com/-whitepapers/rational.asp, 1999.

[22] *B. Selic und J. Rumbaugh*: Using UML for modelling complex real-time systems. Verfügbar unter http://www.objectime.com/ uml, April 1998.

[23] Rational Software: Rational Rose Real-Time,. http://www.rational.com/products/rose/index.jsp, 2001.

[24] *OMG – Object Management Group*: UML™ Profile for Scheduling, Performance, and Time. Verfügbar unter http://www.omg.org/techprocess/meetings/schedule/UML_Profile_for_Scheduling_RFP.html, 1999.

[25] *OMG – Object Management Group*: Response to the OMG RFP for Scheduling, Performance, and Time. Verfügbar unter http://www.omg.org/techprocess/meetings/schedule/-UML_Profile_for_Scheduling_RFP.html, 1999.

[26] *B. Selic*: Architectural Design Patterns for Embedded Software. Rational Software, Verfügbar unter http://-www.rational.com/ products/rose/whitepapers.jsp, 2000.

[27] *Forschungsverbund FORSOFT:* Automotive – Requirements-Engineering for embedded systems. Information verfügbar unter http://www.forsoft.de.

[28] *DaimlerChrysler:* Automotive-UML. Information verfügbar unter http://www.automotive-uml.de.