Erfahrungen bei der systematischen Entwicklung kleiner eingebetteter Systeme mit der COMET-Methode

Alexander Nyßen[1], Peter Müller[2], Jan Suchotzki[2], Horst Lichter[1]

1 Einleitung

Der Einsatz objektorientierter Entwicklungsmethoden im Bereich eingebetteter Systeme ist zwar seit einigen Jahren Gegenstand intensiver Forschung ([SGW94] [SR98] [Do01]) – man betrachte nur die Anstrengungen, die die OMG in diesem Zusammenhang bei der Verabschiedung ihres UML-2.0 Standards unternimmt – nach wie vor liegen aber nur wenige Erfahrungen über deren Einsatz aus der industriellen Praxis vor.

In diesem Beitrag präsentieren und diskutieren wir Erfahrungen, die wir bei der Entwicklung kleiner eingebetteter Echtzeitsysteme im ABB-Konzern mit der objektorientierten Entwicklungsmethode COMET [Go00] gewonnen haben. Die Entwicklung solcher eingebetteter Systeme wird massiv durch Ressourcenbeschränkungen bezüglich Speicherplatz (üblicherweise haben solche Systeme zwischen 0,5-64 KByte RAM und 32-256 KByte ROM), Stromverbrauch und Rechenzeit beeinflusst, die sich durch die speziellen Einsatzgebiete im industriellen Umfeld ergeben.

2 Vorteile eines methodischen Vorgehens

Der Einsatz einer objektorientierten Methode stellt im Kontext der Entwicklung von kleinen ressourcenbeschränkten Echtzeitsystemen eine besondere Herausforderung dar:

- Die Entwickler solcher Software-Systeme haben traditionell einen mathematisch-naturwissenschaftlichen bzw. einen elektrotechnischen Hindergrund, sie haben in der Regel geringere Kenntnisse bezüglich der systematischen Software-Entwicklung.

- Die verwendeten Entwicklungsumgebungen sind stark geprägt von Maschinencode- und prozeduraler Programmierung.

Der Einsatz einer objektorientierten Entwicklungsmethode erfordert daher einen grundlegenden Paradigmenwechsel. Zudem ist ein zusätzlicher nicht zu unterschätzender Einarbeitungsaufwand notwendig, wenn UML – wie bei der COMET-Methode - als Notation für Analyse- und Design-Dokumente eingesetzt werden soll. Betrachtet man aber den in den letzten Jahren stetig steigenden Funktionsumfang und – dadurch bedingt – die stetig steigende Komplexität der Software in diesem Bereich, so ist die Notwendigkeit, systematische Methoden zur Entwicklung solcher Systeme einzusetzen, immanent. Wir sehen die folgenden Vorteile beim Einsatz einer objektorientierten Entwicklungsmethode:

- Das systematische Vorgehen erlaubt es, die steigende Komplexität zu beherrschen.
- Durch die objektorientierte Modellierung kann der Grad der Wiederverwendung gesteigert und die Wartbarkeit der Software verbessert werden.
- Der Einsatz der UML als Notation ermöglicht, Modellierungswerkzeuge einzusetzen und Analyse- und Design-Dokumente in standardisierter Form zwischen Ingenieuren verschiedener Standorte des Unternehmens auszutauschen.

Nachfolgend wollen wir die Erfahrungen beleuchten, die wir beim Einsatz der COMET-Methode zur Entwicklung von kleinen eingebetteten Echtzeitsystemen gewonnen haben.

3 Iterative Entwicklung - Prototyping

Die Entscheidung, die COMET-Methode für die Entwicklung von Echtzeit-Software einzusetzen, wurde getroffen, da sie sehr systematisch und zielgerichtet ausgelegt ist und die speziellen Anforderungen eingebetteter Systeme explizit berücksichtigt. Wie in den Anforderungen an die COMET-Methode formuliert, wurde sie innerhalb eines bereits in der Organisation etablierten iterativen Entwicklungsprozesses eingesetzt (und deckt dort die Analyse- und Design-Phase ab).

Wenngleich der iterative Charakter der Methode in [Go00] betont wird, und sowohl exploratives als auch evolutionäres Prototyping adressiert werden, so hat der praktische Einsatz gezeigt, dass die Methode dazu verleitet, Prototyping erst spät einzusetzen und insbesondere Performance-Analysen erst spät durchzuführen. Dies ist unserer Meinung nach darauf zurückzuführen, dass bei der Beschreibung der Methode in [Go00] nur in den einleitenden Kapiteln auf den iterativen Charakter der Methode hingewiesen wird, und dies in der detaillierten Beschreibung der einzelnen Tätigkeiten nicht explizit berücksichtigt wird (zudem wird die Performance-Analyse als eine Tätigkeit innerhalb der COMET-Methode in [Go00] zuletzt beschrieben). Dies hat im Hinblick auf die starken Ressourcenbeschränkungen, die sich in diesem Bereich auf die Software auswirken, dazu geführt, dass Missstände bei entscheidenden nichtfunktionalen Anforderungen erst spät erkannt wurden und dann umfangreiche Gegenmaßnahmen erforderlich waren.

4 Modellierungserfahrungen

Unsere Erfahrungen mit der COMET-Methode haben gezeigt, dass bei der Modellierung kleiner eingebetteter Systeme neben der bereits beschriebenen sehr systematischen und zielgerichteten Vorgehensweise vor allem die angebotenen Strukturierungs- und Klassifizierungskriterien sehr hilfreich sind, um geeignete Analyse- und Design-Elemente zu identifizieren. Da dies aber nicht für alle Modellierungsschritte in gleichem Maße gilt und nicht alle Schritte die gleiche Praxisrelevanz besitzen, wollen wir nachfolgend die besonders relevanten hervorheben und einzeln bewerten:

- *Requirements-Modellierung mit Use-Cases:* Gegenüber der bisher eingesetzten rein textuellen Beschreibung von Anforderungen hat sich das Modellieren mit Anwendungsfällen als sehr hilfreich erwiesen, um ein besseres Verständnis der funktionalen Anforderungen zu erreichen. Die Use-Case-Diagramme und die textuellen Use-Case-Beschreibungen haben sich zudem als ein adäquates Mittel zur Kommunikation zwischen den Entwicklern verschiedener Entwicklergruppen herausgestellt. Die von COMET bereitgestellten Klassifizierungsmechanismen zum Auffinden der mit dem System interagierenden Akteure sowie die systematische Vorgehensbeschreibung haben sich in der Praxis als sehr hilfreich erwiesen.

- *Analyse (dynamische und statische Modellierung):* Das statische Modellieren der Systemumgebung und der Entitäten des Systems mit Hilfe von Klassendiagrammen nimmt unserer Meinung nach eine entscheidende Stellung innerhalb der Entwicklung ein, da dadurch bereits zentrale Bestandteile der Grobarchitektur festgelegt werden. COMET liefert hier durch die angebotenen Klassifizierungshilfen gute Unterstützung und stellt so sicher, dass wichtige Daten wie zum Beispiel Konfigurations- und Kalibierdaten, die im System verwaltet werden müssen, von Anfang an beachtet werden. Das dynamische Modellieren der in der Requirements-Modellierung identifizierten Anwendungsfälle in Form von Kollaborationsdiagrammen ist der grundlegende Schritt auf dem Weg zu einer Grobarchitektur. Hier erweisen sich vor allem die durch Stereotypen in den Kollaborationsdiagrammen ausgedrückten Objekt-Klassifizierungskriterien (z.B. Coordinator, Controler, Timer, Algorithm) als hilfreich beim nachträglichen Verständnis des Designs durch Dritte.

- *Design*: Wie schon bei der Requirements-Modellierung und der Analyse stellen sich auch beim Design die in COMET definierten Klassifizierungs- und Strukturierungskriterien als ausgesprochen hilfreich heraus. So wird vor allem das Task-Design, also das Festlegen der aktiven und passiven Objekte und das Gruppieren der Objekte in Tasks, unterstützt. Die COMET-Methode bietet hier nicht nur sehr gute Strukturierungskriterien zum Auffinden der aktiven Objekte, sondern auch Optimierungskriterien zum Gruppieren von aktiven Objekten. Durch die systematische Vorgehensweise und die aus unserer Sicht weitestgehend vollständigen Strukturierungs- und Optimierungskriterien sind auch nachträgliche Optimierungen durchführbar und bleiben jederzeit gut nachvollziehbar

Obwohl die COMET-Methode zur Entwicklung eingebetteter Systeme entworfen wurde, werden keine Aussagen darüber gemacht, wie ein objektorientiertes Design in einer prozeduralen Sprache umgesetzt werden kann. Im Umfeld von kleinen eingebetteten Echtzeitsystemen ist der Einsatz von C oder gar Assembler aber immer noch Stand der Technik. Die Praxis hat gezeigt, dass bereits früh verschiedene Aspekte beachtet werden müssen, um ein realisierbares Design für diese Plattformen zu erhalten. Generell sollte eine - je nach Zielumgebung - zu definierende Untermenge von objektorientierten Konzepten verwendet werden, um eine effiziente Umsetzung nach C zu gewährleisten. Zum Beispiel ist die Verwendung von abstrakten Klassen oder Methoden möglich, jedoch ist die Umsetzung nur auf Kosten der knappen Systemressourcen realisierbar.

Schon beim Klassendesign sollte berücksichtigt werden, ob auf dem Zielsystem ein Betriebssystem vorhanden ist oder nicht. Wenn ein Betriebssystem existiert, dann kann das Design alle Möglichkeiten eines preemtiven Schedulings nutzen (Aufteilen von Nebenläufigkeiten in verschiedene Tasks). Im Falle eines Foreground/Background Systems [BW01] müssen Abläufe und Algorithmen von Anfang an für die Abarbeitung in kurzen Stücken modelliert werden.

5 Wiederverwendung

Das mit der Objektorientierung eng verknüpfte Ziel, wiederverwendbare Software-Bausteine zu entwickeln, wird unserer Meinung nach in der COMET-Methode nicht stark genug adressiert. So bleibt offen, welche Granularität bei der Wiederverwendung anzustreben ist (z.B. Klassen, Tasks, Subsysteme) und auf welcher Ebene (Code, Analyse- oder Design-Dokumente) Wiederverwendung erreicht werden kann. Unterstützung bei der Modellierung wiederkehrender Entwurfsprobleme (z.B. Anbindung an Bussysteme wie HART, PROFIBUS) - durch entsprechende Architektur- und Entwurfsmuster - wird ebenso wenig geboten wie Unterstützung bei der Modellierung von Produktlinien - eine Technik, die in der umgebenden Hardware bereits seit einiger Zeit sehr erfolgreich eingesetzt wird. Dies führt dazu, dass der Aufbau an Erfahrung in diesem Bereich (da keine Hilfestellung geboten wird) erst mit viel Lehrgeld bezahlt werden muss.

6 Werkzeugunterstützung

Anders als viele andere objektorientierte Entwicklungsmethoden für eingebettete Systeme ([SGW94] [SR98] [Do01]) ist die COMET-Methode nicht direkt an ein Entwicklungswerkzeug eines bestimmten Anbieters gebunden. Dies ermöglicht es, aus einer breiten Palette von unabhängigen (d.h. nicht an eine bestimmte Methode gebundenen) UML-Werkzeugen wählen zu können. Bei Einsatz eines solchen unabhängigen Werkzeugs können viele Designentscheidungen wie zum Beispiel die Ausgestaltung der Inter-Task-Kommunikation (via Message-Queue des Betriebssystems, mit Hilfe von Semaphoren, etc.), die sonst durch das eingesetzte methodisch gebundene Modellierungswerkzeug fest vorgegeben werden, flexibel getroffen werden.

Diese Flexibilität wird allerdings dadurch bezahlt, dass der Code, der hierfür üblicherweise durch das Werkzeug generiert wird, per Hand erstellt werden muss. Im Bereich kleiner eingebetteter Systeme, die diesem Erfahrungsbericht zugrunde liegen, ist aber eine wie oben beschriebene Flexibilität essentiell notwendig, da die sehr starken nichtfunktionalen Anforderungen sonst kaum einzuhalten sind. Da der Umfang des sonst automatisch generierten „Gluecodes“ in einem überschaubaren Rahmen bleibt (oft werden nur 2-3 Tasks in einem System benötigt), ist die Tatsache, dass dieser Code von Hand erstellt werden muss, in der praktischen Anwendung der Methode nicht als Nachteil zu bewerten.

Was sich jedoch nachteilig auswirkt ist, dass die meisten UML-Werkzeuge, die nicht direkt eine konkrete Entwicklungsmethode für eingebettete Systeme unterstützen, entweder überhaupt nicht in der Lage sind, Code in einer prozeduralen Programmiersprache zu erzeugen (sofern es sich nicht um spezielle Real-Time-Editionen handelt) oder sich auch hier als nicht flexibel genug erweisen, um im Rahmen der COMET-Methode bei der Entwicklung solch kleiner Systeme eingesetzt werden zu können. Daher muss die Abbildung der Architektur auf die Implementierung entweder vollständig manuell durchgeführt werden oder zumindest manuell angepasst werden, was einen erheblichen Aufwand bedeutet (und zudem eine Fehlerquelle darstellt).

7 Zusammenfassung und Ausblick

Trotz einiger Probleme hat sich der Einsatz der COMET-Methode bei der Entwicklung kleiner eingebetteter Systeme als praktikabel herausgestellt. Wir haben versucht, einige essentielle Mängel herauszustellen, die als Gegenstand zukünftiger Forschungsarbeiten angesehen werden können. So denken wir, dass vor allem der Aspekt der Wiederverwendung durch ein Umstellen der Methode auf den kommenden UML-2.0 Standard oder das Aufbauen einer Sammlung von Mustern für wiederkehrende Entwurfsprobleme deutlich verbessert werden könnte. Ebenfalls sehen wir Bedarf für UML-basierte Modellierungswerkzeuge, die den Bedürfnissen kleiner eingebetteter Systeme durch mehr Flexibilität bei der Code-Generierung – und vielleicht auch durch eine explizite Unterstützung der COMET-Methode – besser Rechnung tragen.

Literatur

[Go00] Gomaa, Hassan: Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison Wesley - Object Technology Series, 2000.

[SGW94] Selic, Bran; Gullekson, Garth; Ward, Paul T.: Real-Time Object-Oriented Modeling, Wiley Professional Computing, 1994.

[SR98] Selic, Bran; Rumbaugh, Jim: Using UML for Modeling Complex Real-Time Systems, http://www.rational.com/products/whitepapers/UML-rt.pdf, 1998.

[Do01] Douglass, Bruce P.: Doing Hard Time - Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, Addison Wesley - Object Technology Series, 2001.

[BW01] Burns, Alan; Wellings, Andy: Real-Time Systems and Programming Languages, Addison Wesley Longmain, 2001.

[1] RWTH Aachen, Lehr- und Forschungsgebiet Informatik III, Ahornstraße 55, D-72074 Aachen, Tel.: +49 (241) 80-21340, E-Mail: {any|lichter}@informatik.rwth-aachen.de

[2] ABB Corporate Research, Wallstadter Straße 59, D-68526 Ladenburg,
Tel.: +49 (6203) 71-6223, E-Mail: {peter.o.mueller|jan.suchotzki}@de.abb.com