4. Workshop Automatisierungstechnische Verfahren für die Medizin vom 26. bis 27. März 2003 in Karlsruhe

„Implementierung einer modernen Softwarearchitektur am Beispiel des Medizinroboters“

M. Götz, B. Gutmann
INNOMEDIC Gesellschaft für innovative Medizintechnik und Informatik mbR, Herxheim, Deutschland
E-Mail: mario.goetz@innomedic.de

R. de Klerk
ITK Consulting Ingenieurbüro für technische Kybernetik, Herxheim, Deutschland

Band: AUTOMED 2003 - 4 . Workshop "Automatisierungstechnische Methoden und Systeme für die Medizin"
Editors: U. Voges, G. Bretthauer
ISSN: 0947-8620
Pages: 28-29

Implementierung einer modernen Softwarearchitektur am Beispiel des Medizinroboters MIRA

M. Götz[1], B. Gutmann[1], R. de Klerk[2]

[1]INNOMEDIC Gesellschaft für innovative Medizintechnik und Informatik mbH
Luitpoldstr. 59, 76863 Herxheim
[2]ITK Consulting Ingenieurbüro für technische Kybernetik
Luitpoldstr. 59, 76863 Herxheim

mario.goetz@innomedic.de

MIRA ist ein Manipulator für die interventionelle Radiologie[Gutmann2002], den die Firma Innomedic GmbH in Zusammenarbeit mit dem Forschungszentrum Karlsruhe im Rahmen eines Technologie-transferprojektes entwickelt. Der Manipulator wird für den Einsatz im Magnetresonanztomographen (MR) konzipiert. Das bedeutet u.a., dass die komplette Mechanik, Aktorik und Sensorik nicht ferromagnetisch sein darf.

Die Aufgabe von MIRA ist es, in einem CT oder MR unter Bildgebung eine Kanüle zu setzen. MIRA soll in erster medizinischer Anwendung in der periradikulären Therapie (PRT) eingesetzt werden und wird daher als Medizinroboter der Klasse 2B eingestuft.

Die Umsetzung der damit verbundenen Anforderungen an die Software soll nachfolgend an einem kleinen Beispiel, dem Auslesen eines Sensorwertes, veranschaulicht werden. Hierbei wird die schrittweise Verfeinerung bzw. Umsetzung der Anforderungen in ein Softwaremodell veranschaulicht. Zusätzlich werden die eingesetzten Sicherheitsmechanismen und deren Aufgabe erläutert.

Die Softwareentwicklung bei der Firma Innomedic GmbH basiert auf der Unified Modeling Language (UML) [Booch1998, Douglass1999-1] und einem definierten Entwicklungsprozess, der auf dem allgemeinen Unified Software Development Process [Jacobson1999, Douglass1999-2] basiert. Die UML kennt verschiedene Diagrammtypen zur relationalen, funktionalen und zeitlichen Beschreibung des Systems.

In der ersten Entwicklungsphase wird eine Analyse der Anforderungen durchgeführt, die an das zu entwickelnde System gestellt werden. Die UML stellt die Anforderungen u.A. als sogenannte Use-Cases (UC) in Use-Case-Diagrammen (UCD) dar. Abb.1 zeigt einen Ausschnitt des Systemüberblicks für MIRA, der den Arzt als Akteur und einige große Anwendungsfälle enthält.

Die Use-Cases werden durch Ellipsen dargestellt. Diese können dann beliebig in weiteren UCDs verfeinert werden.

Abb.2 zeigt ausschnittsweise die Verfeinerung des UCs „Perform Intervention". Der für das gewählte Beispiel relevante UC „MoveToEntryPoint" hat zusätzlich eine „Usage"-Relation zum UC „MoveAxis".

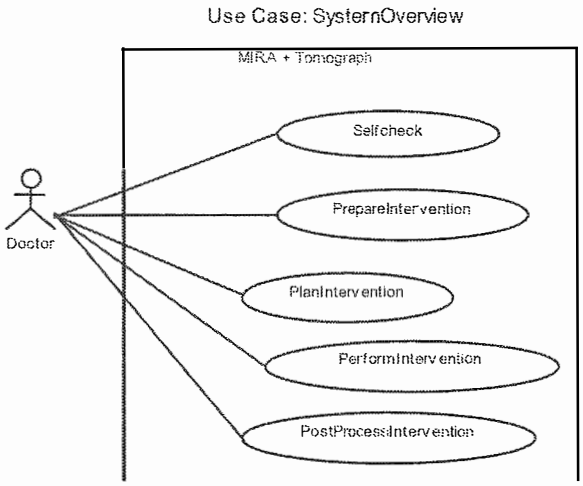

Abb.1 UCD „System Overview"

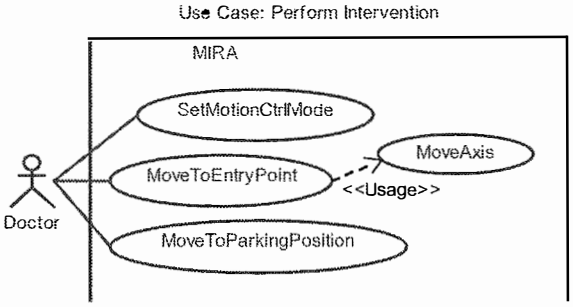

Abb.2: UCD „Perform Intervention"

In der zweiten Stufe der Softwareentwicklung wird das so erstellte Analysemodell in ein Designmodell umgesetzt. Hierzu werden in der UML die sogenannten Object Model Diagramme (OMD) verwendet. Abb.3 zeigt ein einfaches OMD der Klasse „Axis", deren Aufgabe die Achssteuerung ist. Sie hat in diesem Fall Assoziationen zu den Klassen „WriteActor" und „ReadSensor".

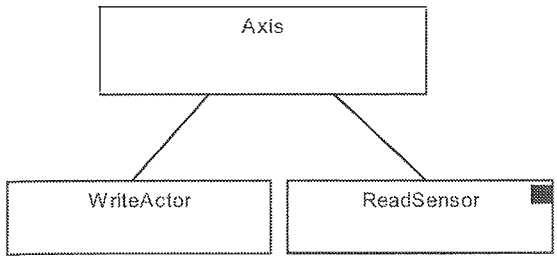

Abb.3: OMD der Klasse „Axis“

Das Zustandsverhalten einer Klasse wird in der UML über Statecharts (SC) modelliert. Abb.4 zeigt das SC der Klasse „ReadSensor“.

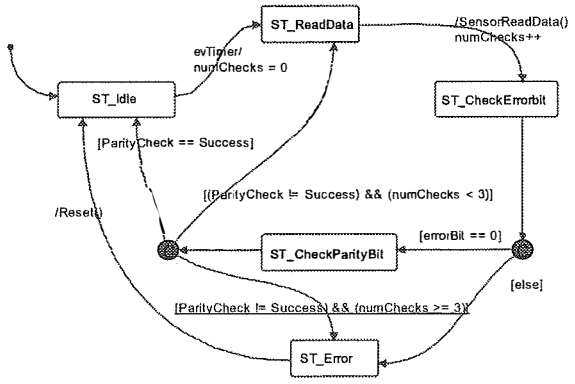

Abb4.: SC der Klasse „ReadSensor“

Die aktuellen Sensorwerte in unserem Fall werden timergesteuert (erweitertes SSI-Protokoll) ausgelesen. Nach dem Auslesen werden die Statusbits „parityBit“ und „errorBit“ überprüft. Im Falle eines Übertragungsfehlers (ParityCheck() != Success) kann der Vorgang bis zu drei mal wiederholt werden, ansonsten wird ein Fehler angezeigt. Ist das errorBit gesetzt, liegt ein LWL-Bruch vor.

Dies stellt den Sicherheitsmechanismus auf unterster Softwareebene dar. Zusätzlich können relevante Daten, wie hier der Sensorwert, invertiert gespiegelt abgelegt werden, um Speicherfehler zu entdecken. Der Zugriff auf gemeinsam genutzte Daten in einer Multitaskingumgebung ist semaphorgeschützt.

Übergeordnet wird in der Klasse „Axis“ eine Plausibilitätsprüfung des gelesenen Sensorwertes durchgeführt.

Die Systemintegrität kann in sicherheitsrelevanten Systemen mittels SW- und/oder HW-Watchdogs überprüft werden. Sie führen Alive-Checks und zusätzliche Plausibilitätsprüfungen durch.

Das erstellte Design-Modell wird in der anschließenden Implementierungsphase weiter verfeinert und Schritt für Schritt (optional toolgestützt) umgesetzt. Anschließend wird die Umsetzung der spezifizierten Funktionalität in der Testphase überprüft.

Als letzte Phase folgt dann die Validierung im realen Einsatz.

Die Verwendung der UML ermöglicht eine übersichtliche und konsistente Software-Entwicklung. Durch den Einsatz von Case-Tools wird neben der Erstellung der Diagramme noch die Dokumentation der Software erleichtert.

Um jedoch eine gleichbleibende Software-Qualität über den gesamten Entwicklungsprozess zu ermöglichen, müssen auch im Prozess selbst qualitätssichernde Maßnahmen umgesetzt werden. Hierzu werden beispielsweise am Ende jeder Entwicklungsphase Reviews durchgeführt, um das erstellte Konzept sowie die korrekte Umsetzung der Spezifikation zu überprüfen.

Die Einhaltung der Programmierrichtlinien werden durch Code-Reading und Tools für statische Softwaretests [Spillner2003] durchgeführt.

Die Verifikation der Software wird dann mittels dynamischer Tests [Spillner2003] durchgeführt. Hierbei wird mittels Blackbox- und Whitebox-Tests die Funktionalität der Software in einem dokumentierten, objektiven und nachvollziehbaren Verfahren überprüft, das dann abschließend als Basis für die Zulassung dient.

LITERATURHINWEISE

[Booch1998]
G. Booch, I. Jacobson, J. Rumbaugh, „*UML konzentriert*“ Addison-Wesley 1998

[Douglass1999-1]
B.P. Douglass, „*Doing Hard Time*“ Addision Wesley Longman, 1999

[Douglass1999-2]
B.P. Douglass, „*Realtime UML*“ Addision Wesley Longman, 1999

[Gutmann2002]
B. Gutmann et al., „MR/CT compatible robotics for image guided procedures“ SMIT 2002, Conference of the society for Medical Innovation and Technologie, Olso, Norway, September 5-7th, 2002

[Jacobson1999]
I. Jacobson, G. Booch, J. Rumbaugh, „*The Unified Software Development Process*“ Addision Wesley Longman, 1999

[Spillner2003]
A. Spillner, Tilo Linz, „*Basiswissen Softwaretest*“ dpunkt Verlag, 2003