Modellbasiert zum Seriencode

Zwei Projekte, zwei Tools, viele Erkenntnisse

Joachim Terasa, coming GmbH

Zwei Werkzeuge für MBSE, die sich durch ihre Eigenheiten und einzigartigen Features unterscheiden, haben dabei aber auf den ersten Blick eine Menge Gemeinsamkeiten. In der Praxis entpuppen sich einige als durchaus relevant. Im den hier betrachteten Fällen sind besonders die „polled Statemachines“ gegenüber den „eventgetriebenen Statemachines“ die hervorstechendsten Unterschiede, die zu einem nicht vergleichbaren Verhalten des generierten embedded Systems führen.

MBSE? - MBSE!

Modellbasiertes Softwareentwicklung für Embedded Systeme ist ein Fakt und seit vielen Jahren praktizierte Realität.

Konventionell - Modellbasiert

Bei der konventionellen Entwicklung, also z.B. Codieren in C/C++, gab es immer schon die notwendige Entscheidung für eine Toolchain, wie z.B.:
Versionsverwaltung / Editor / Compiler / Linker / Debughilfsmittel / Testhilfsmittel
Das Glied der Kette, das hier im Fokus steht, ist der Editor.

Vom Editor zum Modellierungswerkzeug

Die Wahl eines geeigneten Editors ist nicht kritisch, seine Benutzung führt immer zu dem einen definierten Ergebnis: dem Sourcecode. Dessen Inhalt bestimmt der Programmierer.
Beim Einsatz eine Modellierungswerkzeugs wird auch innerhalb des Modells manuell erstellter Sourcecode verwendet. Auch wenn sich Werkzeuge gemeinsam an Standards, wie UML orientieren, sind die Spielräume, die diese Standards zulassen, groß, wenn es darum geht aus grafischen Modellbestandteilen Code zu erzeugen.
Anhand zweier Beispiele von Modellierungswerkzeugen soll gezeigt werden, dass oberflächliche Ähnlichkeit nicht ausreicht. Aufgrund umfangreicher Praxiserfahrung in mehreren Projekten der vergangenen 5 Jahre stehen sich diese beiden gegenüber:

radCASE / Hersteller: IMACS GmbH, Deutschland / www.radcase.com

QP/QM-Framework / Hersteller: Quantum Leaps, LLC, USA/ www.state-machine.com

Gemeinsamkeiten

Beide…

- sind Modellierungswerkzeuge für embedded Systeme, die aus dem Modell C-Code erzeugen, der direkt targetfähig ist.
- basieren auf der UML und verwenden als wesentlichen codeerzeugenden grafischen Bestandteil hierarchische Statemachines.
- sind objektorientiert

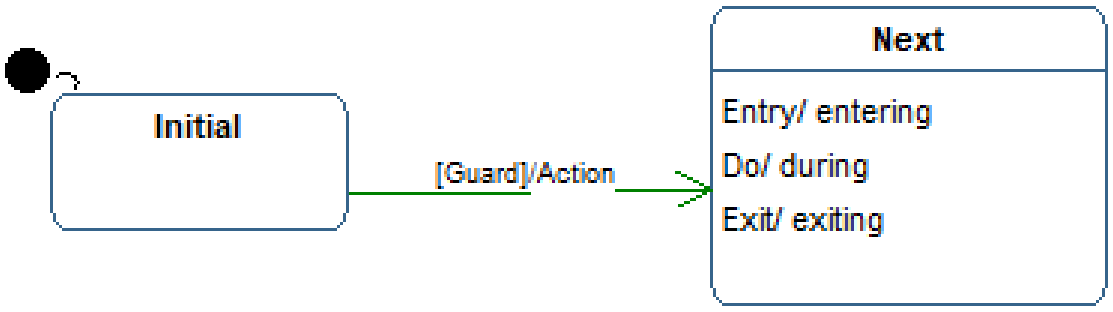
- unterstützen eine Form von Tasking.
- bieten die Möglichkeit, Projekte zu stückeln.
- bieten die Möglichkeit auf dem PC eine Simulationsumgebung einzurichten.
- unterstützen Target-Debugging.

Detaillierter Vergleich der „Gemeinsamkeiten“
Die oben genannten Gemeinsamkeiten sind oberflächlich, wären aber bei einem Auswahlverfahren mittels einer Featureliste durchaus ausreichend aussagefähig

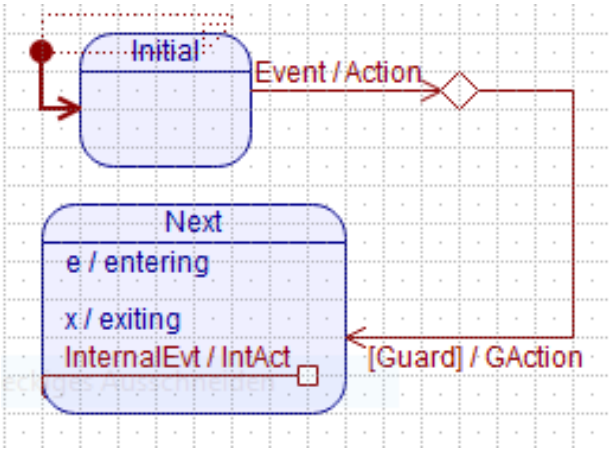
Wie spricht die Software mit der Hardware?
Stichworte: Anbindung an die Hardware / Zugriff auf IO-Ressourcen / mehrfache Instanzen einer Klasse
radCASE verlangt zu jeder Hardware zwingend eine API, die zur Applikation hin eine fest definierte Schnittstelle vorweist. Damit ist das Modell auf jede beliebige Hardware mit existenter API übertragbar.
Die Zuordnung von Hardwareressourcen erfolgt außerhalb der eigentlichen Software in der „Entity-Tab“. Hierin werden logische Ressourcen innerhalb der Software und der Instanzen mit physikalischen Ressourcen verbunden. Die Anpassung an Hardwareänderungen ist allein durch Anpassung der Entity-Tab, ohne Änderungen am Code möglich.

Für QP muss ein BSP erstellt werden. Dieses hat keine vorgegebene feste Form oder Schnittstelle.
Jedes Objekt muss auf eine maximale Zahl von Instanzen ausgelegt sein. Anhand einer Instanznummer wird der Zugriff auf physikalische Ressourcen unterschieden. Eine Hardwareänderung hat auch eine SW-Änderung zur Folge.

Wie arbeiten die Statemachines?
Laut UML beinhaltet ein Zustand eine Entry- und eine Exit-Action. Eine Transition hat ein auslösendes Event, eine Schutzbedingung und eine Aktion (Event[Guard]/Action). Beide Werkzeuge weichen hiervon ab.



Bei radCASE kennt in Zuständen zusätzlich eine Do-Action und bei den Transitions lediglich [Guard]/Action. Actions sind C-Codefragmente, die mit der Codegenerierung in den Code eingefügt werden.



Bei QP/QM hat eine Transition Event/Action, was sich allerdings mit einem einfachen Kunstgriff auf UML-Form Event[Guard]/Action erweitern lässt. Außerdem gibt es hier sogenannte interne Transitions. Actions sind auch hier C-Codefragmente, die mit der Codegenerierung in lokale Unterprogramme abgebildet werden.

Die zugrundeliegenden Laufzeitsysteme sind grundsätzlich unterschiedlich.
Statemachines in radCASE pollen anhand der [Guards] zyklisch auf Zustandsänderungen. Die Do-Action innerhalb eines aktiven Zustands wird zyklisch ausgeführt. Für jedes Objekt dürfen auch mehrere Statemachines existieren. Das System hat durch das Polling immer eine Grundlast.
QP setzt auf ein voll asynchrones Konzept. Events, das sind Nachrichten, die neben einer Kennung auch Inhalte enthalten, werden entweder systemweit oder gezielt verschickt. Empänger ist immer die Statemachine eines AO, ohne die sich im nichts bewegt. Die internen Transitions führen nicht zu einem Zustandswechsel, sondern führen eine lokale Aktion aus. Im Laufzeitsystem ist ein aufwändiger Event-Queue-Mechanismus enthalten. Eine Statemachine tut ohne Event nichts.

Objektorientierung
Beide Werkzeuge verwenden zur Definition von Klassen eine textuelle Baumstruktur.

QP/QM erlaubt hier für Methoden und Attributen alles, was der verwendete Compiler erlaubt. Außerdem muss eine Statemachine Bestandteil einer Klasse sein. Methoden sind dabei Unterprogramme oder Funktionen in C. Die Instanzen einer Klasse nennen sich „Active Objects“ und sind die kleinsten Einheiten des Tasking in QP.

radCASE verfolgt für Methoden die üblichen Sichtbarkeitstypen und zusätzlich den Typ „Permanent“, der die betreffende Methode in eine Liste von zyklisch auszuführenden Methoden einfügt. Diese sind Basis des kooperativen Tasking in radCASE.
Für Attribute wird ein eigenes Typkonzept verwendet. Jede Variable („Element“) beinhaltet Zusatzinformationen über ihre Lokalisierung in Code- oder Speichersegmenten, d.h. ob sie unter anderen konstant, persistent, analog IO, digital IO oder variabel sind. Alle Typen verfügen über Metadaten, wie Einheit,

Darstellungsformat, Ober- und Untergrenze, Beschreibung und Kommentar um komfortabel in GUI-Bausteinen verwendet zu werden.

Tasking
radCASE setzt auf ein Round-Robin-Konzept, bei dem alle als „Permanent“ markierten Methoden zyklisch aktiviert werden. Statemachines werden hier identisch wie Methoden behandelt. Unterschiedliche Zykluszeiten sind möglich. Ein kooperatives Tasking ist die Grundlage und muss bei der Implementierung der Methoden berücksichtigt werden.

Die kleinste Einheit des Taskings bei QP ist das Active Object, bei dem die Statemachine zentrales steuerndes Element ist, ohne das nichts läuft. Allein aufgrund eines zu bedienenden Events werden Aktionen durchgeführt, Events generiert, Berechnungen durchgeführt. Jedes AO erhält dabei eine eindeutige Priorität, die über die Reihenfolge der Abarbeitung von Events entscheidet.

GUI
Beide Werkzeuge bieten die Möglichkeit einer Entwicklung mit Simulation unter Zuhilfenahme einer GUI am PC.
QP/QM setzt da auf die Möglichkeiten von VC++ mit einer C-API zur Anbindung an die Targetsoftware. Auf diese Weise lässt sich die komplette Zielhardware simulieren, mit Anzeige von Variablen, Ausgängen und Ansteuerung von Eingängen. Anhand der Simulation lässt sich die Applikation austesten.

radCASE geht hier einen Schritt weiter. Es unterstützt lokale Displays am Controller mit diversen Steuer- und Bedienmechanismen. Remote-GUIs für PC oder Web werden unterstütz. Diese können für eine Simulation oder als Visualisierung und Bedienoberfläche über eine Datenverbindung im laufenden Betrieb genutzt werden. Diese GUI-Elemente sind jeweils Bestandteil der Klassen. Jedes GUI Element wird mit dem zugehörigen Objekt instanziiert und wird über eine übergeordnete GUI dargestellt, ohne dass irgendetwas kopiert oder modifiziert werden muss. Ohne GUI steht eine Standard- Oberfläche mit Zugriff auf alle Variablen, IO-Werten und animierten Statemachines zur Verfügung.

Debugging
Bei beiden Werkzeugen basieren die eingebauten Debugging-Möglichkeiten auf der Kommunikation mit einem externen Rechner.

radCASE greift zum Debugging auf die umfangreichen GUI-Funktionen mit triggerbare Record/Replay-Funktion zurück. Diese Funktion läuft vollständig auf dem PC. Fehlersituationen lassen sich gezielt einkreisen.

QP bedient sich eine Zusatzpaketes namens QSpy, das auf dem Target Events und Zustandsübergänge an einen mit einem QSpy-Client ausgestatteten PC meldet. Es lassen sich benutzerdefinierte Ausgaben einfügen. Das ganze wird von dem Client lesbar mit timestamps ausgegeben. Dieser Trace lässt sich zur Compilezeit nach diversen Kriterien filtern. Trotzdem ist die Analyse aufgrund der immer noch immensen Datenmengen aufwändig.

Stückelung
radCASE verwendet dazu „Bibliotheken“, die typischerweise Typ- oder Klassendefinitionen enthalten. Bibliotheken werden per „include“ eingebunden.
QP/QM bietet die Möglichkeit ein Package in ein „External Package“ auszulagern. Die kleinste auszulagernde Einheit ist damit ein Package.

Fazit
QP/QM zeigt sich als auf Geschwindigkeit optimierte Umgebung, vollständig aber ohne viel Zusatzkomfort. radCASE hat lediglich polles Statemachines, weist aber diverse komfortable Funktionen auf, die einen Entwicklungsprozess stark vereinfachen könenn.

Autor
Joachim Terasa studierte Elektrotechnik/Informationsverarbeitung an der FH Aachen.
Von Anfang an hat er sich den „Embedded Systemen“ verschrieben, die freilich damals anders hießen. Schon frühzeitig in seinem Berufsleben waren methodisches Vorgehen, CASE-Tools und modellbasierte Entwicklung ein Thema.
1994 gründete er mit 3 Kollegen die Softwaredienstleistungsfirma coming GmbH in Ottobrunn und ist seither dort als Geschäftsführer und im normalen Projektgeschäft als Entwickler/Projektleiter tätig.
Schwerpunktmäßig beschäftigt er sich mit hardware- und systemnaher Softwareentwicklung, auch unter Einsatz von Modellierungstools.