

Inhalte

- Preis-Aktion bei KEIL ARM: Reduzierte ARM Tools - erhältlich bis zum 16.12.2008
mehr Infos finden Sie unter - Redaktionelles -
- Artikel - Anforderungen im Embedded Software Engineering
- Tech Tipps - Multiplizität bei statischen Objekten - The Magic of BaseNumberOfInstances
- Nützliche Online Foren
- Tool News - Gleich 2 neue Versionen von Rhapsody®
- Redaktionelles
- Aktuelle Termine

Anforderungen im Embedded Software Engineering

Im Embedded - Software - Engineering sind im Grunde heute die gleichen Anforderungen aktuell, wie sie schon vor 10 Jahren diskutiert wurden, der einzige Unterschied:

Heute sind sie noch akuter, die Probleme sind noch offensichtlicher, Zeitdruck und/oder Qualitätseinbußen sind noch einmal gestiegen.

Was wäre also, wenn

...Embedded-Software wirklich modular aufgebaut wäre?

...wenig Abhängigkeiten und Kopplungen innerhalb der Software bestehen würden?

...Änderungen und Erweiterungen leicht zu implementieren wären?

...Komponenten einfach wieder verwendet werden können?

...die abgelieferte Softwarequalität reproduzierbar gleich gut ist?

...Dokumentation aktuell und intuitiv verstehbar ist?

Die gegenwärtige Situation sieht oft anders aus. Entwickler haben „Bauchschmerzen“, wenn von neuen Funktionen oder Änderungen in einer „gewachsenen“ Software die Rede ist, weil

...niemand weiß, wie sich bei Änderungen in der Software Systemverhalten ändert.

...die ursprüngliche Struktur und damit der Überblick der Software verloren gegangen ist.

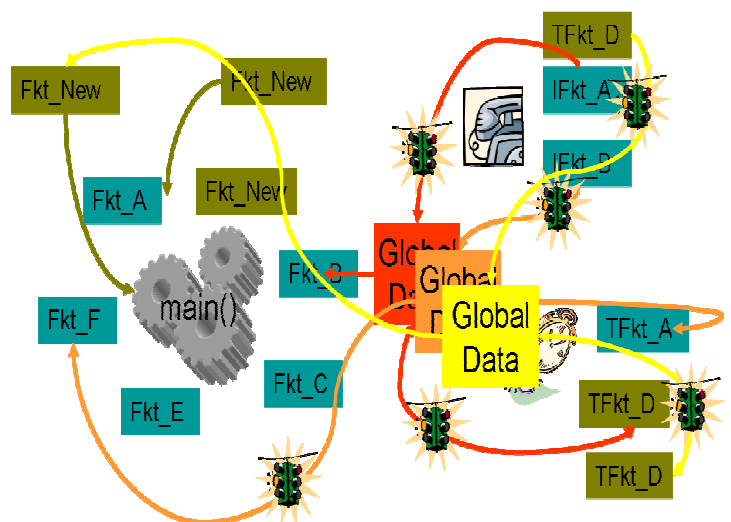
...sich Laufzeiten ändern, Daten inkonsistent werden und Prioritäten verschieben.

...externe oder ausgeschiedene Mitarbeiter nicht verfügbar sind.

...neue Mitarbeiter langwierig in den Strukturen eingearbeitet werden müssen.

...der Termindruck so groß ist, dass tiefgründige Analysen nicht akzeptabel sind.

Zu selten werden Software-Projekte auf Basis eines neuen Architektur-Designs entwickelt. Ein vor Jahren entwickeltes Design lebt bis heute und ist inzwischen an allen Ecken und Kanten erweitert, nachgebessert, umgeschrieben.



Auf Basis solcher Quellen kann nicht mehr effizient genug entwickelt werden. Änderungen an einer

Funktion wirken sich ungewollt in anderen Bereichen der Applikation aus.

Um die gewünschte Qualität unter diesen Bedingungen aufrecht zu erhalten erhöht sich der Testaufwand. Und das unter permanent steigendem Zeitdruck.

Gleichzeitig wünschen Produkt-Management und/oder Vertrieb, dass neue Funktionen am Besten gestern implementiert wurden, um die geplanten Verkaufszahlen erreichen zu können.

Warum aber werden notwendige Schritte, um diesen Zustand zu verbessern, von einem Projekt zum nächsten immer wieder vor sich her geschoben?

Eine Zeit lang waren die besonderen Bedingungen von Embedded - Projekten ein Grund:

...Prozedurale Methoden vs. (dynamischen!) objektorientierten Ansätzen

...wenige Speicher - Ressourcen, kein „Overhead“ akzeptabel

...harte Echtzeit, Integration von kritischen Interrupt - Service - Routinen und Treibern

...Interrupt - Prioritäten und Systemverhalten

...nicht vorhandene Tool-Chain, kein Target-Remote - Debug auf Source - Level möglich

Das es inzwischen praktikable Software Engineering Lösungen auch für den Einsatz in Embedded - Projekten gibt und wie diese aussehen, davon handelt dieser Beitrag.

Ansätze im Embedded Software Engineering

Wollen Sie das Softwaredesign für Ihre Systeme oder Projekte überarbeiten? Denken Sie über eine ganzheitliche Softwarearchitektur und den Einsatz bewährter Methoden und Tools nach?

Achten Sie auf die Schwerpunkte:

- Architekturdesign und Software-Entwurf mit Standards (z.B. UML)
- Softwarearchitektur mit:
 - Zeitlicher Entkopplung
 - Funktionaler Entkopplung
 - Daten-Entkopplung
 - Prioritäten-Entkopplung
 - Treiber/Interrupt Service Ebene

- RTOS Nutzung bei größeren Projekten

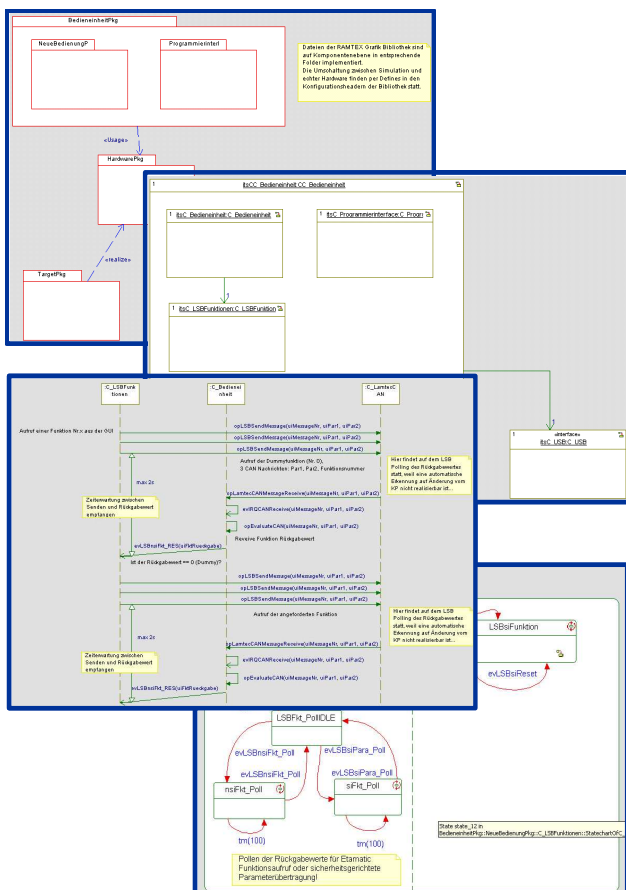
Dabei treten die konkreten einzelnen Maßnahmen und Methoden schnell in den Hintergrund. Bei der ganzheitlichen Betrachtung gewinnt die neue Philosophie. Umsetzbar ist diese mit verschiedenen Werkzeugen gleichermaßen.

Durch den Einsatz definierter Prozesse bekommen Sie ebenfalls diverse Kennzahlen, um den Reifegrad und den momentanen Zustand Ihres Projekts messen zu können. Das macht sogar Softwareentwicklung planbar und transparent!

Wie UML nicht funktioniert

Seit Jahren wird im Bereich der Embedded-Software- Entwicklung auf Basis der Notation ANSI-C programmiert. Gegenüber der davor üblichen Programmierung in Assembler hat diese so genannte Hochsprache Vorteile. Ein wesentlicher sind Sprachkonstrukte, die die so genannte strukturierte Programmierung (prozedurale Programmierung) als Engineering- Methode unterstützt, welche die Notation Assembler nicht besitzt. Das ist der Hauptgrund, warum die Softwareentwicklung auf Basis einer Hochsprache wesentlich effizienter ist, als auf Basis von Assembler.

Die Unterschiede beider Techniken sind so groß, dass wir bei dem Übergang von einem Paradigmen-Wechsel sprechen. Seit der Einführung der Hochsprache ANSI-C hat es keinen weiteren Paradigmenwechsel im Software Engineering gegeben und das ist heute überall zu spüren. Mit der Methode „Strukturierte Programmierung“ auf Basis der Notation „ANSI-C“ und den Tools „Compiler, IDE und HLL-Debugger“ sind in vielen Fällen die Anforderungen an das heutige Software-Engineering nicht mehr zu erfüllen.



Es steht ein neuer Paradigmen-Wechsel an, hin zur Methode „Objekt Orientiertes Design“ auf Basis der Notation „UML“ (Unified- Modelling- Language) und mit Hilfe von so genannten „CASE“ (Computer Aided Software Engineering) Tools.

Sie sehen schon, andere Methode, andere Notation, andere Tools und Sie denken das klingt nach einem Haufen Aufwand, nach langen Einarbeitungszeiten, nach hohen Investitionen und letztendlich nach großem Risiko. Ja, Sie haben recht und genau das ist der Grund warum dieser Schritt so gerne vor sich her geschoben wird.

Aber es gibt eine Alternative. Wie wäre es, diesen gewaltigen Schritt in kleine verdauliche Häppchen mit entsprechend kurzer Einarbeitungszeit, geringen Investitionen und kleinem Risiko aufzuteilen? Genau so wird es sehr häufig gemacht:

Ein sehr beliebtes Vorgehen dieser Art ist es die UML auf Basis eines preiswerten grafischen UML Editors erst einmal zu Dokumentations-Zwecken einzusetzen. Die Frage, die Sie sich stellen sollten ist grundsätzlich: „Löst das mein Problem?“.

Beispiel:

So hat ein Unternehmen untersucht, ob mit Hilfe der UML die aktuelle Software eines komplexen Systems, die seit ca. 15 Jahren „wächst“, grafisch zu beschreiben. Die Hoffnung war, dass die Verstehbarkeit, und somit die Erweiter- und Wartbarkeit, erhöht wird. Das Resultat nach einigen Monaten: UML ist nicht zur Beschreibung des Systems geeignet. Weiter noch: UML ist für unser Unternehmen nicht einsetzbar.

Die Analyse zeigt, dass hier mit einem andersartigen neuen Ansatz auf ein bereits feststehendes Ergebnis (prozeduraler Code) gezielt wurde, ohne weitere Randbedingungen zu betrachten. Die Gesamtsicht fehlt. Hat man ein eckiges Ei und versucht eine Henne darauf zu adaptieren, welches Resultat ist zu erwarten? Nun könnte man daraus schließen, dass Hennen zum Eier legen nicht geeignet sind...

Nun hört man häufig das Argument: „Aber auf diese Weise können wir erst einmal die UML erlernen und Erfahrungen mit ihr sammeln. Später kann dann ein weiterer Schritt folgen.“ Leider funktioniert das nicht. Ein Vergleich mit dem letzten Paradigmenwechsel macht es offensichtlich. Stellen Sie sich vor, man hätte Ihnen zum Erlernen der Hochsprache ANSI-C erst einmal ein ANSI-C Handbuch und einen Editor, aber noch keinen Compiler zur Verfügung gestellt. Nun hätten Sie ja fleißig in C programmieren können, bzw. in Form von Pseudo Code Ihre Assembler- Sourcen besser dokumentieren können. (Das wurde übrigens auch damals häufig versucht). Und wie aktuell wurde die Dokumentation gehalten?

Aber wenn wir realistisch sind, Sie haben nicht die geringste Chance das Programmieren zu erlernen

bzw. Erfahrungen zu machen ohne einen Compiler und der Möglichkeit das Programm auszuführen.

Warum sollte es in UML anders sein? Die UML ist eine genau so exakte Notation, wie die Hochsprache ANSI-C und für eine erfolgreiche Einarbeitung benötigen Sie genau so Feedback wie für eine Hochsprache.

Wie die UML funktioniert

Wenn wir uns die realen Requirements im Softwareengineering ansehen, dann gilt es, Arbeiten, die auf Grund von Redundanzen im Verlauf der Entwicklung Änderungen an mehreren Stellen (Dokumenten) nach sich ziehen, so gering wie möglich zu halten.

Die Erfahrung von weit mehr als 50 Projekten, in denen die Einführung der UML begleitet wurde zeigt, dass aus diesem Grund nur der konsequente Einsatz der UML zur Modellierung mit anschließender Codegenerierung langfristig erfolgreich ist. Gleichzeitig wird Reverse Engineering benötigt, um existierenden Code nicht wegschmeißen zu müssen, und sogenanntes Round Trip-Engineering, um bei bestimmten Gelegenheiten auch einmal schnell auf der C-Ebene Änderungen durchführen zu können. Automatisches Round-Tripping sorgt dafür, dass diese Änderungen auch auf Modellebene nachgezogen werden.

Um noch genauer verstehen zu können, dass nur dieser konsequente Ansatz zu Erfolg führen kann möchte ich exemplarisch auf einen kleinen Auszug aktueller Anforderungen und deren Lösung eingehen.

Bessere Dokumentation

Blicken wir noch einmal auf das offensichtliche Problem schlecht dokumentierter Software. Die Frage ist nun was ist das eigentliche Problem.

Ich behaupte nicht, dass von den Entwicklern nicht dokumentiert würde, sondern dass mit den heutigen Vorgehensweisen und Tools die Dokumentation tendenziell veraltet ist und nicht dem aktuellen Stand des Codes entspricht.

Im Wesentlichen liegt das daran, dass Dokumentation und Code parallel gepflegt werden müssen. Dazu fehlt in der täglichen Praxis jedoch die Zeit.

In nahezu allen Projekten wird mit der Dokumentation zum Beispiel in Form eines Pflichtenheftes gestartet. Irgendwann wird dann mit der Codierung begonnen, was nicht bedeutet, dass sich die Anforderungen ab diesem Zeitpunkt nicht mehr ändern. Diese fließen dann jedoch aus Zeitmangel nur in den Code ein und die Dokumentation veraltet. Nach einiger Zeit ist sie so veraltet, dass sie nicht mehr sinnvoll genutzt werden kann.

Der Einsatz der UML ermöglicht so genannte Vorwärts-Dokumentation. Die Arbeitsweise ändert sich grundsätzlich, da die UML gleichzeitig Modellierungs-Sprache als auch Dokumentations-Sprache ist, schmelzen Dokumentation und Modellierung zusammen. Aus dem Modell wird dann Code generiert, der dem Stand der Dokumentation entspricht.

Wiederverwendung

Wenn von Wiederverwendung gesprochen wird, dann wird zuerst immer an den C-Code gedacht. Aber es gibt viele weitere Bereiche, in denen Wiederverwendung die Effizienz steigern kann.

Stellen Sie sich vor, Sie arbeiten auf Basis eines Pflichtenheftes (Requirements). Wenn nun dieses Pflichtenheft bereits auf Basis von UML erstellt wurde und in dem Repository (Zentrale Datenbank des CASE Tools) existiert, dann können geeignete Elemente daraus direkt zur Dokumentation von Design oder Implementation genutzt werden.

Dieses geschieht natürlich nicht in Form von Copy & Past und schafft damit redundante Informationen, sondern mit einem direkten Link. Das hat noch weitere Vorteile. Werden z.B. vom Auftraggeber Änderungen gewünscht, können diese auf Basis des Pflichtenheftes durchgeführt werden. Auf Grund der Links ist sofort ersichtlich wo im Design sich evtl. Änderungen ergeben. Das hilft Pflichtenheft, Design, Code und Dokumentation in sich konsistent zu halten.

Aus diesem Vorgehen ergibt sich gleich noch die Lösung eines weiteren, häufig anzutreffenden Problems. Bei manchen unserer Kunden gibt es separate Testabteilungen. Zu einem bestimmten Zeitpunkt des Projektes bekommen diese dann einen Prototypen incl. C-Source Code und dem ursprünglichen Pflichtenheft, um mit den Tests zu starten.

Wenn diese Aufgabe gewissenhaft durchgeführt wird, werden Zustände erkannt, in denen der Prototype nicht so reagiert wie im Pflichtenheft beschrieben. In vielen Fällen beginnt jetzt erst der aufwändigste Teil der Arbeit, nämlich nachträglich herauszufinden, ob das eine geänderte Eigenschaft des Produktes ist und das Pflichtenheft an dieser Stelle veraltet ist oder ein tatsächliches Fehlverhalten.

Hier ergibt sich ein weiterer Nutzen aus der UML. Sie besitzt Diagrammtypen, mit denen sich das prinzipielle Verhalten in Form von Abläufen beschreiben lässt. In vielen Fällen können auf Basis dieser Diagramme bereits festgelegte Abläufe im Pflichtenheft (Requirements Engineering) beschrieben werden. Zu diesen Diagrammen können dann im Verlauf des Designs die Interfaces spezifiziert werden, das ermöglicht dann diese Diagramme direkt als Testfälle wieder zu verwenden. Da wir ja mit der UML eine Formale

Notation verwendet haben ist diese auch für Testtools verständlich. Die beste Voraussetzung Testabläufe zu automatisieren und das sogar auf Basis der aktuellen Requirements. (Tracability vom Requirement bis zum Test und umgekehrt ist übrigens eine der Forderungen in der Norm 61508 oder SPICE)

Verstehbarkeit

Natürlich und offensichtlich erhöht die UML als Grafische Notation die Verstehbarkeit. Ein Bild sagt mehr als tausend Worte, aber was genau ist mit Verstehbarkeit denn eigentlich gemeint? Z.B., dass Sie besser verstehen was Ihr Kollege erschaffen hat und umgekehrt. Aber es geht noch weiter. Wie zum Beispiel kommunizieren Sie derzeit mit Kollegen aus anderen Ingenieurs-Disziplinen? Wahrscheinlich auf Basis von Handskizzen auf Flipcharts. Die UML ist eine Notation, die in großen Bereichen auch von NICHT- Softwareentwicklern verstanden wird. Auf dieser Basis kann immer am aktuellen Stand des Modells kommuniziert werden.

Verstehbarkeit bedeutet auch, dass schnell ein tiefes Verständnis der Software erreicht wird. Dieses liegt sehr häufig im Bereich des so genannten Architektur-Designs. Was passiert an anderen Stellen meines Systems, wenn an dieser Stelle diese Änderung durchgeführt wird. Wie sind die Zusammenhänge? In C gibt es nur sehr wenig bis gar keine Konstrukte, die die Architektur einer Software beschreiben. Jegliches Wissen über die Architektur existiert oft nur in den Köpfen der Entwickler. Aus meiner persönlichen Sicht ist eine der größten Vorteile der UML gegenüber herkömmlichen Hochsprachen, dass sie Konstrukte zur Modellierung der Laufzeit und Kommunikationsarchitektur besitzt und vor allem das Design von Interfaces unterstützt.

Der Einsatz von UML in kleinen Embedded Systemen

Hindernisse

Keine Zeit dafür? Zu teuer? Vom Kunden abgelehnt?

Aber...

Wer wird in 5 Jahren erfolgreich sein?

Sehen Sie sich die parallelen zur Hardwareentwicklung an. Überlegen Sie einmal, wie sich Hardwareentwicklung in den letzten 15 Jahren entwickelt hat. Hardware bestand schon immer aus entkoppelten Komponenten, die sich über einen standardisierten Kommunikationspfad verständigt hat (Daten-/Adressbus, SPI, Feldbusse, etc.). Hardware wurde immer auf Erweiterbarkeit entkoppelt und modular entworfen. Hat die Leistung nicht mehr ausgereicht (im Gegensatz zur Software eine messbare Größe), wurde eine neue Plattform entwickelt. Die meisten alten Komponenten konnten wieder verwendet werden.

Die grafische Darstellung der Hardware ist standardisiert, damit jeder Entwickler verstehen kann, was in der Schaltung vor sich geht - weltweit!

Aufgrund dieser Gegebenheit wurden mächtige Tools entwickelt, die hierarchisch orientierte grafische Eingabe erlaubt, daraus Netzlisten erstellt und die physikalische Realisierung bestmöglich unterstützt und vereinfacht.

Die UML mit Code-Generator bildet sich letztendlich auf objekt-orientierte Konstrukte ab. Diese müssen in C darstellbar sein, damit die Anwendung erfolgreich ist. Da dies (je nach vorhandenen Ressourcen) nur teilweise möglich ist, ist der verwendbare UML-Vorrat nicht immer nutzbar. Folgend die wichtigsten Konstrukte:

- Klassen
- Instanziierung von Objekten
- Timer
- Mailboxen
- Events, Messages
- Ports

Einige dieser Konstrukte werden üblicherweise durch ein (Embedded-) Betriebssystem (RTOS, Real Time Operating System) zur Verfügung gestellt (Timer, Mailboxen, Events, Messages und ein Scheduler mit Taskverwaltung).

Andererseits benötigt ein Embedded-System hardwarenahe zum Teil hart-echtzeitkritische (<100µs) Reaktionen und Determinismus bei hardware-nahen Diensten. Dies betrifft:

- harte Echtzeit (unterer µs-Bereich)
- Interrupts, Interruptprioritäten
- hardware-nahe Routinen (DMA (Direct Memory Access), ADC-Wandlung, etc.)
- Determinismus

Diese Anforderungen sind in UML nicht ohne Einschränkung objektorientiert modellierbar. Ohne Verwendung von Klassen, Timern und Events/Messages wäre es theoretisch möglich die UML Notation trotzdem eingeschränkt zu verwenden, allerdings ist der Nutzen in Frage gestellt.

Grenzen des sinnvollen UML-Einsatzes in Embedded Systemen

Daraus folgt, dass man sehr wohl kleine Systeme mit der UML modellieren und generieren könnte, wenn auch mit Einschränkungen. Werden kritische OOP-Konstrukte weggelassen, ist sie fast beliebig anwendbar und ohne RTOS in C umsetzbar. Im Extremfall können z.B. statt Messages so genannte Guards (Bedingungen, „Wächter“) zur Steuerung von UML-Statecharts genutzt werden; Messages und Mailboxen entfallen dann (Synchrones Modell).

Eine oft genutzte Einschränkung ist der Verzicht auf Klassen und dynamischer Instanziierung von

Objekten. Stattdessen arbeiten kleine Embedded-Systeme mit statischen Objekten und Singleton-Objekten. Damit braucht das System keine dynamische Speicherverwaltung (malloc & Co), die in den meisten Embedded-Targets ohne MMU (Memory-Management-Unit) zur gefürchteten Speicherfragmentierung und damit zum sporadischen Fehlverhalten des Systems führen kann.

Bestimmte Funktionalitäten des Embedded-Systems sind in UML (noch) nicht ausreichend beschreibbar (Interrupt-Services, Interrupt-Prioritäten, Verschachtelungen und hardware-nahe Echtzeit-Routinen).

Beispielhaft ist hier einmal die harte Echtzeit untersucht:

Objekte können in UML „aktiv“ sein, sie haben zur Laufzeit einen eigenen Task/Thread. Dies ist über ein RTOS effizient darstellbar. Die maximale (Echtzeit-) Auflösung hängt stark von der Task-(Kontext)-Umschaltung des verwendeten Betriebssystems ab.

- 8051, 8-Bit, RTX51 (KEIL), bis 100µs [7]
- C166, 16-Bit, ARTX (KEIL), bis 25 µs [7]
- M16, 16-Bit, EmbOS (Segger), bis 32 µs [8]
- ARM7, 32-Bit, ARTX-ARM (KEIL), bis 5 µs [7]

Die Interrupt-Latenzzeiten in den untersuchten Betriebssystemen liegen zwischen 50µs (8051) und 1,8µs (ARM7).

Es ist nahe liegend, dass der Laufzeitanteil des RTOS keinen erheblichen Teil der verfügbaren Ressourcen verbrauchen soll. Anhand der Task-Umschaltzeiten kann abgeleitet werden, dass diese Mechanismen nur für wesentlich geringere Laufzeitanforderungen verwendet werden sollten.

Reaktionszeiten der Applikation sollten mit RTOS/ UML bei 8-Bit Mikrocontroller deutlich im Millisekunden- und beim 32-Bit Mikrocontroller im 100µs Bereich sein (zusätzlich abhängig von der Anzahl der Nebenläufigkeiten, die diese Zeiten ebenfalls noch erhöhen können).

Alle Funktionalitäten mit höherer Performance gehören normalerweise in eine Treiberschicht, meist prozedural und hardware-nah in C implementiert.

Unabhängigkeit, Wiederverwendung, Test & Co

Ziel sollte es immer sein, das Know-How eines Unternehmens, das prozessbezogen und nicht target-abhängig ist, so zu entkoppeln, dass es problemlos in neuen Plattformen wieder verwendbar und übersichtlich dokumentiert ist. Je größer die Applikation, umso mehr profitiert das

Unternehmen von einer schnellen Portierbarkeit (Abkündigung von Bauteilen, zu wenig Performance oder Ressourcen). Dies bedingt eine gute Software-Architektur, was wiederum durch UML unterstützt wird.

Die Scheu davor, eine Embedded- Software neu zu entwerfen („Redesign“) oder auf eine andere Plattform zu portieren liegt meist nicht in der Verwendung neuer Hardware (und neuer Mikrocontroller) sondern in der gesamten Software selbst.

Die entstandenen Schichten (falls überhaupt davon gesprochen werden kann) sind nicht sauber entkoppelt. Die entstandene Applikation ist viel zu eng und in direkter Form mit den darunter befindlichen Treiber-, Interrupt-, Library-Schichten verbunden und kann nicht ohne weiteres davon getrennt behandelt werden.

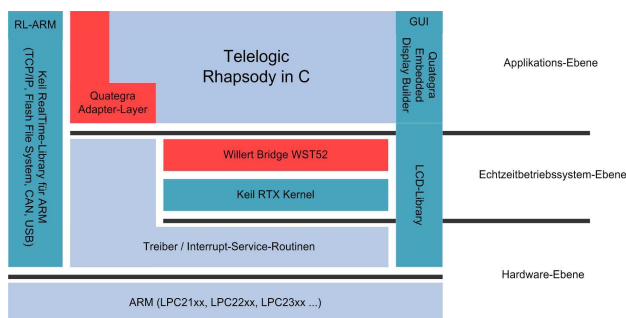
Um ein Redesign der Applikation mit UML durchführen zu können, sind weit reichende Anpassungen in allen Schichten nötig. Hier ist das Problem - und die Lösung gleichermaßen!

Eine saubere Verwendung des Schichtenmodells, ohne die Verwendung von Querbeziehungen, kann den Aufwand und die Kosten für eine solche Portierung auf ein Minimum reduzieren. Im optimalen Fall sollte dabei eine Applikation entstehen, die das Prozess-Know-How in sich vereint, jedoch plattformunabhängig ist. Eine solche Applikation kann somit nach dem Austausch der Hardware- und Echtzeitbetriebssystem-Ebene wieder verwendet werden, inklusive aller Dokumentation und erstellten Testfällen.

Chancen

Im nächsten Projekt sollen diese Chancen genutzt werden. Aber wo anfangen?

Wir orientieren uns beispielhaft an einem Referenzprojekt:



Alle Bausteine dieses Referenzprojektes sind austauschbar und geben hier ein Beispiel aus der Praxis wieder. Es findet keine Wertung statt, andere Umgebungen und Tools können dieselben Leistungen erbringen.

Als Target kommen verschiedene gängige ARM-

Architekturen zum Einsatz (STR9, LPC2378, STM32). Basierend auf diesen Targets wird die Entwicklungs-Toolchain KEIL RealView® MDK eingesetzt. Das passende Echtzeitbetriebssystem RTX-Kernel ist darin bereits enthalten und „kostet“ < 6kB ROM.

Darauf basierend kann die RealView® Real-Time Library der Firma KEIL eingesetzt werden. Diese enthält eine TCP/IP Networking Suite, ein Flash-File-System sowie CAN und USB Protokoll-Stacks. TCP/IP mit Web Server und einigen dynamischen HTML/ CGI-Seiten „kosten“ < 10kB ROM.

Treiberschichten sind Target- abhängig prozedural in C implementiert. Hardware-nahe Treiber und Interrupt-Services sind entweder in Bord-Support-Packages erhältlich oder können mit vertretbarem Aufwand neu geschrieben werden.

Die Applikationsentwicklung erfolgt in UML inklusive Code Generierung mit ‚Rhapsody® in C‘ der Firma Telelogic.

Die Adaption des generierten Codes an das RTOS auf dem ARM-Target in der RealView® MDK gewährleistet eine „Environment Bridge“ mit integrierten OSAL (Operating System Abstraction Layer) der Firma Willert Software Tools. Diese ist für viele weitere Toolchains und Targets verfügbar und gewährleistet maximale Unabhängigkeit von der Zielplattform und des eingesetzten RTOS. Diese Bridge „kostet“ <6kB ROM.

Ein „Adapter-Layer“ als Kommunikationsschicht zwischen proprietärem prozeduralem Code (Treiber, Interrupt-Service-Routinen oder Bibliotheken) und der in UML modellierten objektorientierten Applikation sichert die vollständige Target-Unabhängigkeit der Applikation. Dies „kostet“ einige hundert Byte ROM in mittelgroßen Projekten.

Simulanten...

Ist die Applikation mit dem Firmen-Know-How nun unabhängig vom Target entworfen, eröffnen sich viele Möglichkeiten. Durch „Umschalten“ auf ein anderes Target mit angepasster Adapter-Schicht ergibt sich, dass die entwickelte Applikation auch simuliert werden kann. Dazu kann z.B. der Entwurfsrechner mit Windows genutzt werden.

Das Tool Rhapsody in C bietet entsprechende Möglichkeiten, UML- Diagramme im Zusammenspiel mit höheren Betriebssystemen zu animieren und die Applikation zu testen. Dafür kann der Adapter-Layer entsprechend ausgetauscht und angepasst werden (z.B. TCP/IP via Windows Socket statt Real-Time Library, USB/CAN Umsetzer statt CAN Controller, etc) und die hardware-nahen Treiber werden umgesetzt oder simuliert. Die Applikation selber braucht nicht „berührt“ werden. Umkehrschluss: Die Applikation kann unabhängig von einem Target entworfen und (systematisch) getestet werden.

Tech Tipps:

Bei der Anwendung von Tools zur Entwicklung von Software mit Echtzeitanforderungen und/oder begrenztem Speicher ergeben sich spezifische Anforderungen, die in der Literatur, den Handbüchern oder Trainings nicht angesprochen werden. In der Rubrik Tech Tipps gehen wir auf häufig gestellte Fragen an unser Support Team ein und geben Antworten.

Tech Tipp 1:

Multiplizität bei Statischen Objekten

Problem:

Wie erzeuge ich in Rhapsody Objekte statisch, wenn Multiplizitäten genutzt werden?

Beschreibung:

Wenn man rein statisch arbeiten möchte, darf man kein malloc, also die Standard <Klasse>_create Funktionen nutzen. Wie erzeugt man nun die Objekte die man benötigt ?

Beispiel: 2 Klassen A und B mit einer gerichteten Assoziation, die die Multiplizität 8 für die Klasse B hat.

Lösung:

Man erzeugt 2 explizite Objekte für die Klassen A und B und verbindet diese mit einem Link. Zusätzlich gibt man dem B - Objekt (meist itsB) die Multiplizität 8. Nun kann man beobachten, wie im c File des dazugehörigen Packages [Funktion: <PackageName>_initRelations(void)] die Objekte angelegt werden.

Tech Tipp 2:

The magic of BaseNumberOfInstances

One property in Rhapsody in C led to some confusion in the past. I'm talking about BaseNumberOfInstances, which can be found under CG::Event. First, let's have a look at Telelogic's description of that property:

The BaseNumberOfInstances property is a string that specifies the size of the local heap memory pool allocated for either:

- * Instances of the class (CPP_CG::Class)
- * Instances of the event (CPP_CG::Event)

This property provides support for static

architectures found in hard real-time and safety-critical systems without memory management capabilities during run time. All instances of events are dynamically allocated during initialization.

Once allocated, a thread's event queue remains static in size.

Triggered operations use the properties defined for events.

When the memory pool is exhausted, an additional amount, specified by the AdditionalNumberOfInstances property, is allocated.

Memory pools for classes can be used only with the Flat statechart implementation scheme.

The possible values are as follows:

* An empty string (blank) - Memory is always dynamically allocated.

* n (positive integer) - An array is allocated in this size for instances.

We need to be careful with that description. In Rhapsody® in C there is only one BaseNumberOfInstances property available for events. And not under <lang>_CG, but under CG::Event::BaseNumberOfInstances.

However, in some earlier WST releases (V3.x) BaseNumberOfInstances also found its way into C_CG for a few products. That was a mistake at WST. In Rhapsody in C the property CG::Event::BaseNumberOfInstances is the only one that should be available and that influences code generation.

What the property mainly does, is controlling code generation of event allocation. If the property is set to a positive value, the resulting event code looks similar to this in a package C file:

```
/**## event evX() */  
  
RIC_IMPLEMENT_MEMORY_ALLOCATOR(evX, 1, 0,  
TRUE)  
...  
  
evX * RiC_Create_evX( void ) {  
evX* me = RIC_MEMORY_ALLOCATOR_GET(evX);  
if(me!=NULL)  
{  
evX_Init(me);  
}  
return me;  
}
```

```
void RiC_Destroy_evX(evX* const me) {
if(me!=NULL)
{
evX_Cleanup(me);
}
RIC_MEMORY_ALLOCATOR_RETURN(me, evX);
}
```

The empty code does not use the memory allocation macros, but malloc and free:

```
/*## event evX() */
...

evX * RiC_Create_evX( void ) {
evX* me = (evX*) malloc(sizeof(evX));
if(me!=NULL)
{
evX_Init(me);
}
return me;
}

void RiC_Destroy_evX(evX* const me) {
if(me!=NULL)
{
evX_Cleanup(me);
}
free(me);
}
```

Next, we need to differ, if we are using an OO RTX based framework (also referred as lite) or an RXF depending on a full-blown RTOS (oxf based).

For OO RTX frameworks it's really simple: the property always needs to have the value 1. Loading the product's profile (e.g. Beethoven.sbs) into your model and assigning the Stereotype RXFComponent takes care of setting it to "1".

For RTOS based frameworks there are two ways. Working with dynamically allocated events is possible by setting BaseNumberOfInstances to an empty value. No 0, but just empty. This should also be the default. If you need to work with runtime-static buffers, you must set the property to the maximum number of events that could be in use at any point of time in your model. Do not forget, that events might not be freed before new related events are created. So you should assign a value that is high enough.

The last mystery: In older RXF releases (3.x) we did not use profiles. So the only way to setting BaseNumberOfInstances to 1 for OO RTX (lite) frameworks, was to set it via property file. But as the property is general and not under an environment metaclass, it influences all

environments. This could cause problems with the Microsoft and other parallelly installed environments, where the property manually needed to be overridden.

Weitere Tech Tipps finden Sie auch in folgenden Foren:

Embedded UML Forum

Dieses neue Forum ermöglicht Entwicklern von Embedded Systemen mit begrenzten Ressourcen einen Know-How-Austausch. Hier können allgemeine Embedded UML-relevante Themen und Fragestellungen ebenso platziert und diskutiert werden wie spezielle technische Fragen zu Targets, Tools und Utilities.

www.uml-forum.de

Anwender Forum RHAPSODY4YOU.

Unabhängiges Forum für Rhapsody User. Hier findet ein Austausch über technische Fragen zu *Rhapsody*®, zu [Rhapsody® AddOn's und Third Party Tools](#) sowie zu sonstigen mit *Rhapsody*® verbundenen Themen statt.

www.rhapsody4you.org

Tool News

Gleich 2 neue Versionen von Rhapsody®.

Seit unserer letzten NewsLetter gab es gleich 2 neue Releases von Rhapsody®.

- 7.3 MR1, die „Maintenance“ Release. Diese Release enthält einige bug-fixes. Es gibt keine großen Produktänderungen. Die WST Bridges laufen mit den gleichen Bedingungen, wie unter der 7.3 Release. **Status: Empfohlen**
- 7.4, die "Bluewash" Release.

Durch die Übernahme von Telelogic durch IBM müsste jetzt auch Rhapsody® angepasst werden an das IBM look-and-feel und sämtliche IBM-Eigenschaften. Es sind aber auch noch Neuigkeiten und bug-fixes enthalten.

- Architect enthält ab jetzt auch Reverse Engineering
- Integration von Test RealTime
- Verbesserungen in dem XI import von u.A. Enterprise Architect

Diese Release wird momentan durch WST Entwicklung getestet und ist noch nicht freigegeben fuer den Einsatz von WST Bridges.

Status: noch nicht installieren

Lesen Sie unsere Guides für die Installation einer neuen Version von Rhapsody und das Installieren von mehreren Versionen gleichzeitig auf www.umlforum.de

Mehr Information erhalten Sie in der Defect Fix List und den ReadMe's, die auf www.telelogic.com

verfügbar sind oder nehmen bei Fragen einfach Kontakt mit uns auf !

support@willert.de

Weitere aktuelle Informationen zu neuen Releases finden sie auch in oben genannten Foren.

Redaktionelles

Sicherheitsrelevante Komponenten unter höchsten Qualitätsanforderungen entwickeln.

Dieser Herausforderung begegnet die Schmidhauser AG erfolgreich mit Model Driven Development und Embedded UML Studio™ von Willert Software Tools.

Die Schmidhauser AG ist ein Mitglied der Lenze Gruppe und ein Kompetenzzentrum für kundenspezifische Antriebssteuerungen. Mit gut 55 Angestellten am Sitz in Romanshorn, Schweiz, erbringt das Unternehmen verschiedenste Dienstleistungen in der Produktentwicklung (Hardware, Software und Leistungselektronik). Ende 2006 entschied Schmidhauser, ihre mobilen Antriebssteuerungen für Fahrzeuge neu mit Embedded UML Studio™ von Willert Software Tools zu entwickeln; das Tool basiert auf Telelogic Rhapsody® und ist auf Zielplattformen mit kleinem Speicherplatz ausgerichtet. Ausschlaggebend für den Entscheid waren die hohen Qualitätsanforderungen, welche die sicherheitsrelevanten Komponenten von Schmidhauser erfüllen müssen.

„Ein Bild sagt mehr als tausend Worte. Dank den Modellen von Rhapsody® können wir neue Funktionalitäten viel besser mit unseren Kunden diskutieren – auch ohne Code-Kenntnisse.“ Peter Bode, Projektleiter Mobile Drives, Schmidhauser AG

Für mehr Informationen:

www.evocean.ch/lang-de/ihr-vorteil/beratungsprojekte.html

Preis-Aktion für ARM-Werkzeuge

Sie überlegen, ob Sie dieses Jahr noch in ARM-Werkzeuge investieren sollen?

Hier eine kleine Entscheidungshilfe:

15% Ersparnis für folgende ARM-Tools

- MDK-ARM + RL-ARM zusammen nur 5900 € (statt 7000 € regulärem Preis)
- RL-ARM für 2900 € (statt 3400 € regulärem Preis) in Verbindung mit einem Wartungsvertrag für den Compiler (360 € für ½ Jahr)
- RL-ARM für 2900 € (statt 3400 € regulärem Preis) in Verbindung mit einem gültigen Support-Vertrag (PSN vom MDK-ARM ist erforderlich)

Dieses Angebot gilt bis zum 16. Dez.2008

Kontakt: rschaak@willert.de Herr Schaak

Termine

UML Schnupper-Workshop

Kinderleicht sind die ersten Schritte mit der UML nicht, aber wir machen Ihnen die erste Begegnung so leicht wie möglich. Der ideale Einstieg, wenn Sie sich für UML interessieren und einen ersten Eindruck von den Vorteilen und Einsatzmöglichkeiten in Ihren Projekten gewinnen möchten. Es werden typische Einsteigerfragen beantwortet und bereits die ersten Schritte mit UML und Rhapsody® trainiert. Außerdem erhalten Sie wertvolle Tipps und Ratschläge für eine UML-Tool-Evaluierung.

Nächster Schnupper-Workshop in:

Ulm 02.12.2008
www.willert.de/events

Rotkreuz/Schweiz
www.evocean.ch/lang-de/schulungen.html

Embedded UML Start-Up Training

Intensivtraining für alle, die sich auf eine qualifizierte Evaluierung oder gründlich auf den Einsatz von UML vorbereiten wollen. Nach diesem Training sind Sie fit für die Nutzung der UML und von Rhapsody® in Ihren Projekten.

Nächster Trainingstermin: 26.- 30.01.2009
www.willert.de/events

Das Besondere an unseren Schulungen: Jeder Teilnehmer erhält ein MCB2300 Evaluation Board mit einem Keil ULINK2 USB-JTAG Adapter

und allen notwendigen Tools, um die Übungen auch nach dem Training vertiefen zu können.



Embedded Software Engineering Kongress

08.- 10.Dezember 2008

in Sindelfingen

Der Embedded Software Engineering Kongress 2008 ist die erste deutschsprachige Veranstaltung, die sich ausschließlich und tiefgehend den vielfältigen Themen und Herausforderungen bei der Entwicklung von Geräte- und Systemsoftware für Industrieanwendungen, Kfz-Elektronik, Telecom sowie Consumer- und Medizintechnik widmet.

Das Konferenzprogramm und die Broschüre können Sie einsehen unter :

www.microconsult.de/ese-kongress

Weitere interessante Termine:

- Rhapsody® in C++ Tool Training,
24.- 27.11.2008 in Rotkreuz, CH
- Rhapsody® in C++ Hands-On Seminar,
02.12.2008 in Rotkreuz, CH
- Workshops mit Bruce P. Douglass
im März 2009

Mehr Informationen zu den 3 Terminen:

www.evocean.ch/lang-de/schulungen.html

Neue ARM7 und Cortex-M3 Trainings

Quategra hat das von ARM lizenzierte Training für die ARM7 und CORTEX-M3 basierenden Mikrocontrollern umstrukturiert und bietet ab sofort spezielle Trainings zu aktuellen Controller-Familien, RTOS mit dem RTX-Kernel und der RealView® Real-Time Library an. So kann das erworbene Wissen direkt in neue Projekte einfließen. Dabei wird Wert auf praxisnahe und kompakte Inhalte gelegt, die unmittelbar anwendbar sind. Darüber hinaus bietet Quategra die Entwicklung von kompletten Board-Support-Packages für eigene ARM- basierte Hardware an.

ARM-Trainings-Programm:

- ARM7: NXP LPC2000 Hands-On Workshop:
24.- 25.11.2008
21.- 22.04.2009
- RTOS : Embedded- Echtzeitbetriebssysteme und Keil RTX-Kernel Training
26.- 27.11.2008
23.- 24.04.2009

Weitere Infos unter:

www.quategra.de/download/arm_embedded.pdf

- Cortex-M3 : STM32 Hands-On Workshop
26.- 27.01.2009
11.- 12.05.2009
- Cortex-M3 : Stellaris LM3S8000 Hands-On WS
19.- 20.02.2009
25.- 26.05.2009

Hierzu mehr unter:

www.quategra.de/trainings

In einem 2-tägigen Training von Quategra zur **IEC61508** (International Electrotechnical Commission, Norm 61508) wird am

1. Tag ein umfassender Einstieg in diese Norm und deren 7 Teile gegeben. Der Schwerpunkt liegt dabei auf den Anforderungen an elektronische Steuerungen und richtet sich vorwiegend an Prozessverantwortliche und Entwickler aus diesem Bereich.

Der 2. Tag befasst sich mit den Anforderungen an die Software-Entwicklung aus dem Teil 3 der IEC61508 und der erforderlichen Dokumentation. Ein Einblick in die Verwendung von C in sicherheitskritischen Systemen und mögliche Testmethoden runden diesen Tag ab.

- IEC61508 Training :
Sicherheitsgerichtete Systeme entwickeln,
sichere Software (in C)
26.- 27.03.2009

Mehr dazu finden Sie unter:

www.quategra.de/trainings

Plattformunterstützung durch Environment Bridges

Verfügbarkeit und Voraussetzungen zum Einsatz von Environment Bridges

Voraussetzung, um eine verfügbare Bridge zu erhalten, ist eine gültige Lizenz von Embedded UML RXF™ oder Embedded UML Studio™. Jede Bridge ist dann als shared source verfügbar und kostenlos erhältlich.

Vorteil: Eine Lizenz reicht aus, um alle vorhandenen Bridges zu erhalten (ohne Garantie auf Aktualität)

Aktuell verfügbare Bridges:

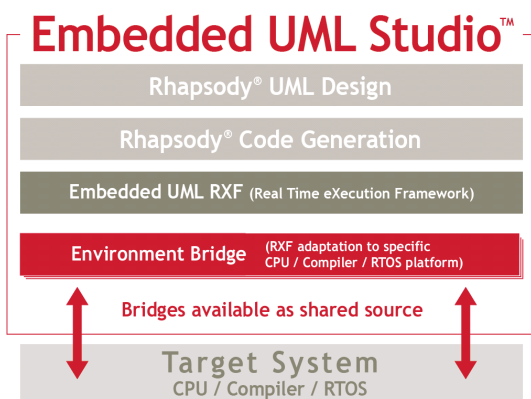
CPU	Compiler	RTOS	Language	Bridge
C16x	Keil C166	CMX-RTX	C	Offenbach
C16x \ XC16x	Keil C166	OO-RTX	C	Gerber
ARM7	Keil MDK ARM	OO-RTX	C	Beethoven
ARM7	Keil MDK ARM	RL-ARM	C	Strauss
ARM7 (GFT)	Keil MDK ARM	OO-RTX	C	Bruckner
ARM	Keil ARM	OO-RTX	C	Liszt
ARM7	IAR EWARM	embOS	C	MonteVerdi
M16C	IAR M16C	embOS	C	Handel
Blackfin	Analog Devices Visual DSP++	OO-RTX	C	Mendelssohn
F2MC16	Fujitsu F2MC16	OO-RTX	C	Brahms
PowerPC	GNU gcc	Linux	C	Sibelius
HCS08	Metrowerks Codewarrior	OO-RTX	C	Levi
M16C	Renesas M16c	OO-RTX	C	Lachner
C16x \ XC16x	Tasking C166 V8.x	OO-RTX	C	Kimberger
TI TMS320	TI CodeComposer	DSP/BIOS	C	Cornelius
C166	Tasking VX C166	OO-RTX	C	Debussy
TI TMS320	TI CodeComposer	OO-RTX	C	Loewe
PIC32	Microchip MPLABC32	OO-RTX	C	Weber
AT90CAN128	AVRStudio gcc	OO-RTX	C	Reinecke

Stand September 2008: Anpassungen sind für das RXF V5 und Rhapsody® V7.3

Willert Software Tools erstellt OSEK - konforme Anpassungen für den Automotive - Bereich. Auf der Basis des Embedded UML RXF™ wird auch der BMW Standardcore von uns unterstützt.

Mit unserem Framework Bruch und Schuhmann unterstützen wir auch die Chipkarten von Infineon: Infineon SLE70 und SLE88.

Wenn es noch keine Bridge für die eigene Zielplattform gibt



Die Firma Willert Software Tools bietet für diesen Fall verschiedene Arten der Unterstützung an. Zusammen mit unseren Partnerfirmen können Zielplattform-Anpassungen (C/C++) als Auftragsentwicklung durchgeführt werden. Alternativ ist es auch möglich das Know How zur Anpassung in einem zweitägigen Inhouse-Workshop zu vermitteln. Sprechen Sie uns an, wir werden sicher einen für Sie attraktiven Weg finden.

Rhapsody® is a registered Trademark of Telelogic (an IBM company)
Embedded UML RXF™ is a trademark of Willert Software Tools
Embedded UML Studio™ is a trademark of Willert Software Tools

Herausgeber:

Willert Software Tools GmbH
Hannoversche Straße 21 - 31675 Bückeburg

www.willert.de - info@willert.de

Tel. 05722 - 9678 60