

Inhalte

- **Gutes Werkzeug ist halbe Arbeit** - Auswahl einer ARM Entwicklungsumgebung orientiert an Projekt-Anforderungen
- **Tech Tips** - Nutzung von eigenen Datentypen in UML-Modellen - Statische Multiplizitäten
- **Nützliche Online Foren**
- **Aktuelle Termine**

Gutes Werkzeug ist halbe Arbeit: Auswahl einer ARM Entwicklungs-Umgebung

Mit entscheidend für erfolgreiches Software Engineering sind die Werkzeuge, die den Software-entwicklern zur Verfügung stehen. Manche Mikrocontroller-Familien lassen an dieser Stelle keine große Wahl zu und Entwickler sind froh, wenn die einzigen verfügbaren Werkzeuge halbwegs stabil funktionieren.

Anders bei Mikrocontroller-Familien (Architekturen), die sich breit am Markt durchgesetzt haben wie z.B. die 8051, C166 oder die ARM-Architekturen. Hier steht eine breite Auswahl an Werkzeugen zur Verfügung.

Grundsätzlich eine hervorragende Situation für effizientes Software- Engineering, aber wird sie auch genutzt? Dass ein C-Compiler benötigt wird steht nicht zur Diskussion. Aber schon bei der Entscheidung ein RTOS einzusetzen scheinen die Vor- und Nachteile nicht so recht klar zu sein. Und was hat es mit OOP auf sich? Sollte bei einem neuen Projekt lieber in C++ programmiert werden? Zu häufig werden derartige Entscheidungen aus Zeitmangel von einem Projekt zum nächsten verschoben und das Software-Engineering bleibt auf der Strecke.

Das es sich lohnt, sowohl aus finanzieller Sicht, als auch aus Sicht der Effizienzsteigerung, sich mit den Projektanforderungen auseinander zu setzen und darauf basierend die Tools auszuwählen, möchten wir exemplarisch in der Zusammenstellung von 4 verschiedenen Entwicklungsumgebungen zeigen.

Immer stehen die Anforderungen der Projekte im Vordergrund. Vor allem auch Anforderungen wie Lebensdauer, Wartbarkeit und Erweiterbarkeit der Produkte, aber auch Sicherheit und Robustheit der zu entwickelnden Software. Und nicht vergessen sollte man die Erfahrungen der Entwickler im Software- Engineering, als auch die



Einarbeitungszeiten in neue Techniken und dem gegenüberstehende ‚Time To Market‘ Situationen.

Die von uns zu Komplettlösungen zusammengestellten Werkzeuge sind optimal aufeinander abgestimmt und haben sich in diesen Kombinationen bereits in zahlreichen Projekten bewährt.

Ein großer Vorteil ist, dass die Entwicklungsumgebungen ohne Kompatibilitätsprobleme, an steigenden Anforderungen orientiert, erweitert werden können. So, dass der Return of Invest in die Tools und in die Einarbeitungszeiten über einen langen Zeitraum gesichert ist, auch wenn sich Engineering-Anforderungen ändern. Ebenso bietet die Erweiterungsmöglichkeit eine ideale Voraussetzung für einen schrittweisen Einstieg in neue Technologien und damit die Verteilung von Investitionen und Einarbeitungszeiten auf einen längeren Zeitraum.

Beispiel:

Hardware Architektur: Aus Kosten und EMV-Gründen ist die Architektur als Single Chip (kein externer Speicher) geplant.

Echtzeitanforderungen: sind hoch, es gibt jedoch nur wenig Nebenläufigkeiten.

Lebensdauer des Produktes: sind 15 Jahre, wobei umfangreiche Änderungen nicht zu erwarten sind. Wartbarkeit muss über den ganzen Lebenszyklus gesichert sein.

Time To Market: es stehen nur wenige Monate zur Realisierung zur Verfügung.

Qualität: Änderungen des Produktes nach der Auslieferung verursachen hohe Kosten im Bereich Service und sind zu vermeiden.

Erfahrung: in der Anwendung der Hochsprache ANSI-C auf verschiedenen Hardwareplattformen ist vorhanden, jedoch keine Erfahrung mit der ARM-Architektur. Erfahrungen im Einsatz eines RTOS oder anderen Werkzeugen zum Architekturdesign sind nicht vorhanden.

Budget: es steht ein kleines Budget für die Anschaffung von Entwicklungstools zur Verfügung.

Auswahl der Entwicklungsumgebung orientiert an den obigen Anforderungen

Da feststeht, dass das Projekt auf Basis eines Single-Chip-Designs niemals mehr als 256 K-ROM benötigt wird die kostengünstige Keil RV ARM Compiler Version mit Speicher-Einschränkungen als Basis der Entwicklungs-Umgebung ausgewählt, um mehr Gestaltungsspielraum mit dem geringen Budget zu bekommen.

Die Analyse des Architektur-Designs ergibt nur wenige Tasks mit unterschiedlichen Nebenläufigkeiten. Trotz der geplanten langen Lebenszeit ist die Wahrscheinlichkeit gering, dass komplexe Erweiterungen zu erwarten sind. Diese Umstände lassen auf eine einfache Laufzeitarchitektur schließen, die sehr effizient auf Basis einer main()-Loop realisiert werden können. Aus diesem Grund und zu Gunsten der Einarbeitungszeit wird auf den Einsatz eines RTOS verzichtet. Echtzeitanteile mit definierten Reaktionszeiten werden auf der Interrupt-Ebene realisiert.

Die Software-Entwicklung mit dieser Umgebung erfolgt in gewohnter Manier: manuelle Erstellung des C-Codes. Trotzdem wird auf handwerklich korrekten ANSI C-Code Wert gelegt. Die Anforderungen an die Code-Qualität werden durch die Integration eines statischen Code- Analysers (PC-Lint) in die Entwicklungsumgebung sichergestellt.

Die geforderte Robustheit des Produktes zur Auslieferung und wenig Weiterentwicklung haben zu der Entscheidung geführt die Tests händisch durchzuführen. Dieses geschieht auf Basis der Keil μ Vision Debugging Umgebung und dem U-Link Debugging Interface.

Die Anforderung an lange Lebensdauer und evtl. notwendige kleinere Anpassungen erfordert eine gute Dokumentation, um sich auch nach Jahren wieder effizient und sicher in die Software einarbeiten zu können. Um dieser Anforderung gerecht zu werden, jedoch gleichzeitig den Aufwand für die Dokumentation so gering wie möglich zu halten und gleichzeitig Inkonsistenz zwischen Code und Dokumentation vorzubeugen,

haben wir uns für den Einsatz des Dokumentation-Tools Doxygen entschieden.

Auf Grund des hohen Zeitdruckes und dem Umstand, dass noch keine Erfahrungen im Einsatz der ARM-Architektur vorhanden sind wird eine 3-tägige ARM-Schulung eingeplant, um lange Einarbeitungszeiten in die HW-Architektur zu vermeiden.

In der Summe zeichnet sich die ausgewählte Umgebung durch geringen Einarbeitungsaufwand aus. Es werden abgesehen von der ARM-HW keine unbekanntes Technologien eingesetzt und daher wenig unerwünschte Verzögerungen in der Realisierung erwartet.

Trotz des geringen Budgets für die Entwicklungsumgebung wurde auf Basis der preiswerten limitierten Version des Compilers (die in diesem Fall keine Einschränkungen beinhaltet) ein ARM- Seminar und die Anschaffung von PC-Lint ermöglicht.

Bestandteile dieser Umgebung

- Keil RV ARM Compiler Limited Version (*mit Speicher-Einschränkungen*)
- PC-Lint
- Doxygen ⁽¹⁾
- U-Link
- Eva Board
- ARM - Architektur Schulung (3 Tage)

Gesamtkosten ca. 4.000 EUR

⁽¹⁾ Doxygen ist nicht im Lieferumfang enthalten. Nutzung als GNU General Public License

Ergänzend zu obiger Umgebung raten wir zum Einsatz eines Configuration Management Tools.

3 weitere Beispiele finden Sie auf unserer Homepage in dem Artikel „Gutes Werkzeug ist halbe Arbeit“

<http://www.willert.de/assets/Datenblaetter/DatBl-Nachfrage-Sog-ARM-Preisinfo-4-Lsungen-V5.0d.pdf>

Wenn Sie ein neues ARM / Cortex Projekt planen beachten Sie auch folgende Veranstaltungen.

- DESIGN&ELEKTRONIK-Workshop „Erfolgreich entwickeln mit ARM“
- Training: „ARM Embedded“ (ARM7 und ARM-CORTEX-M3)
- Training: „KEIL RTX Real-Time Kernel und RL-ARM Realview Real-Time Library“

Weitere Informationen:

<http://www.quategra.de/termine.php>

Tech Tips:

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 Tips gehen wir auf häufig gestellte Fragen an unser Support Team ein und geben Antworten.

Tech Tip: Nutzung von eigenen Datentypen in UML Modellen

Viele *Rhapsody®* Anwender stoßen nach kurzer oder längerer Zeit auf die Frage wie sie eigene Datentypen in einem UML Modell verwenden können. Im Folgenden möchten wir einen Weg aufzeigen wie das möglich ist.

In einem neu angelegten Modell stellt *Rhapsody®* zuerst einmal Standard Datentypen zur Verfügung die der jeweiligen Sprache entsprechen für die Code generiert werden soll. Demzufolge bietet *Rhapsody® in C* folgende Datentypen als default an:

char, short, int, long, double, long double, float, unsigned char, unsigned short, unsigned int, unsigned long, void, void *, char * und die *Rhapsody®* Typen **RiCBoolean** und **RiCString**.

Diese Datentypen reichen für viele Zwecke aus, haben aber auch Nachteile. Wenn man zum Beispiel sein Modell und den daraus generierten Code weitgehend unabhängig von Compiler und CPU machen möchte. In diesem Fall sind z.B. Datentypen sinnvoll, die auf allen Plattformen konsistente Größe und Verhalten haben. Ein anderer Grund kann MISRA konformer Code sein. In verschiedenen Fällen kommt man also um eigene Datentypen nicht herum.

In der Notation ANSI-C wird das durch das anlegen von typedefs erreicht. Zum Beispiel „U8“, „S8“, „U16“, „S16“ usw. wobei U für unsigned, S für signed und die Zahl für die Anzahl der Bits steht.

Auch in *Rhapsody®* ist es sehr komfortabel möglich mit eigenen Datentypen zu arbeiten. Hierfür bietet es sich an die neuen Datentypen alle zusammen in einem gemeinsamen Package zu definieren. Für jede Plattform würde man ein solches Package definieren mit den für diese Plattform entsprechend abgeleiteten Datentypen.

Types werden als UML Type wie folgt definiert: In einem Package „Add New“ und dann „Type“ anwählen. Dann unter „General“ als „Kind“ „Typedef“ auswählen und im Tab „Details“ den richtigen Basistype auswählen. Also für U8 ein „unsigned char“ und für S16 ein „short“.

Wenn das gemacht wurde können diese Datentypen für Attribute, Variablen, Argumente und Returnvalues von Operationen ausgewählt werden. Die ursprünglichen Standardtypen sind weiterhin vorhanden. *Rhapsody®* bietet 2 Properties mit denen auch das geändert werden kann:

```
General:Model:CommonTypes "TypesPkg"
```

```
General:Model:DefaultType "TypesPkg::U16"
```

Alle in dem ersten Property angegebenen Packages bzw. den darin enthaltenen Type Definitionen werden in den *Rhapsody®* Type Dialogen

aufgelistet. Wenn die Property leer ist werden die PredefinedTypes(C) aufgelistet.

Der erste Eintrag eines eigenen Datentype bewirkt, dass die Standard Datentypen NICHT mehr gezeigt werden. Sollen diese weiterhin angezeigt werden, dann müssen die „PredefinedTypes(C) Packages auch mit aufgenommen werden.

\$ALL bedeutet, dass alle Typen im Modell gelistet werden

Die angegebenen Packages müssen mit dem vollständigen Pfad in Form einer Liste durch „;“ getrennt aufgenommen werden.

Die zweite Property enthält den Datentype, den *Rhapsody®* als Default anbietet. Achtung!! Vollständiger Pfad, also Package::Type angeben.

Jetzt gibt es aber noch die Animation zu beachten. Die *Rhapsody®* Animation kann die neuen Datentypen nicht darstellen. Um auch die korrekte Darstellung zu ermöglichen müssen Funktionen zur Verfügung gestellt werden, die eine Konvertierung von Datentype zu String und zurück vornehmen. Die Namen dieser Funktionen sind abzuleiten vom Datentype. Z.B: serialize<DatenType> und unserialize<Datentype>. Für ein U8 müssen also folgende Funktionen implementiert werden:

```
char * serializeU8( U8 n);
```

Diese Funktion reserviert Speicher mit malloc() für ein String und druckt den Inhalt der Variablen dort hinein. *Rhapsody®* wird den Speicher wieder freigeben.

```
U8 unserializeU8( char * s, U8 n);
```

Bekommt einen String, extrahiert den Inhalt und schreibt ihn zurück in die Speicheradresse der Variablen.

Diese Funktionen werden natürlich nur benötigt wenn Animation genutzt wird und Animation eingeschaltet ist. Aus diesem Grund benötigen wir noch einen Stereotype <AnimationOnly> dass folgende Properties setzt:

```
C_CG:Operation:ImplementationEpilog  
"ifdef _OMINSTRUMENT"
```

```
C_CG:Operation:ImplementationProlog  
"#endif // _OMINSTRUMENT"
```

_OMINSTRUMENT wird durch *Rhapsody®* gesetzt wenn Animation aktiviert ist. Dieser Mechanismus bewirkt nun, dass die Funktionen nur dann mit kompiliert werden wenn Animation aktiv ist.

Ein Modell mit einem kleinen Beispiel finden Sie auf unserer Homepage:

<http://www.willert.de/models-source-code/flat/116>

Auf diese Weise können mit *Rhapsody®* auf Unternehmensebene, Projektebene oder Mitarbeiter-Ebene individuelle oder einheitliche Datentypen genutzt werden.

Viel Spaß mit *Rhapsody®*.

Weitere Tech Tips 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

Tool News

Rhapsody Version 7.3 ist verfügbar.

Informationen finden Anwender im Client Center, <https://support.telelogic.com/home.cfm?retry=true>

Alle Embedded UML RXF ab der Version 5.0 und höher sind freigegeben für dieses Release. Wenn Sie ein älteres Produkt einsetzen, wenden Sie sich an unseren Support: mailto: support@willert.de

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

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 Tips und Ratschläge für eine UML-Tool-Evaluierung.

Nächster Schnupper-Workshop: 01.10.2008
<http://www.willert.de/events-2/>

Embedded UML StartUp 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: 22.-27.09.2008
<http://www.willert.de/events-2/>

„TestConductor Training“

Mit dem *Rhapsody*® *TestConductor*™ der Firma OSC-Embedded Systems AG können auf Basis von UML-Modellen Tests automatisiert werden. Wie mit allen Notationen ergeben sich beim Design Spielräume. Das hat zur Folge, dass der Einsatz des

TestConductor™ an fertigen Modellen oftmals zu nachträglichem Aufwand führt wenn bestimmte Modellierungsrichtlinien verletzt werden. ‚Design for Testability‘ ist das Schlagwort für dieses Problem und wird mit diesem Seminar adressiert.

Auch wenn Sie sich einfach informieren möchten, wie auf Basis der UML sehr effizient Tests automatisiert werden können ist dieses Seminar interessant.

Termin: 25.+26. Nov. 2008

Weitere Infos zum *TestConductor*™:

<http://www.osc-es.de/index.php?idcat=22>

Praxis-Workshop: ARM erfolgreich einsetzen

eine Veranstaltung der Design&Elektronik

Termin: 15.10.2008

<http://www.elektroniknet.de/home/termine/foren/entwicklerforum-arm-erfolgreich-einsetzen/>

Training „ARM Embedded“

Die ARM-Architektur avanciert zum Industriestandard. Wenn auch Sie diese moderne Technologie nutzen möchten, können Sie sich mit uns optimal auf Ihren Einsatz vorbereiten.

Nächster Trainingstermin: 22.-24.09.2008

<http://www.quategra.de/termine.php>

Training: „KEIL RTX Real-Time Kernel und RL-ARM Realview Real-Time Library“

Die RL-ARM Realview® Real-Time Library stellt, neben dem einen Echtzeitbetriebssystem, Lösungen für komplexe Anwendungen in echtzeitfähigen Embedded Systemen bereit. Ihr Einsatz minimiert den Entwicklungsaufwand.

Nächster Trainingstermin: 25.-26.09.2008

<http://www.quategra.de/termine.php>

Das Besondere an unseren Schulungen: Jeder Teilnehmer erhält ein MCB2300 Evaluation Board mit einem Keil ULINK2 USB-JTAG Adapter und allen notwendigen Tool, um die Übungen auch nach dem Training vertiefen zu können.

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