Software-Engineering mit UML und C

Praxisgerechter Einsatz für kleine Embedded-Systeme

Thomas Batt, MicroConsult GmbH, t.batt@microconsult.de

1. Vorwort

In vielen Köpfen ist die UML*) (Unified Modeling Language) [2] verknüpft mit objektorientierter Programmierung und Hochsprachen wie C++, Java oder C#. Das ist natürlich so auch richtig, aber die UML ist ebenso bestens für die Modellierung und Implementierung von kleinen Embedded-Applikationen mit prozeduraler Programmierung in ANSI-C einsetzbar. Wie das im Einzelnen für typische Software-Elemente aussieht und wie die Vorgehensweise dazu ist, zeigt der folgende Beitrag.

2. Software-Schichten

Einer der ersten Schritte zur Definition der Software-Architektur ist die Identifikation der Software-Schichten. Je nach Komplexität der Embedded-Applikation sind es mindestens zwei Schichten. Der Anzahl nach oben hin ist offen. Eine typische 2-Schichten-Architektur besteht aus Applikation und Hardware-Treiber. Mit steigender Komplexität ergibt sich beispielsweise eine 7-Schichten-Architektur bestehend aus Mensch-Maschine-Interface, Applikation, Middleware, Betriebssystem-Abstraktion, Betriebssystem, Hardware-Abstraktion und Hardware-Treiber.

Software-Schichten sind mit der UML-Notation als Pakete, mit einem aussagekräftigen Namen und dem Stereotype <<Software_Layer>> modellierbar. Die Abhängigkeit zwischen den Software-Schichten ist durch die Dependency Relation darstellbar. Diese Dependency Relation repräsentiert auf der späteren C-Modulebene mindestens ein include aus Schichtname_2 in Schichtname_1.

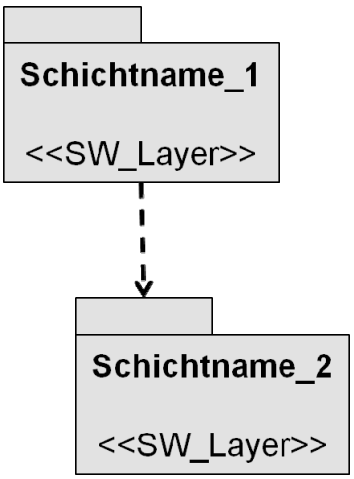

Bild 1: UML-Darstellung von Software-Schichten im Paket- oder Klassendiagramm

Software-Schichten könnten sich als Prefix oder Postfix im C-Modulnamen abbilden. Klassischerweise repräsentiert sich eine Software-Schicht lediglich in der Programmcode-Organisation durch ein Verzeichnis, in dem die entsprechenden Inhalte der Software-Schicht gespeichert sind.

3. Software-Subsysteme

Lässt sich eine Software-Schicht noch weiter strukturieren, ohne die C-Modulebene zu erreichen, kann dies durch Software-Subsysteme erfolgen. Erst die Software-Subsysteme enthalten dann die C-Module. In vielen sehr einfachen Software-Architekturen existiert diese Strukturebene nicht.
In Software-Architekturen, in denen Software-Subsysteme existieren, müssen diese zusammen mit ihren Abhängigkeiten identifiziert und den Software-Schichten zugeordnet werden.
Mischformen sind natürlich auch denkbar, d.h. es existieren Software-Schichten mit und ohne Software-Subsysteme.

Software-Subsysteme sind mit der UML-Notation als Pakete mit einem aussagekräftigen Namen und dem Stereotype <<Software_Subsystem>> modellierbar. Die Abhängigkeit zwischen den Software-Subsystemen ist durch die Dependency Relation darstellbar. Diese Dependency Relation repräsentiert auf der späteren C-Modulebene mindestens ein include aus Subsystem_2 in Subsystem_1.

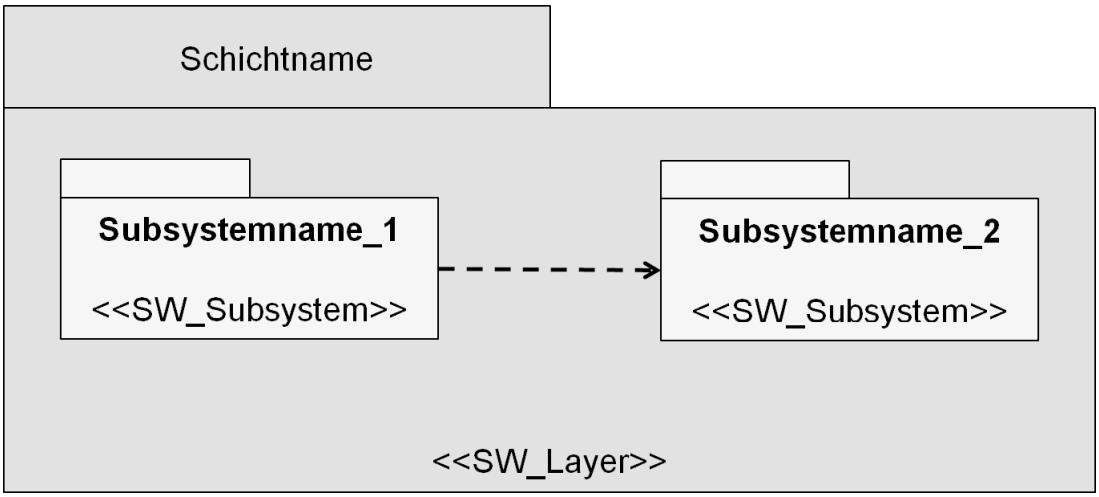

Bild 2: UML-Darstellung von Software-Subsystemen

Software-Subsysteme könnten sich als Prefix oder Postfix im C-Modulnamen abbilden. Klassischerweise repräsentiert sich ein Software-Subsystem lediglich in der Programmcode-Organisation durch ein Verzeichnis, in dem die entsprechenden Inhalte des Software-Subsystems gespeichert sind.

4. Software-Schnittstellen

Was eine Software-Schicht bzw. ein Software-Subsystem einer anderen Software-Schicht bzw. einem anderen Software-Subsystem an Funktionalitäten anbietet, sind Software-Schnittstellen.

Die UML definiert dafür eine spezielle Klasse, die sogenannte Interface-Klasse. Sie unterscheidet sich von der vollwertigen Klassennotation durch den zusätzlichen Stereotype <<Interface>> und das fehlende Feld für die Attribute / Daten.
Eine Software-Schnittstelle repräsentiert sich in C durch eine C-Header-Datei. Da diese neben den Deklarationen der Operationen / Funktionen auch Deklarationen von Daten enthalten kann, ist die C-Header-Datei besser durch eine vollwertige Klassenotation mit Attributen und Operationen modellierbar. Die Klasse sollte durch den Stereotype <<Interface>> und den Stereotype <<C-Header>> gekennzeichnet werden. Zusätzlich zum aussagekräftigen Namen enthalten die Datendeklarationen optional den Datentype. Zusätzlich zum aussagekräftigen Operationsnamen enthalten die Operationsdeklarationen optional den Return-Datentype und die Parameter bestehend aus Namen und Datentype.

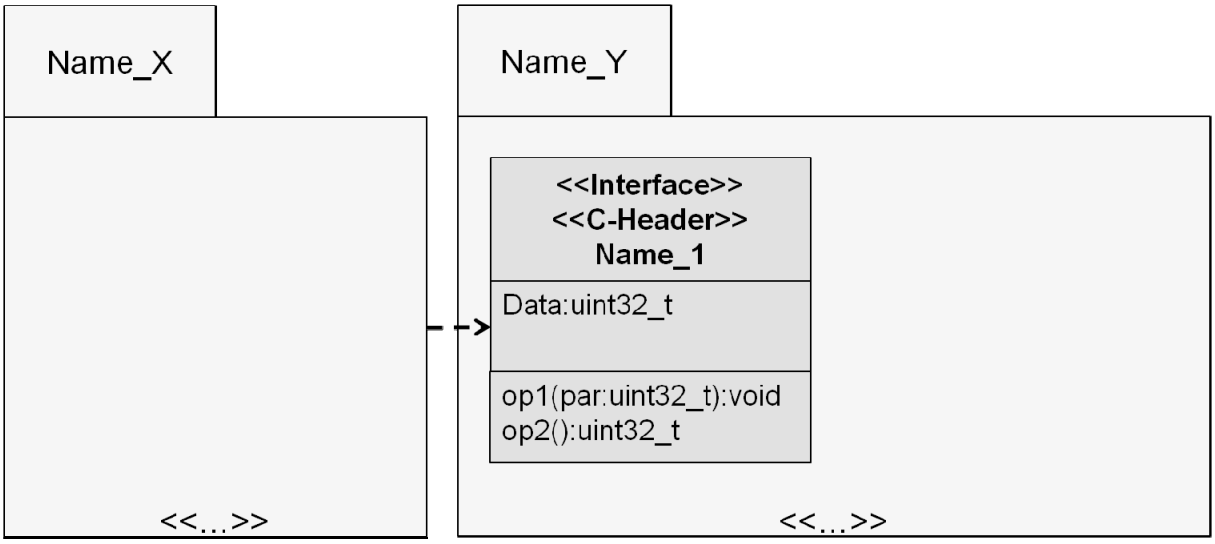

Bild 3: UML-Darstellung von Software-Schichten

Die Dependency Relation, ausgehend von der Kante eins Paketes (Software-Schicht oder Software-Subsystem), endet nun an der Kante der Schnittstellenklasse und nicht mehr an der Kante des anderen Paketes.

5. C-Module und deren Abhängigkeiten

Ausgehend von der bis hierher entstandenen Software-Architektur müssen nun konkrete C-Module identifiziert werden. Dabei ist zu beachten, dass jede Software-

Schnittstellen eine Implementierung in einem entsprechenden C-Modul enthält und jede Software-Schnittstelle von anderen Softwareelementen mindestens einmal genutzt wird.

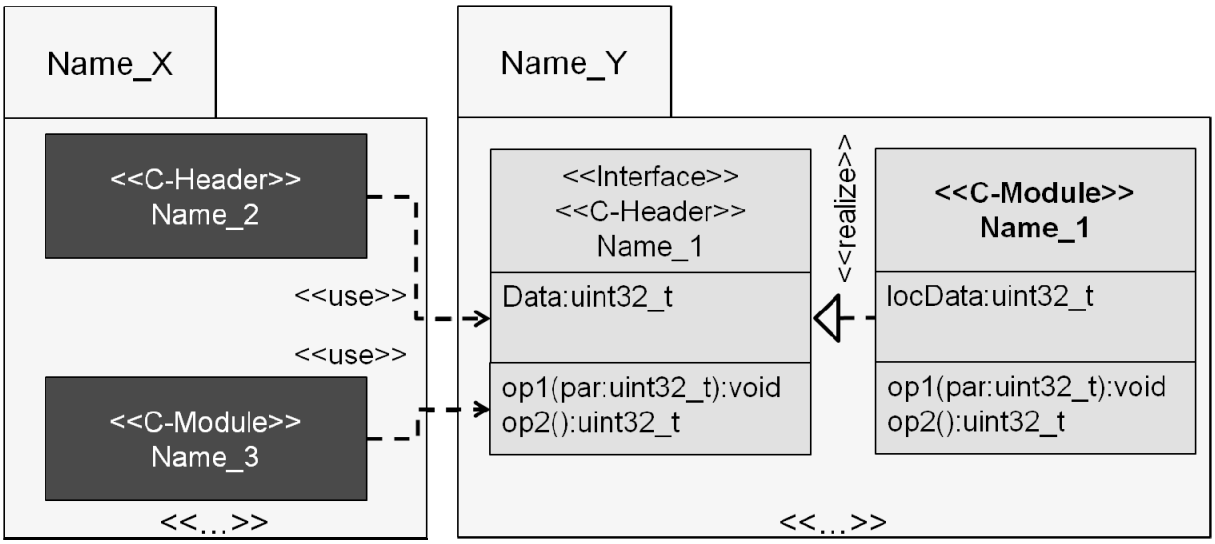

Bild 4: UML-Darstellung von Software-Schichten

Das C-Modul ist mit der UML über eine vollwertige Klasse notierbar. Diese Klasse bekommt den Stereotype <<C-Module>> zugeordnet.
Der Implementierungspfad einer Software-Schnittstelle in einem C-Modul wird durch die Interface- bzw. Realisierungsrelation dargestellt. Der Nutzungspfad einer Software-Schnittstelle oder einer beliebigen anderen C-Header-Datei wird mit der Dependency Relation und dem zusätzlichen Stereotype <<use>> dargestellt. Beide Relationen repräsentieren im C-Code das #include.

6. Software-Abläufe

Die vorangegangene Modellierung der Software-Struktur muss nun noch um die Software-Abläufe ergänzt werden. Diese Abläufe sind klassischerweise in C-Funktionen gekapselt.

Zur Modellierung generischer Software-Abläufe bietet die UML das Aktivitätsdiagramm (ähnlich dem Flussdiagramm) und das Zustandsfolgediagramm an.

Für Software-Abläufe, die wenig ereignisgesteuert und mehr datenflussorientiert sind, ist das Aktivitätsdiagramm zu bevorzugen.

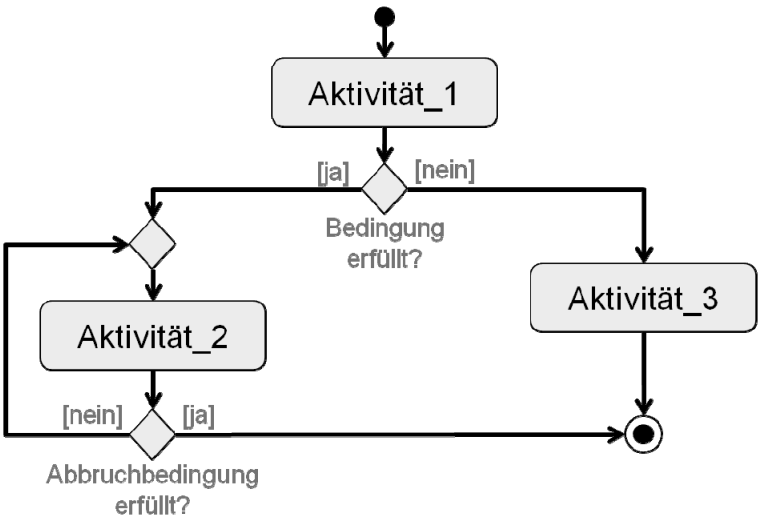

Bild 5: UML-Darstellung von Software-Abläufen

Für Software-Abläufe, die mehr ereignisgetriebenen sind, ist das Zustandsfolgediagramm zu bevorzugen. Dies wird hier aber weiter nicht besprochen.

Beide Diagrammarten enthalten viele Detailnotationen, um auch komplexere Sachverhalte modellieren zu können.

Um Abläufe zu programmieren, bieten sich C-Kontrollkonstrukte wie switch-case und/oder if-else an. Optional lassen sich auch Funktionszeiger geschickt nutzen.

7. main() Funktion

Die zentrale C-Funktion main() kann wie eine spezielle Funktion in einem speziellen C-Modul betrachtet werden.

Mit der UML modelliert ist main() eine Funktion in der Klasse, die das C-Modul der Hauptapplikation repräsentiert. In Ergänzung zum Stereotype <<C-Module>> kennzeichnet sich diese Klasse durch den zweiten <<Main-Application>>.

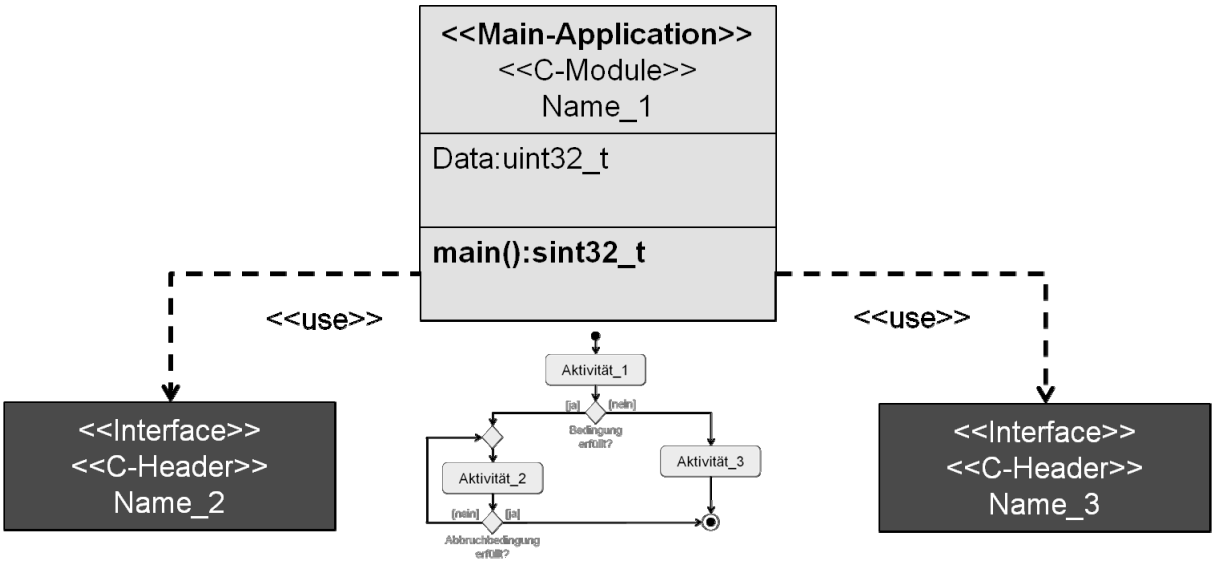

Bild 6: UML-Darstellung der main() Funktion

Alle inkludierten Elemente werden mit der Dependency Relation zusammen mit dem Stereotype <<use>> abgebildet.

Enthält die main() Funktion einen Ablauf, ist dieser durch ein Aktivitätsdiagramm oder einen Zustandsfolgediagramm modellierbar.

8. Zusammenfassung

Software-Element	UML-Notation	Anmerkungen zur C-Implementierung
Software-Schichten	▪ Pakete im Paket- oder Klassendiagramm ▪ Stereotype <<Sofware_Layer>> ▪ Dependency Relation zwischen Paketen	▪ Keine direkte Abbildung im C-Code ▪ Verzeichnis für C-Code-Organisation
Software-Subsysteme	▪ Pakete im Paket- oder Klassendiagramm ▪ Stereotype <<SW_Subsystem>> ▪ Depencency Relation zwischen Paketen	▪ Keine direkte Abbildung im C-Code ▪ Verzeichnis für C-Code-Organisation
Software-Schnittstellen	▪ Klasse im Klassendiagramm ▪ Stereotype <<Interface>> und <<C-Header>> ▪ Dependency Relation zwischen Paket und Klasse	▪ C-Header-Datei mit Deklarationen von Funktionen und optional von Daten ▪ Um eine dynamische Bindung zu bekommen, gibt es komplexere Implementierungen
C-Module	▪ Klassen im Klassendiagramm ▪ Stereotype <<C-Module>>	Keine
C-Modul-Abhängigkeiten	▪ Einfaches include: Dependency Relation zwischen Klassen mit Stereotype <<use>> ▪ Interface-Implementierung: Realize Relation zwischen Klassen	▪ Beide Arten der Abhängigkeitsdarstellung führen im C-Code zum #include ▪ Für die Nachbildung dynamischer Bindung gibt es Varianten der Implementierung
Software-Abläufe	▪ Aktivitätsdiagramm im Kontext einer Funktion (oder Klasse) ▪ Zustandsfolgediagramm im Kontext einer Funktion (oder Klasse)	▪ Einsatz typischer C-Kontrollkonstrukte wie switch-case bzw. if-else ▪ Aufwändiger mit Funktionszeiger
main()	▪ Operation in einer Klasse	Keine

Funktion	„Applikation“ ▪ Stereotype <<Application>> und <<C-Module>>	

Tabelle 1: Software-Elemente und deren UML-Darstellung

9. Schlusswort

Mit den obigen Ausführungen haben Sie nun eine erste Vorstellung über den Einsatz der UML-Diagramme für eine kleine Embedded- und Echtzeitapplikationen in Verbindung mit der Programmiersprache C bekommen. Sicherlich gibt es hierzu Umsetzungsvarianten. Die UML als Standard lässt hier viele Freiräume. Egal für welchen Weg Sie sich letztendlich entscheiden – es ist wichtig, dass dieser Weg gemeinsam für ein Projekt, oder noch besser, gemeinsam im Unternehmen gegangen wird. Sie arbeiten heute nach C-Codierrichtlinien, so sollte es zukünftig auch UML-Modellierungsrichtlinien geben. Da die Modellierung mit „Papier und Bleistift“ sehr mühsam ist, sollten Sie sich dafür ein Tool anschaffen.

Für den hier theoretisch dargestellten Sachverhalt steht unter dem Link [1] ein Download bereit, der eine kleine Applikation für den ASURO Roboter enthält. Diese ist dort als UML-Modell und als ANSI-C Programm enthalten. Da diese Applikation sehr kein ist, enthält sie keine Software-Subsysteme. Sonst sind alle besprochenen Software-Elemente enthalten.

Ich wünsche Ihnen nun viel Erfolg und auch viel Spaß bei der Nutzung der UML in Ihren Projekten.

Thomas Batt

Links:

[1] http://download.microconsult.net/ese2012/uml-c.zip

[2] www.omg.org

Autor:

Dipl.-Ing. (FH) Thomas Batt ist gebürtiger Freiburger. Nach seiner Ausbildung als Radio- und Fernsehtechniker studierte er Nachrichtentechnik an der Fachhochschule in Offenburg. Thomas Batt war in der Entwicklung in den Bereichen Leistungselektronik, Medizinelektronik und CompactPCI/VME-Bus CPU- und Peripherie-Boards tätig.

Heute verantwortet er bei MicroConsult GmbH als Trainer und Coach die Themenbereiche Software Engineering für Embedded-/Realtime-Systeme sowie Entwicklungsprozess-Beratung.

*) UML ist eine eingetragene und geschützte Marke der Object Management Group (OMG) [2]