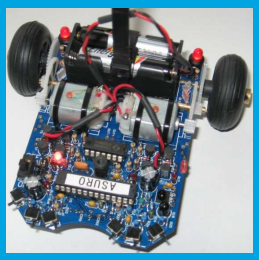
Software Engineering mit UML und C

Inhalt

1 Titel

Vortragsreihe Modellierung I – Embedded Software Engineering Kongress 2012

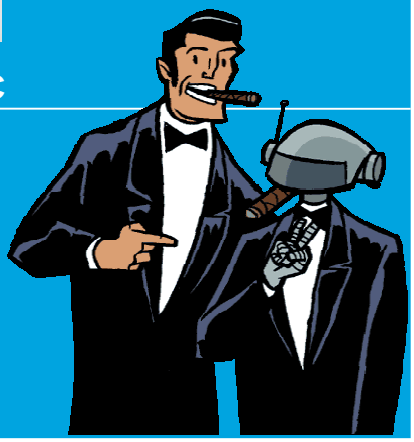
Embedded Software Engineering Kongress



Software-Engineering mit UML und C

Praxisgerechter Einsatz
für kleine Embedded-Systeme

MicroConsult GmbH
Dipl.-Ing. (FH) **Thomas Batt**
Trainer & Coach für Embedded- und Echtzeitsysteme
Charles-de-Gaulle-Str. 6
81737 München • Germany
Tel.: +49 (0)89 450617-35
FAX: +49 (0)89 450617-17
E-Mail: t.batt@microconsult.com
Internet: www.microconsult.de



MicroConsult

2 Inhalt

Inhalt

- UML*) Darstellung (und C-Implementierung) von **Software-Schichten**
- UML*) Darstellung (und C-Implementierung) von **Software-Subsystemen**
- UML*) Darstellung (und C-Implementierung) von **Software-Schnittstellen**
- UML*) Darstellung von **C-Modulen** und deren **Abhängigkeiten**
- UML*) Darstellung (und C-Implementierung) von **Software-Abläufen**
- UML*) Darstellung de C **`main()` Funktion**

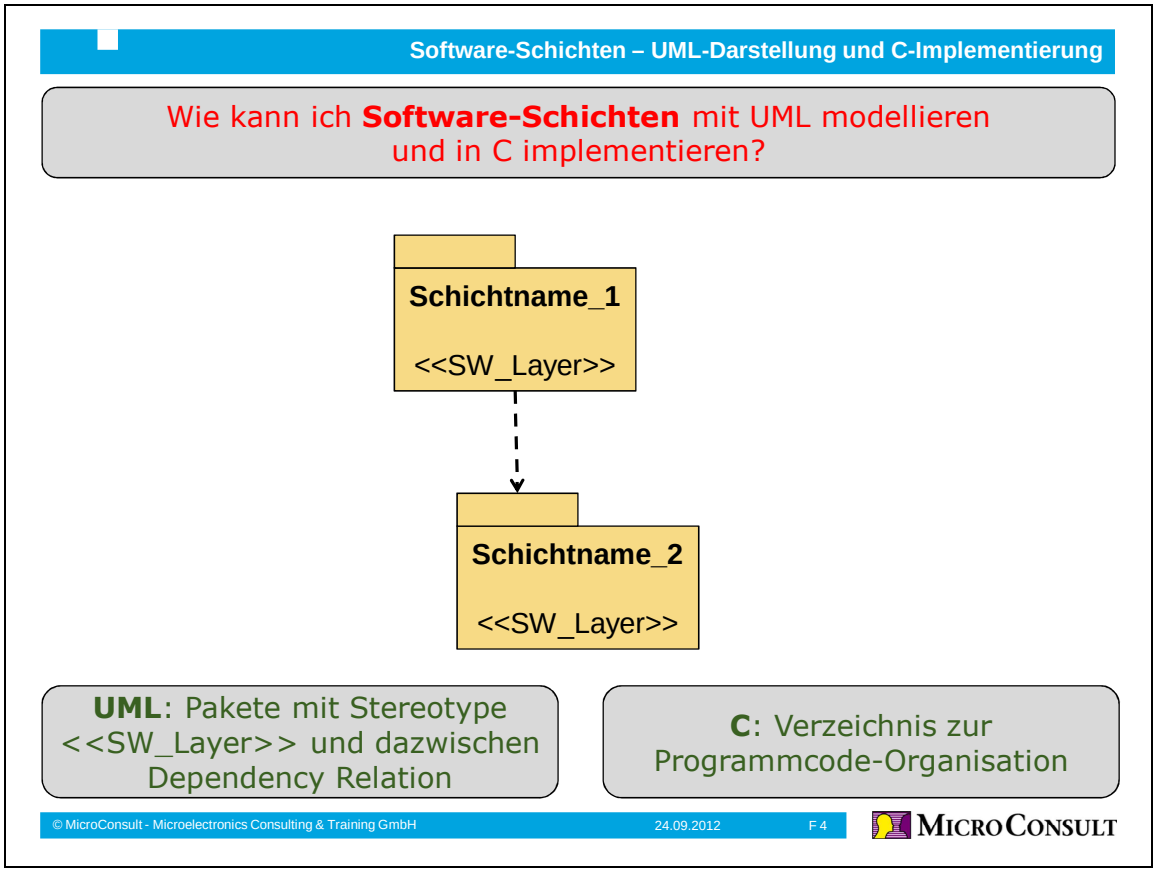
Download-Link für diese Präsentation und das ASURO Roboter-Beispiel:
http://download.microconsult.net/ese2012/uml-c.zip

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

 24.09.2012 F 3 MICRO CONSULT

3 Software-Schichten

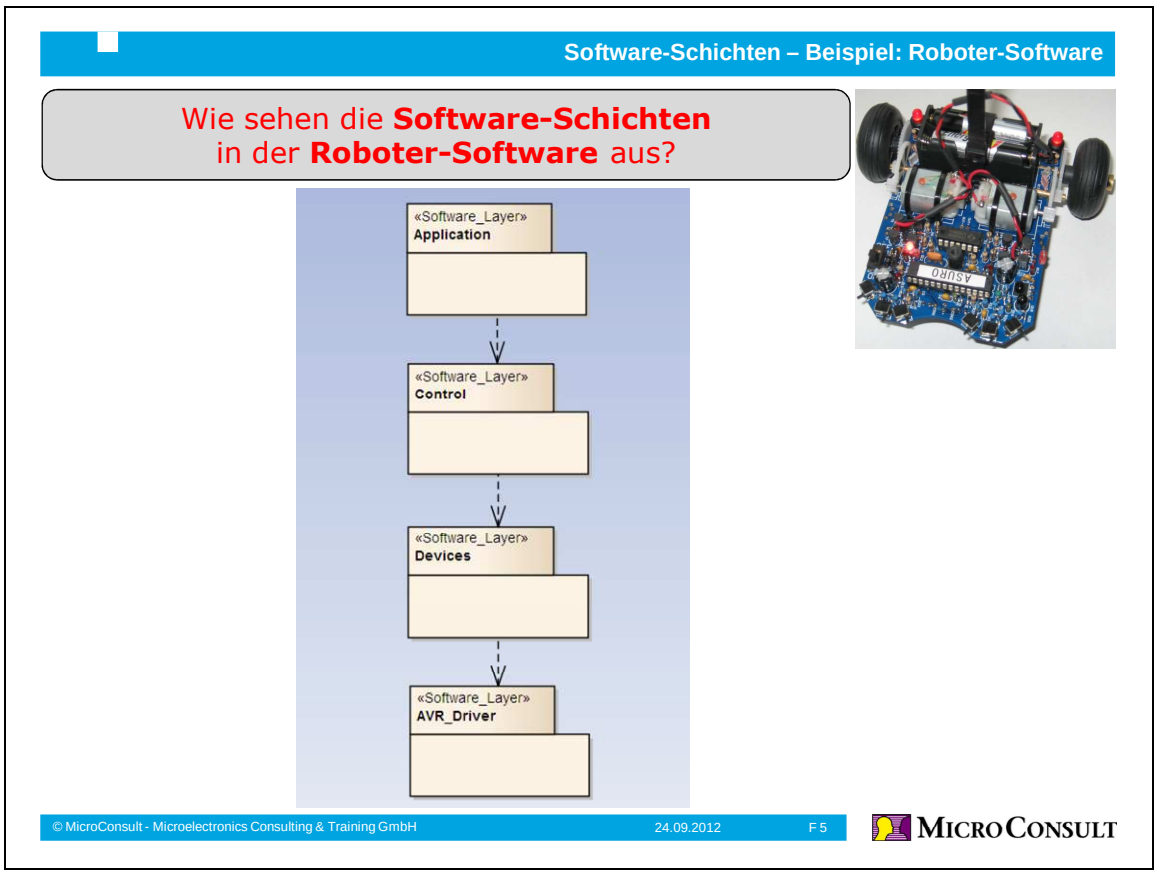
3.1 UML-Darstellung und C-Implementierung



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.

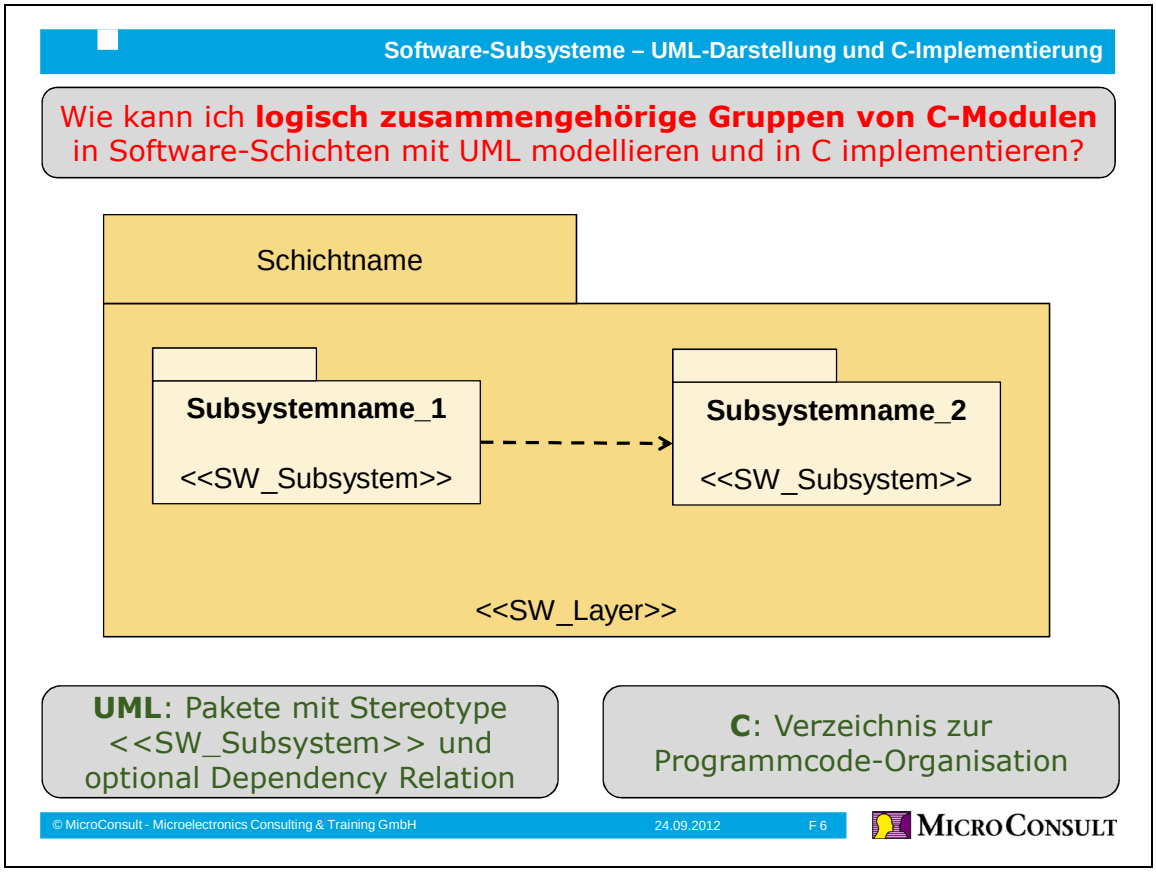
Software-Schichten sind mit der UML-Notation als Pakete, mit einem aussagekräftigen Namen und dem Stereotype <<Software_Layer>> modellierbar. Die Abhängigkeit zwischen den Software-Schichten ist durch die Dependency Relation darstellbar. Diese Dependency Relation repräsentiert auf der späteren C-Modulebene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

3.2 Beispiel: Roboter-Software



4 Software-Subsysteme

4.1 UML-Darstellung und C-Implementierung



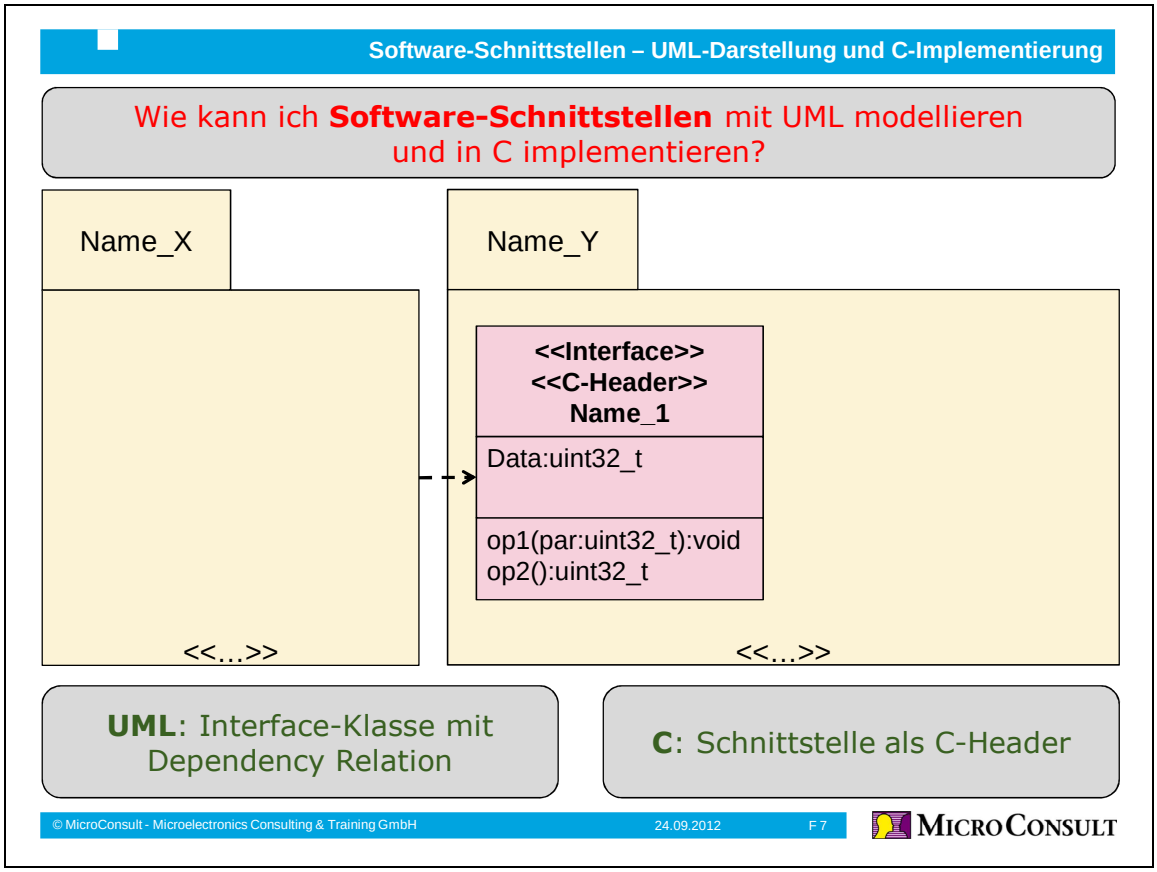
Lässt sich eine Software-Schicht noch weiter strukturieren, ohne die C-Modulebene zu erreichen, kann dies durch Software-Subsysteme erfolgen. Erst die Software-Subsysteme enthalten dann die C-Module. In vielen sehr einfachen Software-Architekturen existiert diese Strukturebene nicht.
In Software-Architekturen, 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.

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.

5 Software-Schnittstellen

5.1 UML-Darstellung und C-Implementierung



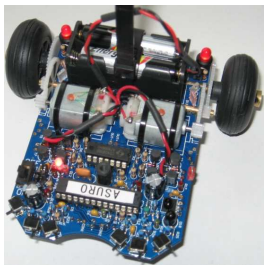
Was eine Software-Schicht bzw. ein Software-Subsystem einer anderen Software-Schicht bzw. einem anderen Software-Subsystem an Funktionalitäten anbietet, sind Software-Schnittstellen. Die UML definiert dafür eine spezielle Klasse, die sogenannte Interface-Klasse. Sie unterscheidet sich von der vollwertigen Klassennotation durch den zusätzlichen Stereotype <<Interface>> und das fehlende Feld für die Attribute / Daten.

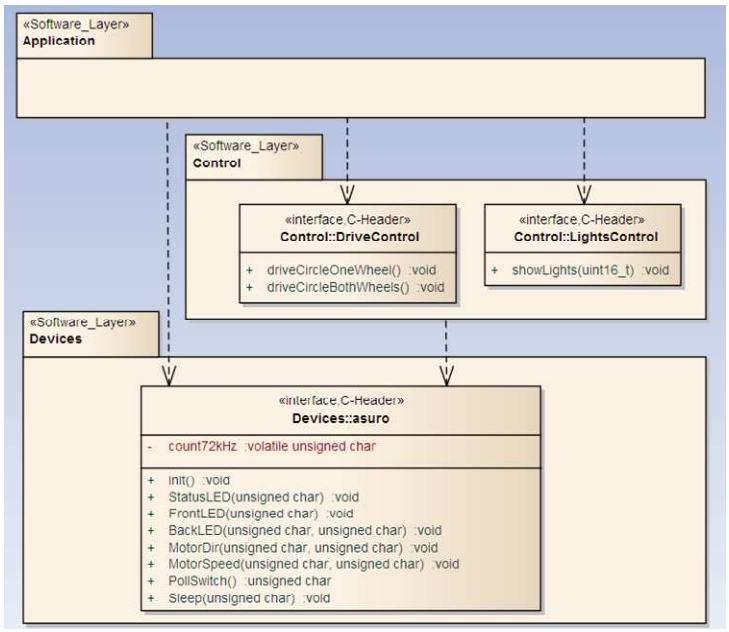
Eine Software-Schnittstelle repräsentiert sich in C durch eine C-Header-Datei. Da diese neben den Deklarationen der Operationen / Funktionen auch Deklarationen von Daten enthalten kann, ist die C-Header-Datei besser durch eine vollwertige Klassenotation mit Attributen und Operationen modellierbar. Die 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.
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.

5.2 Beispiel: Roboter-Software

Software-Schnittstellen – Beispiel: Roboter-Software

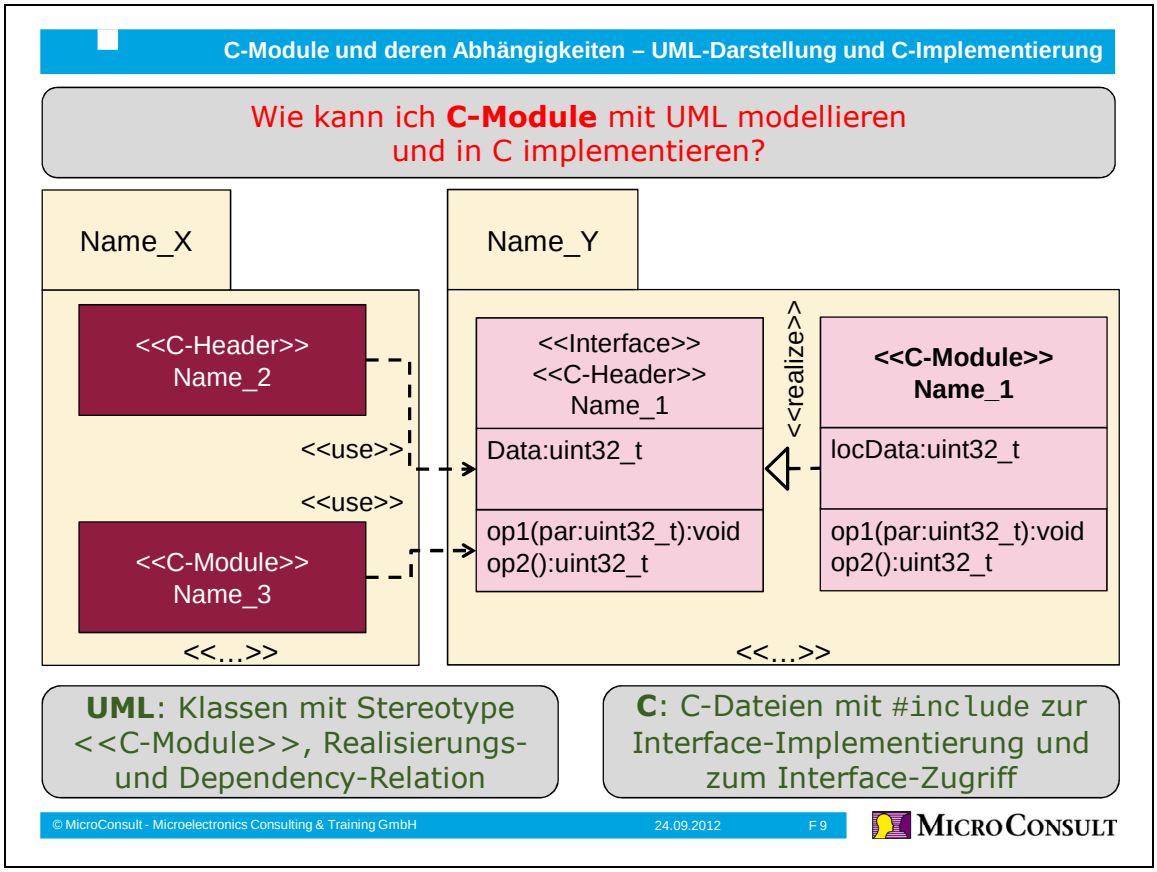
Wie sehen die **Software-Schnittstellen** in der **Roboter-Software** aus?





6 C-Module und deren Anhängigkeiten

6.1 UML-Darstellung und C-Implementierung



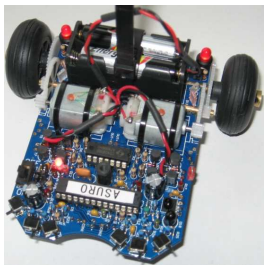
Ausgehend von der bis hierher entstandenen Software-Architektur müssen nun konkrete C-Module identifiziert werden. Dabei ist zu beachten, dass jede Software-Schnittstellen eine Implementierung in einem entsprechenden C-Modul enthält und jede Software-Schnittstelle von anderen Softwareelementen mindestens einmal genutzt wird.

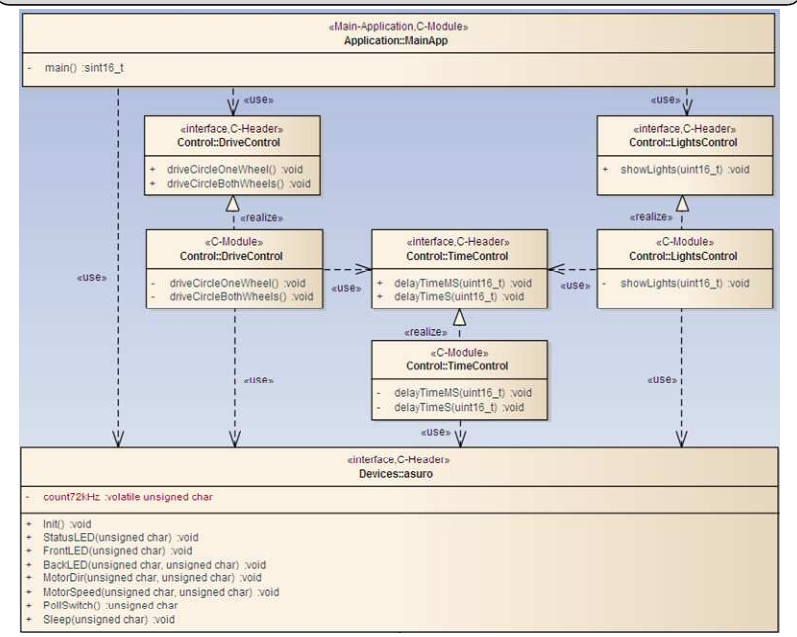
Das C-Modul ist mit der UML über eine vollwertige Klasse notierbar. Diese Klasse bekommt den Stereotype <<C-Module>> zugeordnet.
Der Implementierungspfad einer Software-Schnittstelle in einem C-Modul wird durch die Interface- bzw. Realisierungsrelation dargestellt. Der Nutzungspfad einer Software-Schnittstelle 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.

6.2 Beispiel: Roboter-Software

C-Module und deren Anhängigkeiten – Beispiel: Roboter-Software (1..)

Wie sehen die **C-Module** in der **Roboter-Software** aus?

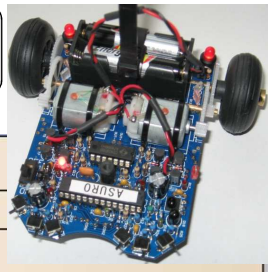




MICRO CONSULT

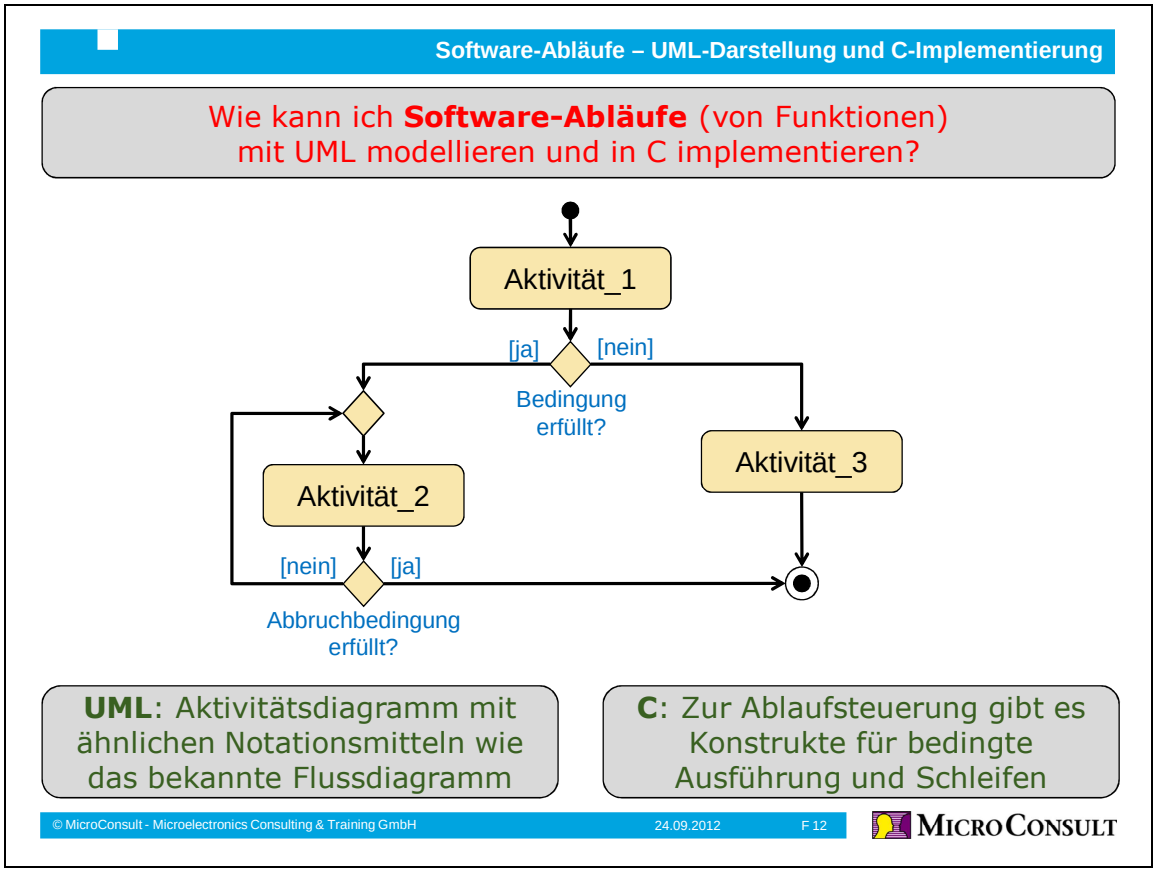
C-Module und deren Abhängigkeiten – Roboter-Software (..2)

Wie sehen die **C-Module**
in der **Roboter-Software** aus?



7 Software-Abläufe

7.1 UML-Darstellung und C-Implementierung



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.

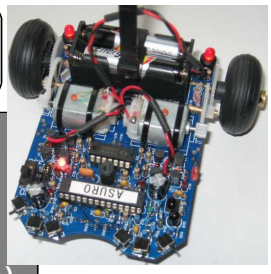
Für Software-Abläufe, die wenig ereignisgesteuert und mehr datenflussorientiert sind, ist das Aktivitätsdiagramm zu bevorzugen. Für Software-Abläufe, die mehr ereignisgetriebenen sind, ist das Zustandsfolgediagramm zu bevorzugen. Dies wird hier aber weiter nicht besprochen. Beide Diagrammarten enthalten 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.

7.2 Beispiel: Roboter-Software

Software-Abläufe – Beispiel: Roboter-Software (1..)

Wie sieht ein Beispiel für einen **Software-Ablauf** in der **Roboter-Software** aus?

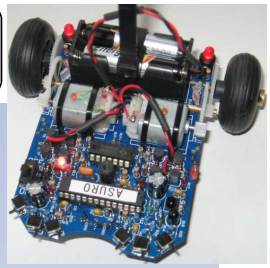


```
void showLights(uint16_t parLoopCount)
{
  uint16_t locCount = 1;
  StatusLED(RED); BackLED(ON, ON);FrontLED(ON);

  for (locCount = 1; locCount <= parLoopCount; locCount++)
  {
    StatusLED(OFF); delayTimeMS(150);
    StatusLED(RED);

    BackLED(ON, OFF); delayTimeMS(150);
    BackLED(ON, ON);

    BackLED(OFF, ON); delayTimeMS(150);
    BackLED(ON, ON);

    FrontLED(OFF); delayTimeMS(150);
    FrontLED(ON);
  }
  StatusLED(OFF); BackLED(OFF, OFF); FrontLED(OFF);
}
```

 24.09.2012 F 13 MICRO CONSULT

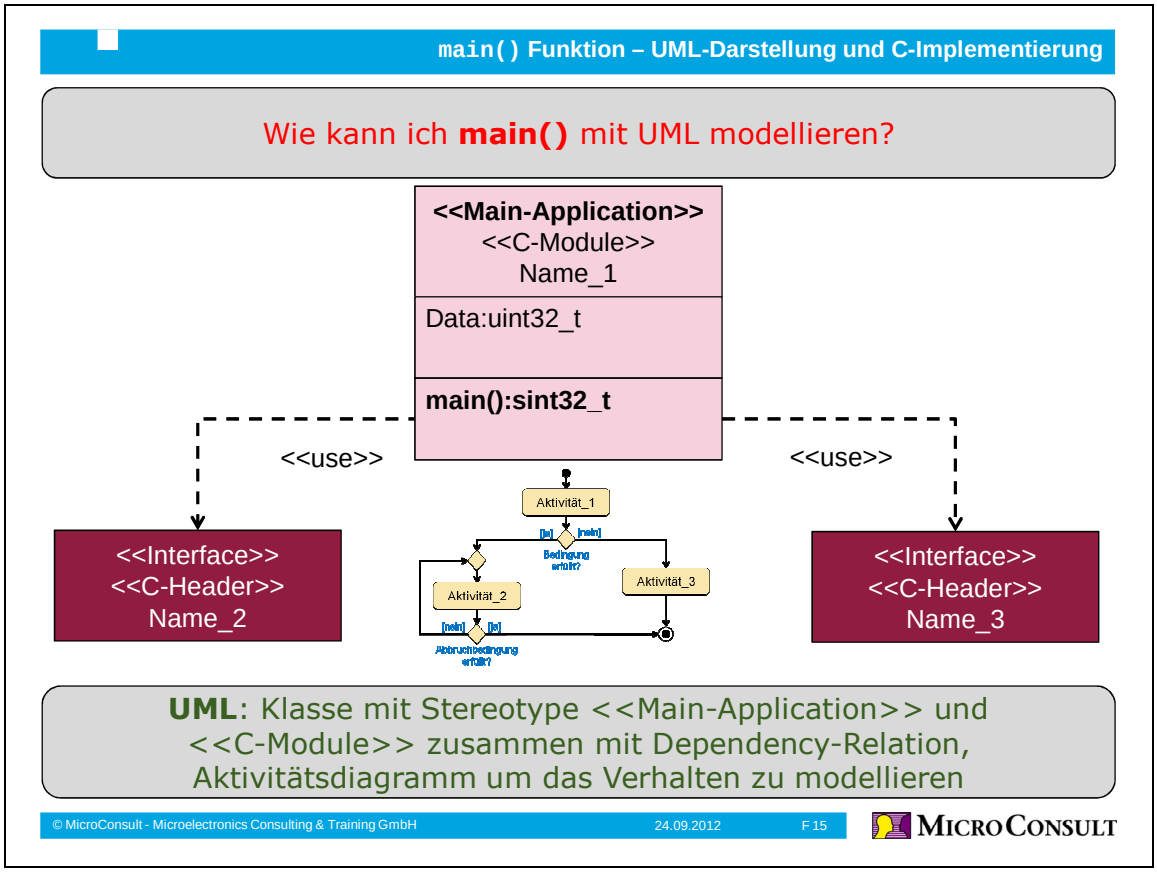
Software-Abläufe – Beispiel: Roboter-Software (..2)

Wie sieht ein Beispiel für einen **Software-Ablauf** in der **Roboter-Software** aus?



8 main() Funktion

8.1 UML-Darstellung und C-Implementierung



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>>.

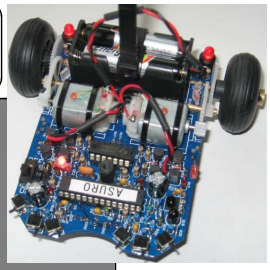
Alle inkludierten Elemente werden mit der Dependency Relation zusammen mit dem Stereotype <<use>> abgebildet.

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

8.1.1 Beispiel: Roboter-Software

main() Funktion – Roboter-Software (1..)

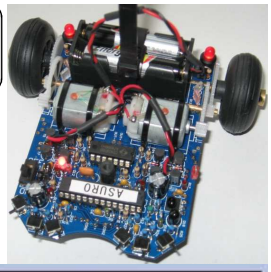
Wie sieht **main()**
in der **Roboter-Software** aus?

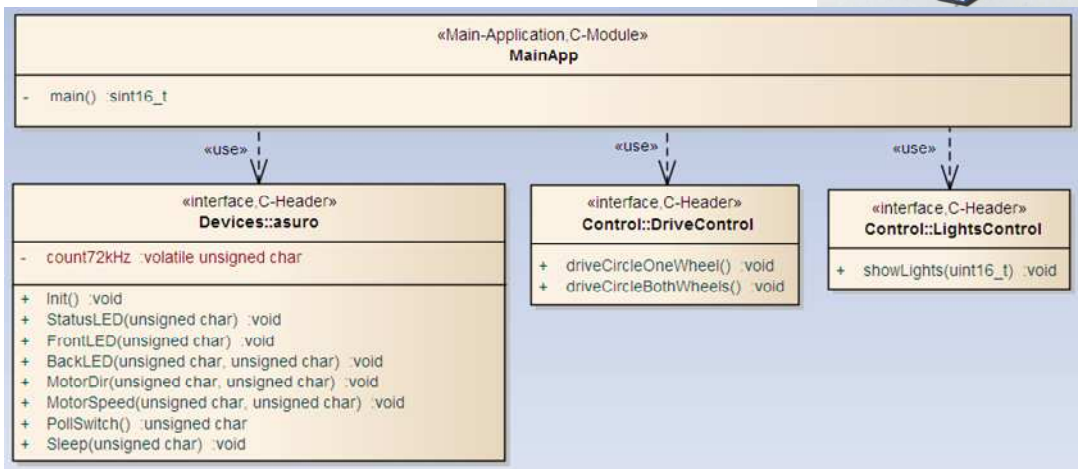



```
#include "asuro.h"
#include "DriveControl.h"
#include "LightsControl.h"

sint16_t main(void) {
  uint8_t locKeyCode = 0;
  Init();
  switch(locKeyCode = PollSwitch()) {
    case 0: {  //no key pressed
      while(1) { showLights(20); }
    } break;
    case 32: { //K1 pressed
      while(1) { showLights(20); driveCircleOneWheel();}
    } break;
    case 16: { //K2 pressed
      while(1) { showLights(20); driveCircleBothWheels();}
    } break;
    default: {  //all other keys
      StatusLED(GREEN); while(1);
    } break;
  } return 0;
}
```

main( ) Funktion – Beispiel: Roboter-Software (..2..)

Wie sieht **main()**
in der **Roboter-Software** aus?

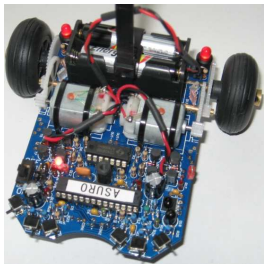


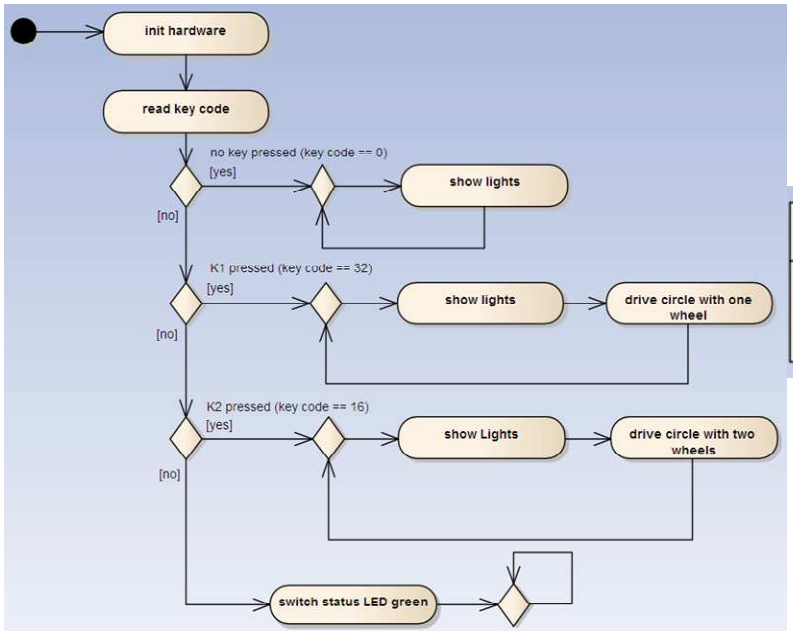


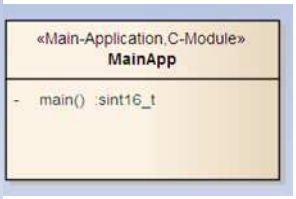
 24.09.2012 F 17 MICRO CONSULT

main( ) Funktion – Beispiel: Roboter-Software (..3)

Wie sieht **main()**
in der **Roboter-Software** aus?







9 Zusammenfassung

Zusammenfassung (1..)

Software-Elemente	UML-Notation	Anmerkungen zur C-Implementierung
Software-Schichten	▪ Pakete im Paket- oder Klassendiagramm ▪ Stereotype <<SW_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>> ▪ Dependency 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 ein dynamische Bindung zu bekommen, gibt es komplexere Implementierungen

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.

Zusammenfassung (..2)

Software-Elemente	UML-Notation	Anmerkungen zur C-Implementierung
C-Module	▪ Klasse 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 Varianten der Implementierung
Software-Abläufe	▪ Aktivitätsdiagramm im Kontext einer Funktion (oder Klasse) ▪ Zustandsfolgediagramm im Kontext einer Funktion (oder Klasse)	▪ Einsatz typischer Kontrollstrukte wie `switch-case` bzw.`if-else` ▪ Aufwändiger mit Funktionszeigern
`main()` Funktion	▪ Operation in einer Klasse „Application“ ▪ Stereotype <<Application>> und <<C-Module>>	Keine