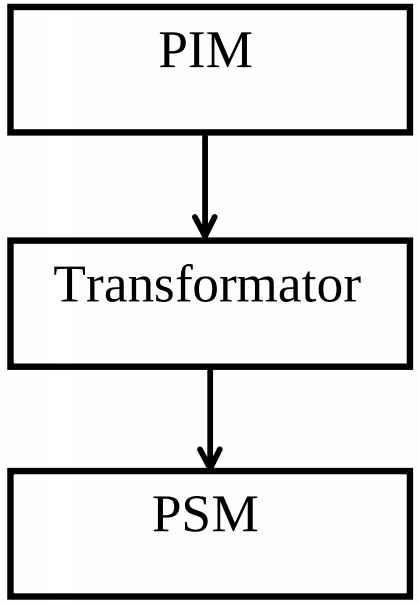
Making Ideas a Reality


# Das Raven MDA Profil für die Modellierung sicherheitskritischer Systeme

Klaus Wachsmuth

# UML und MDA

- Modelle nehmen *zentrale* Stellung im Entwicklungsprozess ein
- Erstellen von Platform Independant Model‘s (PIM)
  - Verwendung von UML Notation
  - Fokus liegt auf der Domäne
- Abbilden auf Platform Specific Model‘s (PSM)
  - Erweiterung des PIM durch die technische Infrastruktur
  - Durch Transformatoren
    - Maximizing automation of the mapping step is the goal (MDA Paper, OMG)
  - Von Hand
- Source Code ist ein PSM




# Ziel ist ein lauffähiges System

- Es muß Code erzeugt werden
  - Ada, Java, C++, C, ..
- Es muß getestet werden
  - Testfälle, Test Scripte, ...
- Es muß dokumentiert werden
  - HTML, XML, RTF, ...

=>

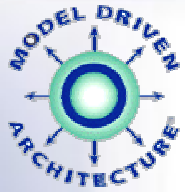
**Umsetzung** der Modelle in die Implementierung
**Konsistenz** der verschiedenen Entwicklungsstufen


# MDA und Profile

- Ideal für MDA ist die Bereitstellung semantisch eindeutiger UML Entwurfselemente mit hohem Abstraktionsgrad durch UML Profilen
- Profil
  - Definiert einfache Entwurfselemente und komplexe Entwurfsmuster
  - Definiert Abbildungsvorschriften  von PIM nach PSM
  - Nutzen der Erweiterungsmechanismen der UML
  - Vordefinierte Stereotypes, Constraints, Tagged Values
  - Profile müssen dokumentiert sein
- Einfluss von Profilen
  - Auf die Modellierung
  - Auf die Transformation der Modelle
- Profile sind domainspezifisch

# Ein auf MDA basierender Entwicklungsprozess

Aonix




PIM

High level
UML Model

*Fachliche Aspekte*

*Technische Aspekte*

Transformation rules

Transformation
Engine

Abbildung UML -> Ada
Design Patterns
RT Mechanismus

PSM

Ada/Raven

# OMG Profil für RT und Safety

OMG Standard

„Profile for Schedulability, Performance, and Time Specification“ (SPT)

- Niedrige Abstraktionsebene
  - Es müssen viele technische Details modelliert werden
  - keine vordefinierten Kommunikationsmuster
- MDA Ziel:
  - möglichst viel transformieren
  - wiederverwendbare Muster bereitstellen

**Höhere Entwurfselemente bereitstellen und deren Semantik über Stereotypen definieren**


# Ravenscar Profil

- Industriestandard für sicherheitskritische Realzeitsysteme mit Ada

Idee des Profils:

- Natürliche Parallelität der Anwendung durch Strukturierung einer Applikation in eine Menge unabhängiger Tasks realisieren
  - zyklische
  - sporadische
  - kooperierende
- die über Ereignisse und Puffer miteinander kommunizieren
- Einschränkung auf eine zertifizierbare Sprachuntermenge


# Realtime Situation

Menge von konzeptuellen oder physischen parallelen Tasks

- Tasks haben Hard- oder Soft-Deadlines
- Tasks mit Hard-Deadlines sind periodisch

**Schedulability Aanalyse:** Könnnen Tasks so ausgeführt werden, dass alle Deadlines getroffen werden?

Statische und dynamische SchedulabilityAlgorithmen, z.B.

- Rate Monotonic Scheduling (statische Prioritäten)
- Earliest Deadline First (Dynamische priorities)

# Realtime Situation 2

## Task Scheduling

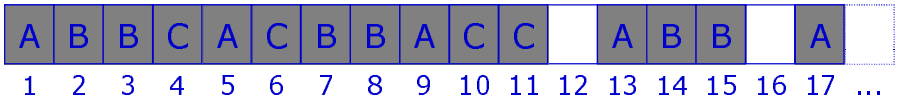
**Analysis**

Parallel tasks

| | |
|---|---|
| Task A | Period 4, Duration 1 |
| Task B | Period 6, Duration 2 |
| Task C | Period 8, Duration 2 |

**Execution**

Sequential processing



# Schedulability

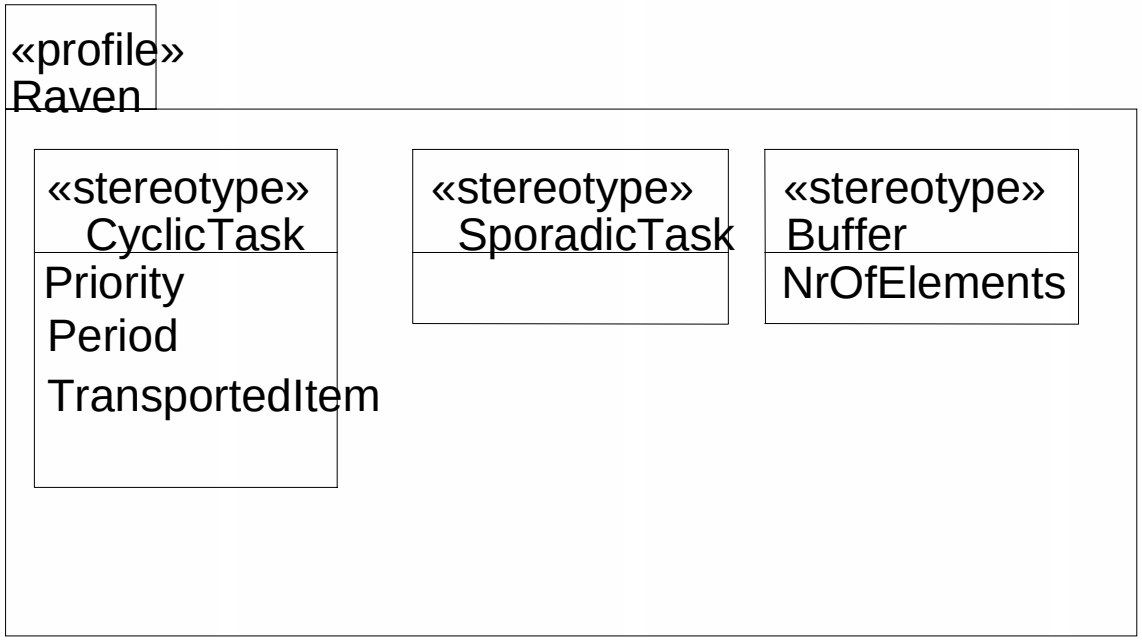
Zusammenarbeit von Tasks, Scheduler und Prozessor

**Rate-Monotonic-Analyse:** vereinfachte Voraussetzungen

- Uni Prozessor System
- Tasks sind unabhängig
- Realtime tasks sind alle periodisch
- Excecution times und Ressourcen alle Tasks sind bekannt

# Raven-Profil

Einflüsse für das Raven Profile: AdaRaven Subset, *SPT* und dem *ARINC 653* Standard.



# RC: Musterelemente aktiv (Beispiele)

- RepetetiveTask
  - Endlos. Ohne feste Periode
  - Tagged Values: priority, stacksize
- CyclicTask
  - wie RepetetiveTask, jedoch mit fester Periode
  - Tagged Values: priority, stacksize, period

=>Transporter Muster

  - wie cyclic, jedoch
  - es gibt eine Get-Association
  - es gibt eine Put-Association
  - Item_Type

- SporadicTask
  - wartet auf einen Event oder einen Interrupt
  - hat eine Assoziation zu einer „Event“-Klasse

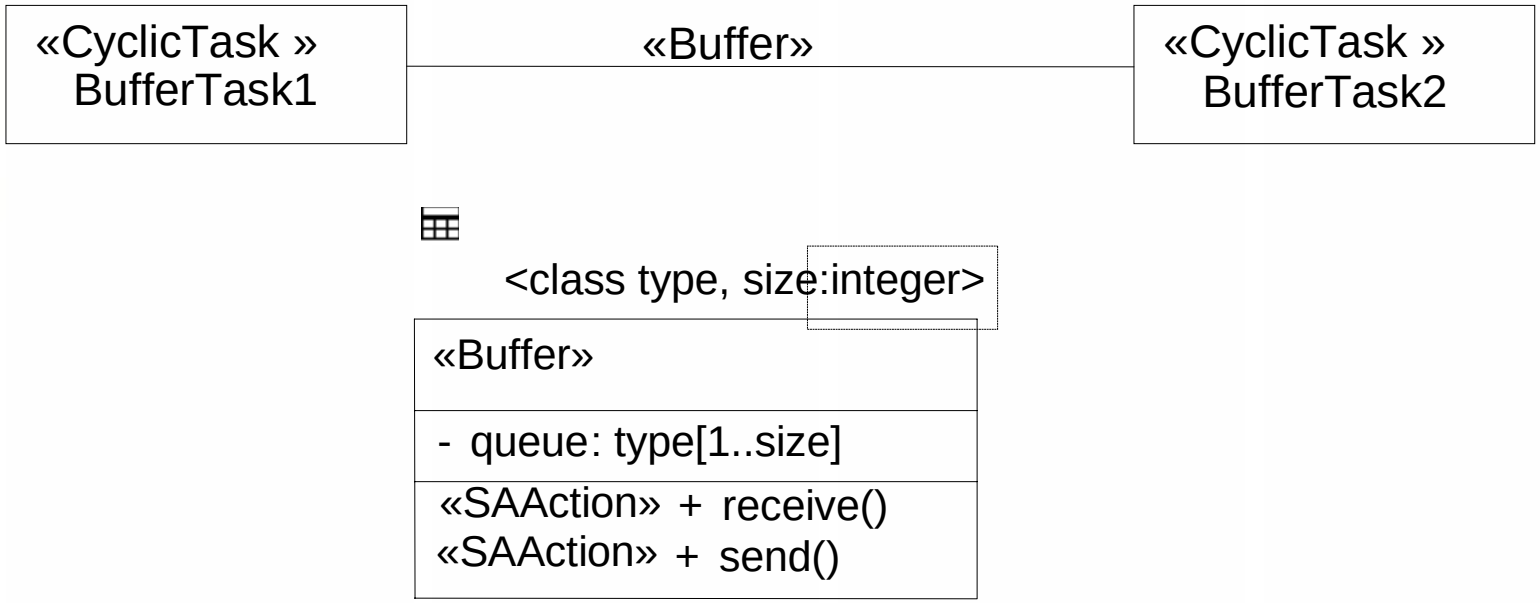
# Kommunikation Muster

Stereotypen für die Darstellung fundamentaler Kommunikation Muster aus dem *ARINC 653* Standard:

- Buffer
- Blackboard (ResourceControl)
- Event
- Interrupt

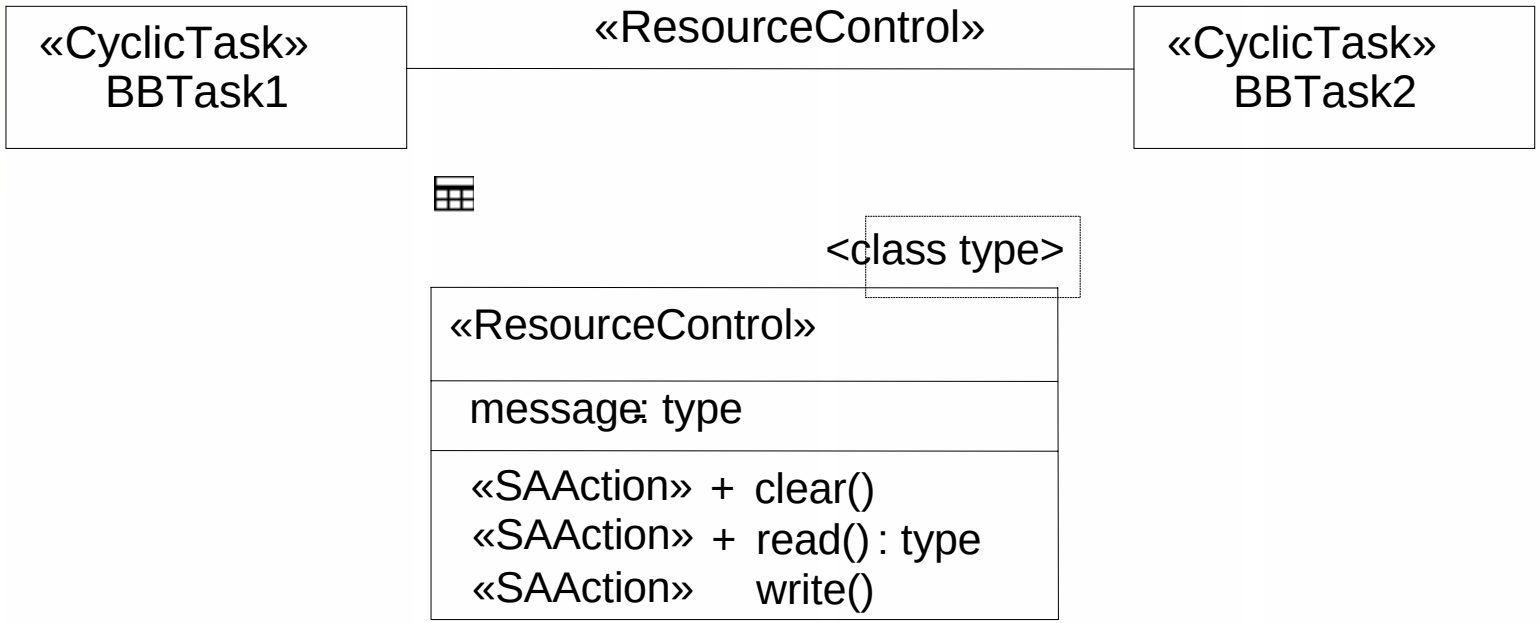
# Buffer

**Buffer:** Messages werden mittles eines FIFO Puffers mit vordefinierten Grösse übertragen.



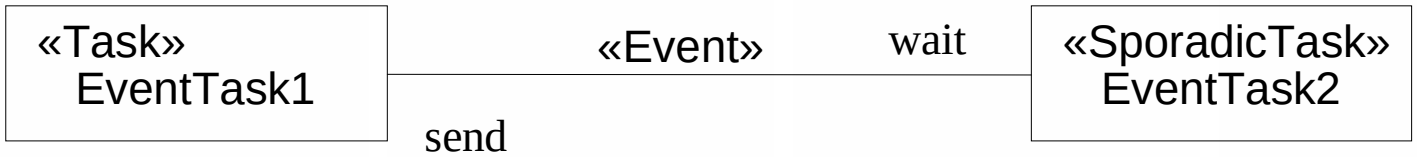
# Blackboard or ResourceControl

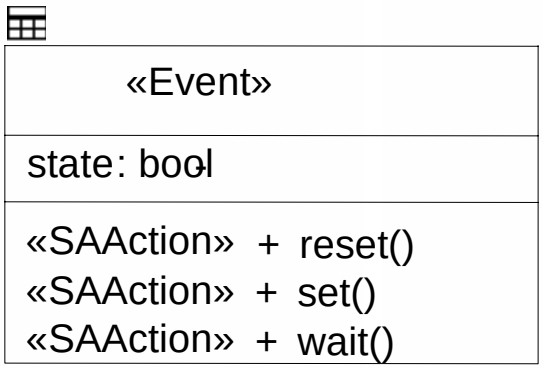
**Blackboard:** Ungepufferte Übertragung von Messages. Eine Message in einem Board steht solange zur Verfügung, bis sie überschrieben wird.



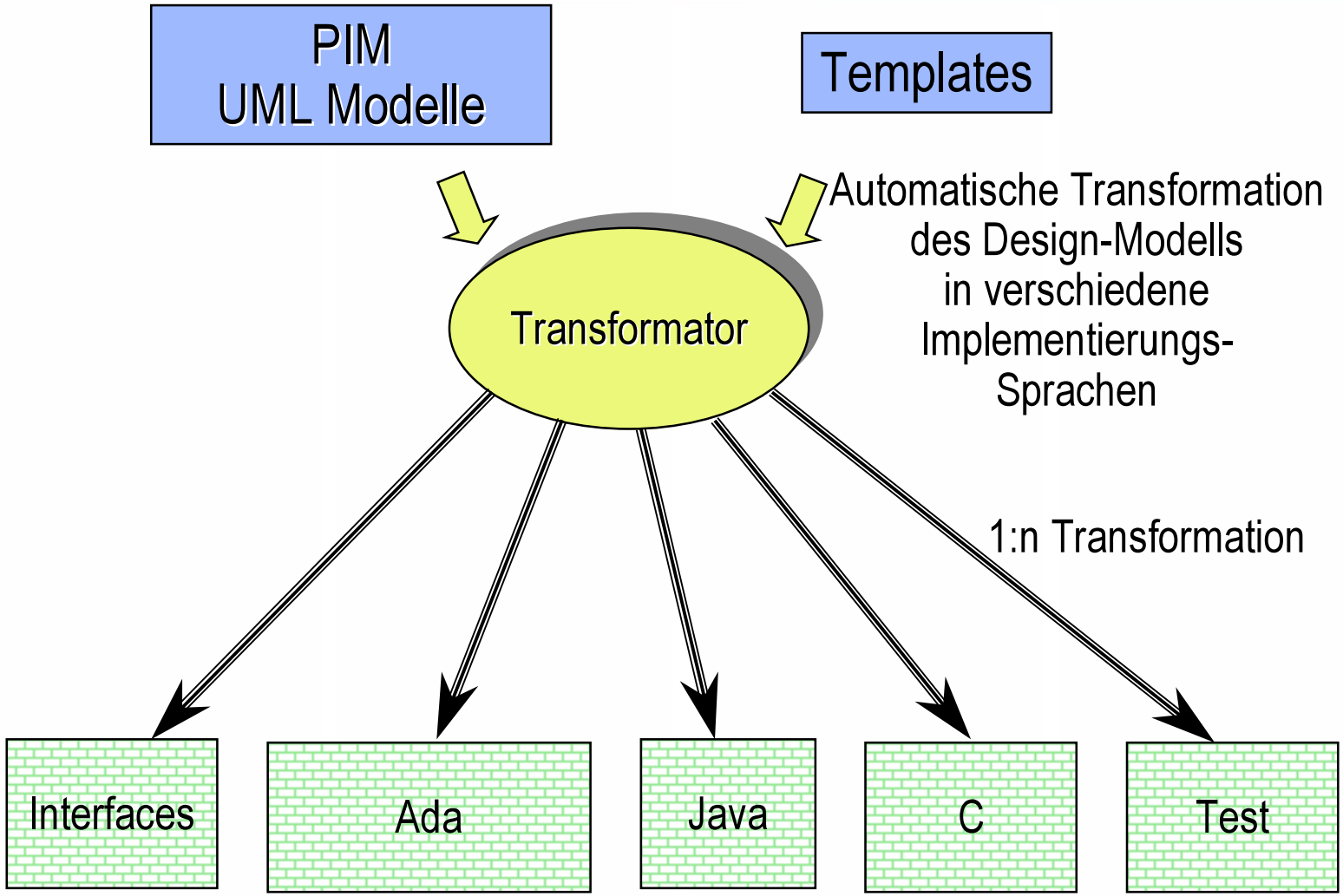
# Events

**Events:** Asynchrone Benachrichtigung über ein Ereignis



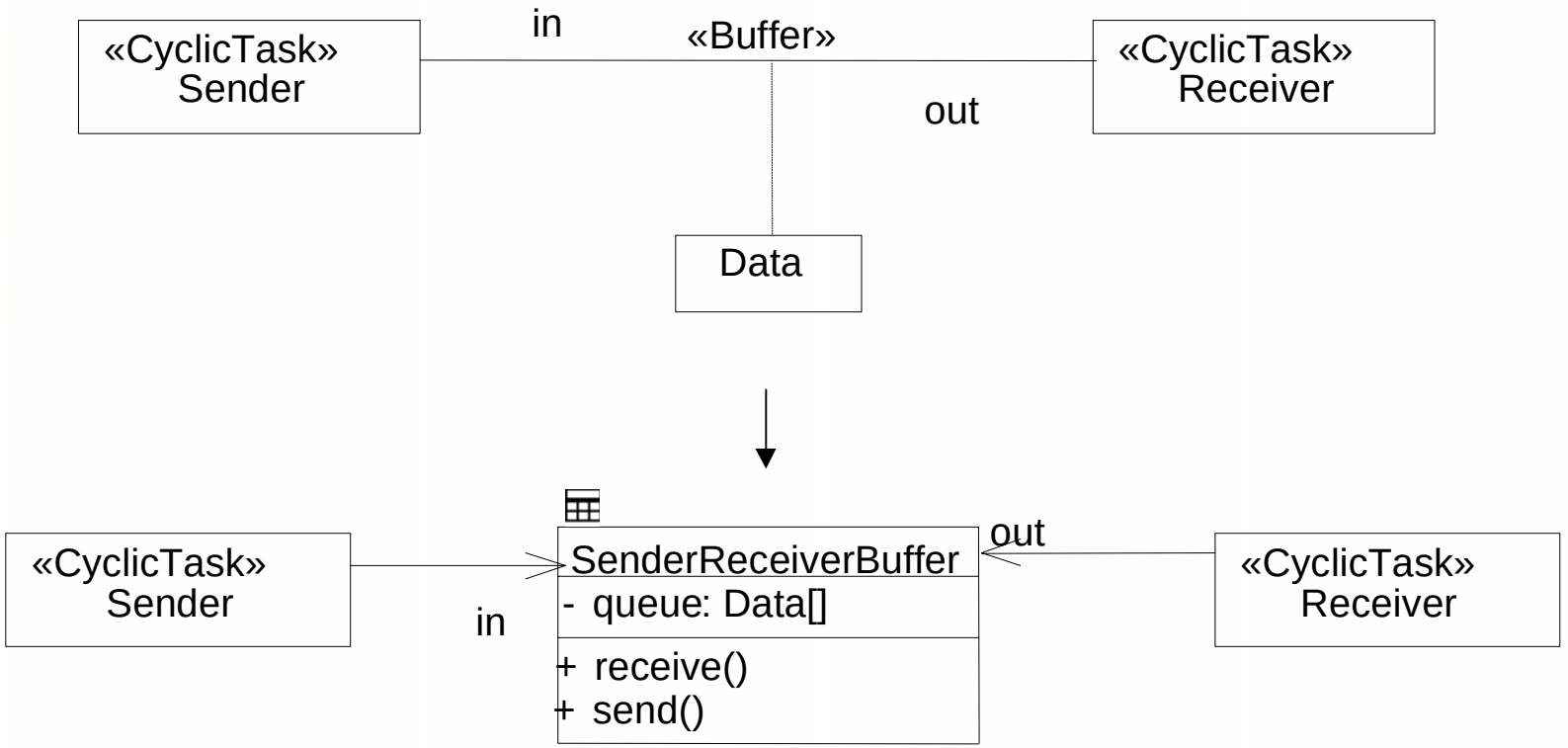



# Codegenerierung 1

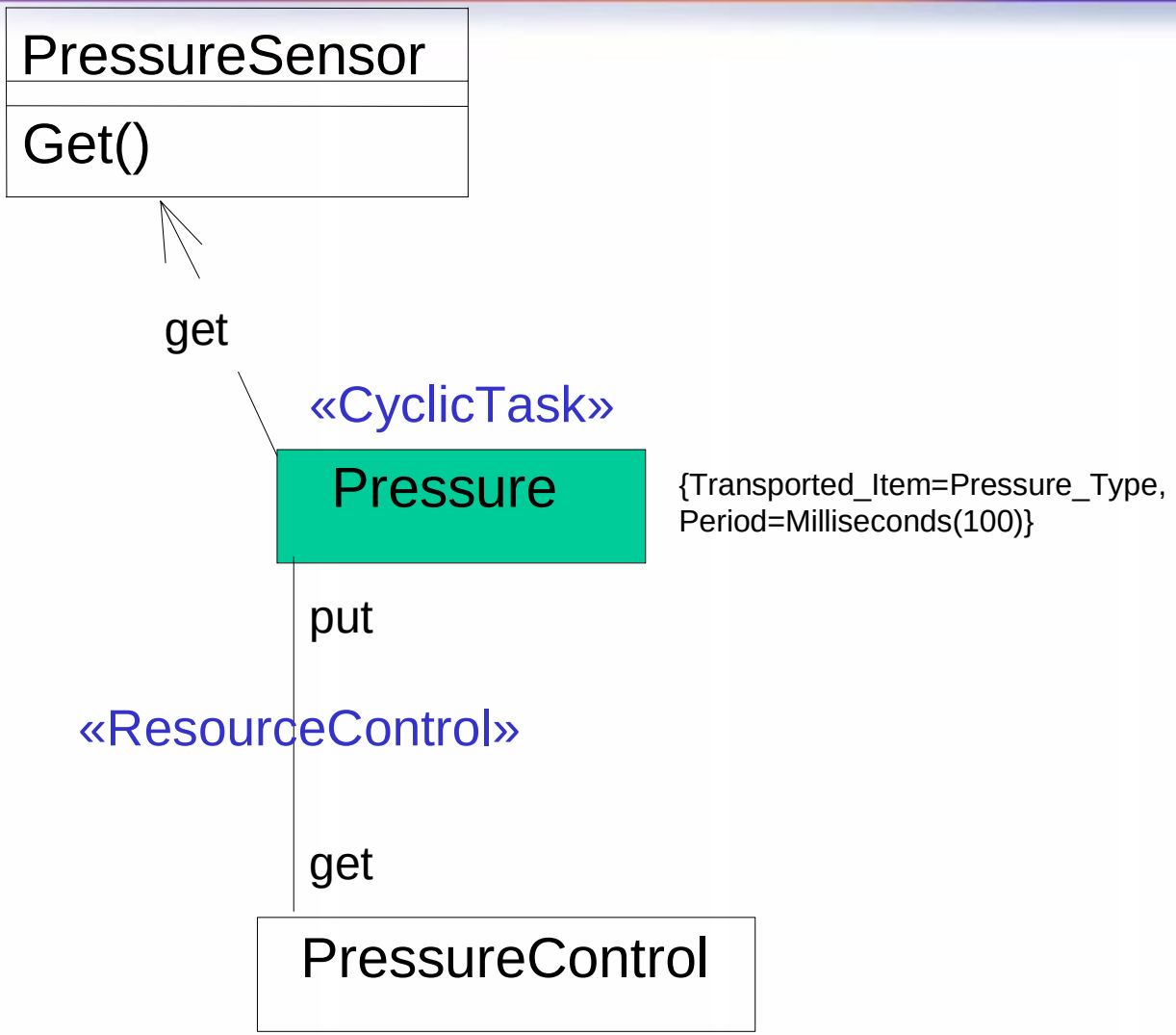


# Code Generierung 2

Abbildung von highlevel Modellierung in einfachere Assoziationen (Modell Transformation => MDA)



# Kommunikationschema Transporter



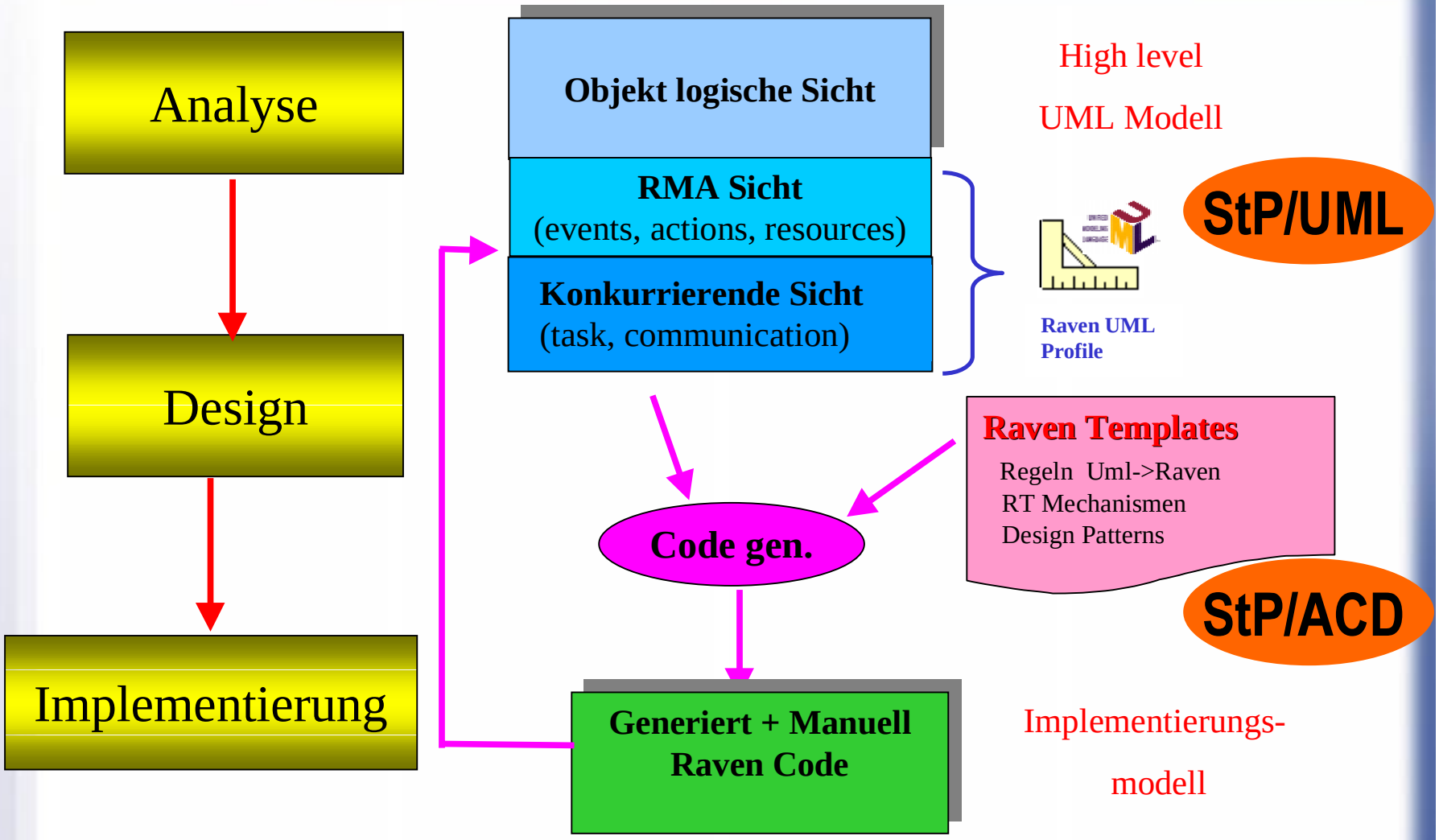
# Generierter Code: Transporter Task

```
with PressureSensor_Pkg; with PressureBuffer_Pkg;
with Ada.Real_Time;  use Ada.Real_Time;
package body Pressure_Pkg is
   task body Pressure is
      Next_Time : Ada.Real_Time.Time;
      Period    : constant Ada.Real_Time.Time_Span := Milliseconds(100);
      Item : Pressure_Type;
   begin
      Next_Time := Ada.Real_Time.Clock;
      loop
         delay until Next_Time;
         Item := PressureSensor_Pkg.read; -- Raven Class Package
         --#ACD# M(UCOD:: 102:BOTTOM) User Defined Code
               -- Section for  User Defined Code
         --#end ACD#
         PressureBuffer_Pkg.Write (Item); -- Raven Class Package
         Next_Time := Next_Time + Period;
      end loop;
   end Pressure;
end Pressure_Pkg;
```

# Template: Cyclic Task

```
template CyclicTaskBody(MClass)
with Ada.Real_Time; use  Ada.Real_Time; -- To get visibility to the "+" operator.
package body [MClass.name]_Pkg is
  task body [MClass.name] is
     Next_Time : Ada.Real_Time.Time;
     Period    : constant Ada.Real_Time.Time_Span := [Period([MClass])];
     [transporter_decl([MClass])]
  begin
     Next_Time := Ada.Real_Time.Clock;
     loop
        delay until Next_Time;
        [transporter_read([MClass])]
        [genStateMachineCall([MClass])]
        [mergeOut("UCOD:: "getUniqueId([Mclass,"User Def Code", "")]
        [transporter_write([MClass])]
        Next_Time := Next_Time + Period;
    end loop;
  end [MClass.name];
end [MClass.name]_Pkg;
end template
```

# Der Entwicklungsprozess




# Zusammenfassung

- Einige Muster aus dem Ravenscar Profile
  - sprachunabhängig, aber Abbildung nach Ada natürliche Vorteile
  - Einhaltung der Sprachteilmenge, die sicherheitskritischen Aspekten genügt.
- Hohes Abstraktionsniveau
- Abbildung in Zielsprache durch Template gesteuerte Codegenerierung
- Berücksichtigung von Eigenschaften von Safety-Standards
- Abbildung des Modells auf Code mittels MDA ist durch den Benutzer direkt spezifizierbar und erweiterbar