



ulm university universität
uulm



ActiveCharts

Verknüpfung von Modellen und Code bei der
modellgetriebenen Softwareentwicklung mit UML 2.0

Jens Kohlmeyer | 05. März 2007 |
Institut für Programmiermethodik und
Compilerbau

Übersicht

Unser Ansatz **ActiveCharts** kombiniert Modelle aus den Analyse/Design-Phasen und (handgeschriebenen) Code aus der Implementierungsphase auf eine neue Art:

Der Kontrollfluss einer Anwendung wird mit UML 2.0 Aktivitätsdiagrammen modelliert, die von einer Laufzeitumgebung (ActiveChartsIDE) interpretiert werden. Mit dieser Laufzeitumgebung können die Aktivitätsdiagramme simuliert und visualisiert werden. Funktionen wie beispielsweise Breakpoints oder schrittweises Vorgehen ermöglichen ein einfaches Debuggen der Anwendung.

Analyse- bzw. Designartefakte werden in unserem Ansatz nahtlos für die Implementierungsphase übernommen. Wir wollen damit die Lücken zwischen den unterschiedlichen Phasen im Softwareentwicklungsprozess verkleinern und die Dokumentation und die Wartbarkeit von Softwareprodukten verbessern.

Inhalt

- Motivation
- Forschungsansatz ActiveCharts
- ActiveCharts Architektur
- Beispiel
- Diskussion
- Aktuelle Forschungsarbeiten
- Zusammenfassung und Ausblick

Motivation (1/2)

Probleme bei der Softwareentwicklung

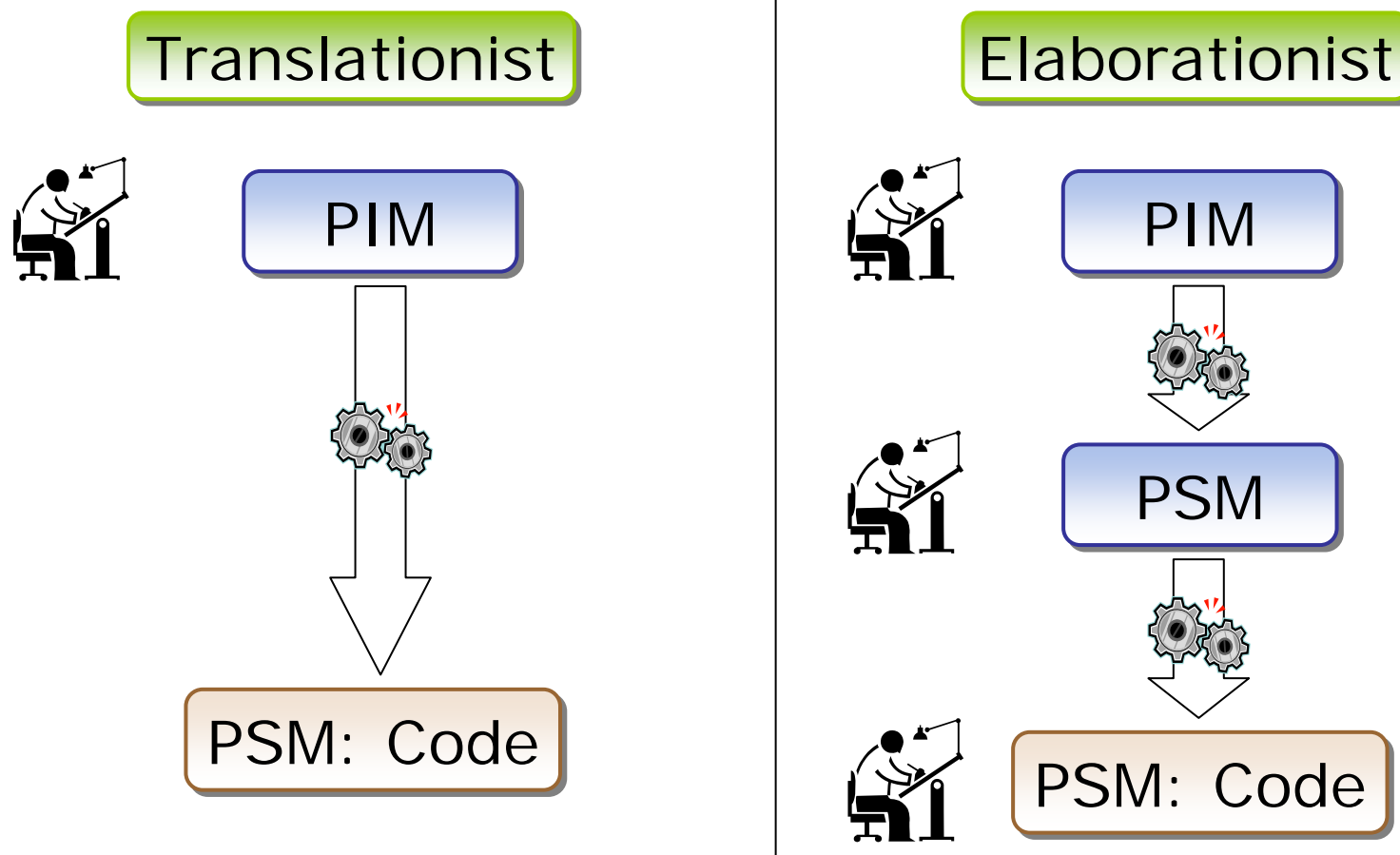
- Lücke zwischen Design-Phase und Implementierung
- Modelle werden nicht wieder verwendet
- Modellinhalte müssen “nachimplementiert” werden
- Dokumentation ist unter Umständen nicht aktuell
- Die Semantik von verwendeten Modellen ist oft unklar
 - Freie Auslegung durch Entwickler kann zu falscher Implementierung führen
 - Fehlende Ausführbarkeit der Modelle kann zu falschem Verhalten führen

Ein Lösungsansatz: Model Driven Architecture, MDA

- Idee: Lücke zwischen Design-Phase und Implementierung schließen
- Anwendungen aus Modellen (und nötigen Erweiterungen generieren)
- Wiederverwendung der Business-Logik durch plattformunabhängige Modelle (PIM)
- Erzeugung von plattformspezifischem Code durch entsprechende Modelle (PSM)

Motivation (2/2)

Zwei Hauptinterpretationen der MDA:



Forschungsansatz ActiveCharts – ein MDA-Ansatz (1/2)

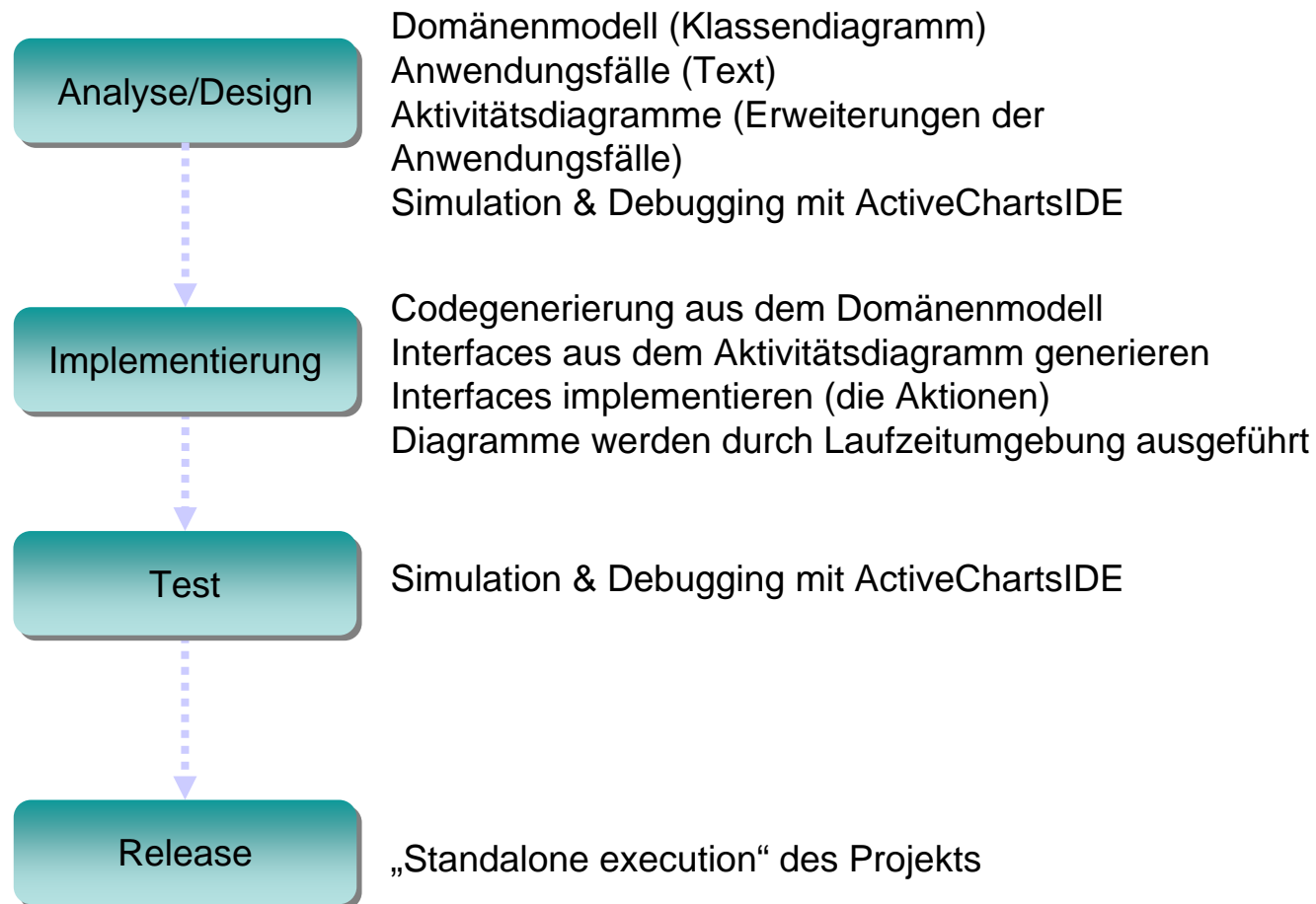
1. Reines „Forward Engineering“

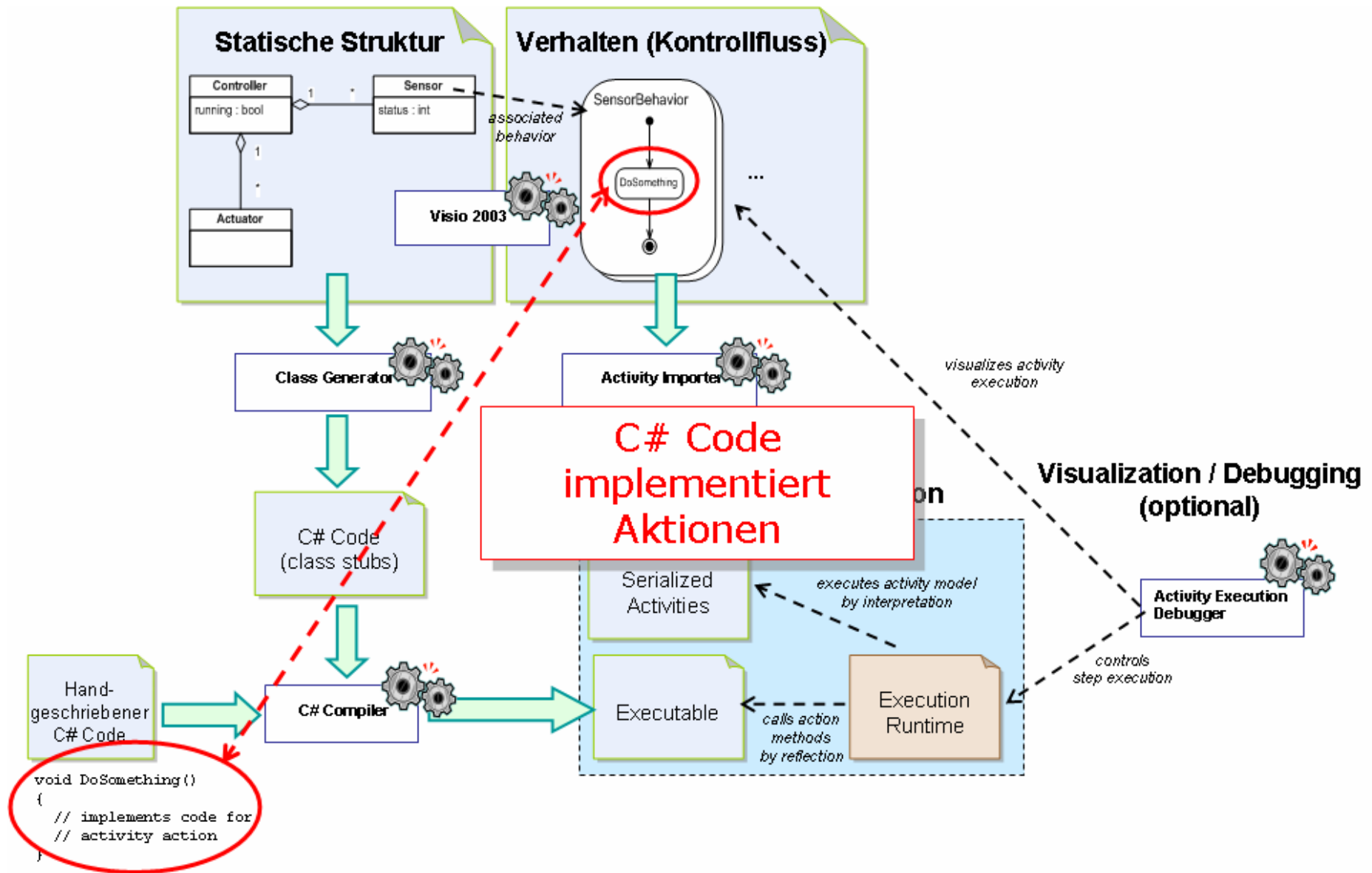
- Beschreiben der statischen Struktur mit UML 2.0 Klassendiagrammen
- Beschreiben des Verhaltens mit UML 2.0 Aktivitätsdiagrammen
 - Modellieren den Kontrollfluss der Anwendung
 - Werden zur Laufzeit interpretiert

2. Verknüpfung mit herkömmlicher objektorientierter Programmiersprache

- C#
- Aktionen aus den Aktivitätsdiagrammen können mit C# Code ausgearbeitet werden

Forschungsansatz ActiveCharts – ein MDA-Ansatz (2/2)





Inhalt

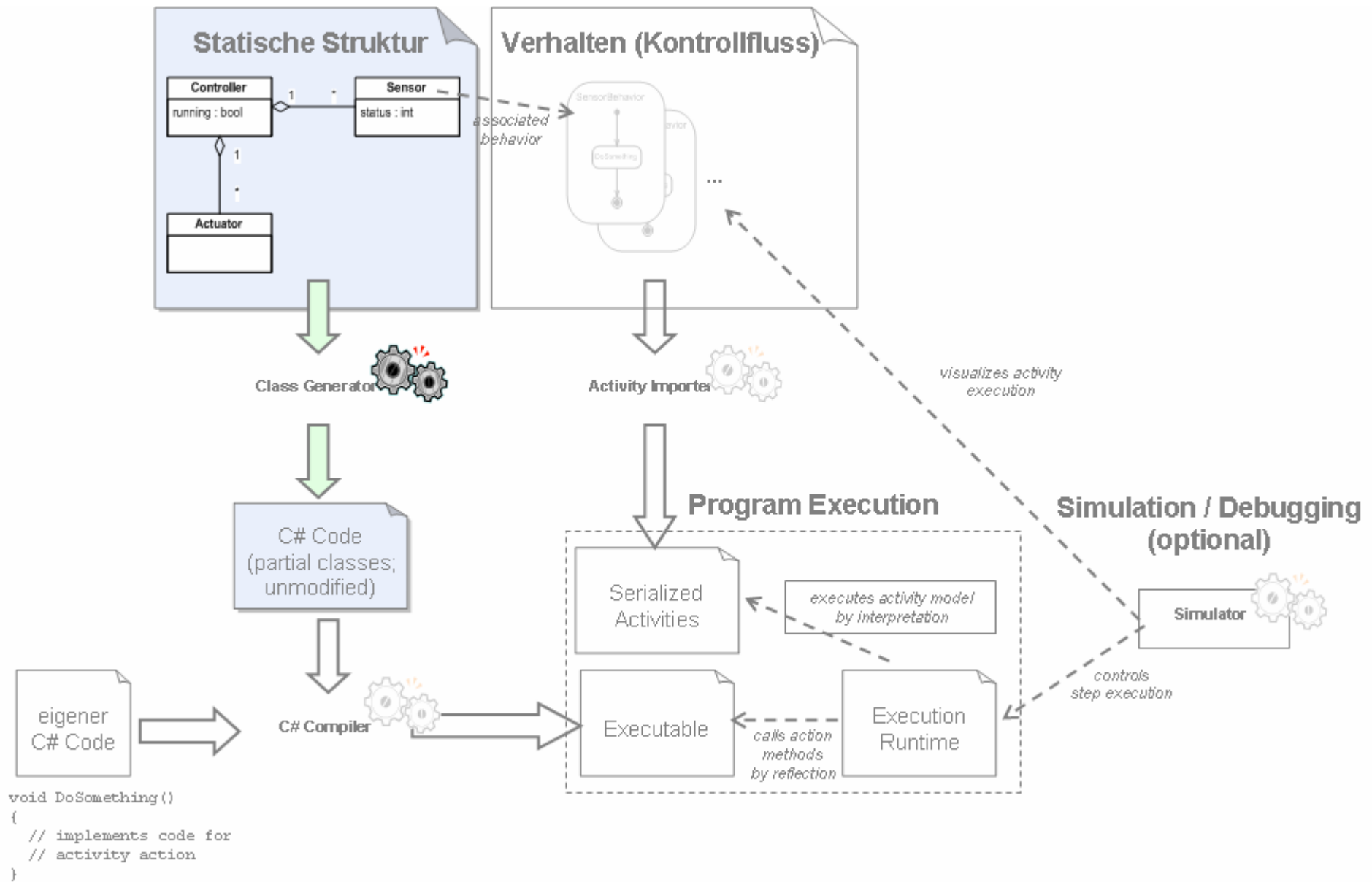
- Motivation
- Forschungsansatz ActiveCharts
- ActiveCharts Architektur
- **Beispiel**
- Diskussion
- Aktuelle Forschungsarbeiten
- Zusammenfassung und Ausblick

Ein Beispiel – eine vereinfachte Alarmanlage

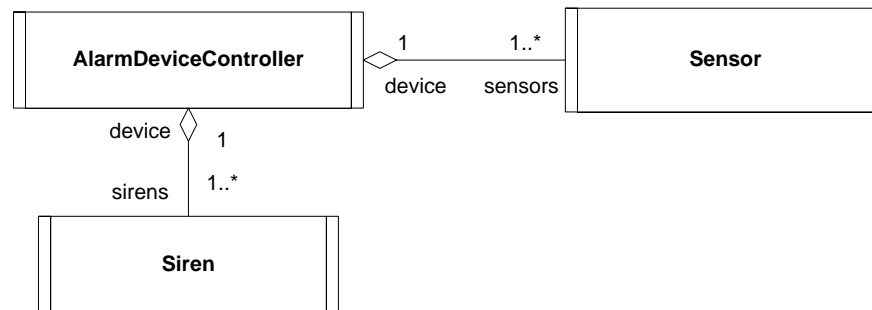
Alarmanlage



- hat Sensoren, Sirenen und einen Controller
- Sensoren senden Heartbeat-Signal
- Wird der Heartbeat nicht innerhalb einer bestimmten Zeit empfangen: **Alarm!**



Vereinfachte Alarmanlage – statische Struktur



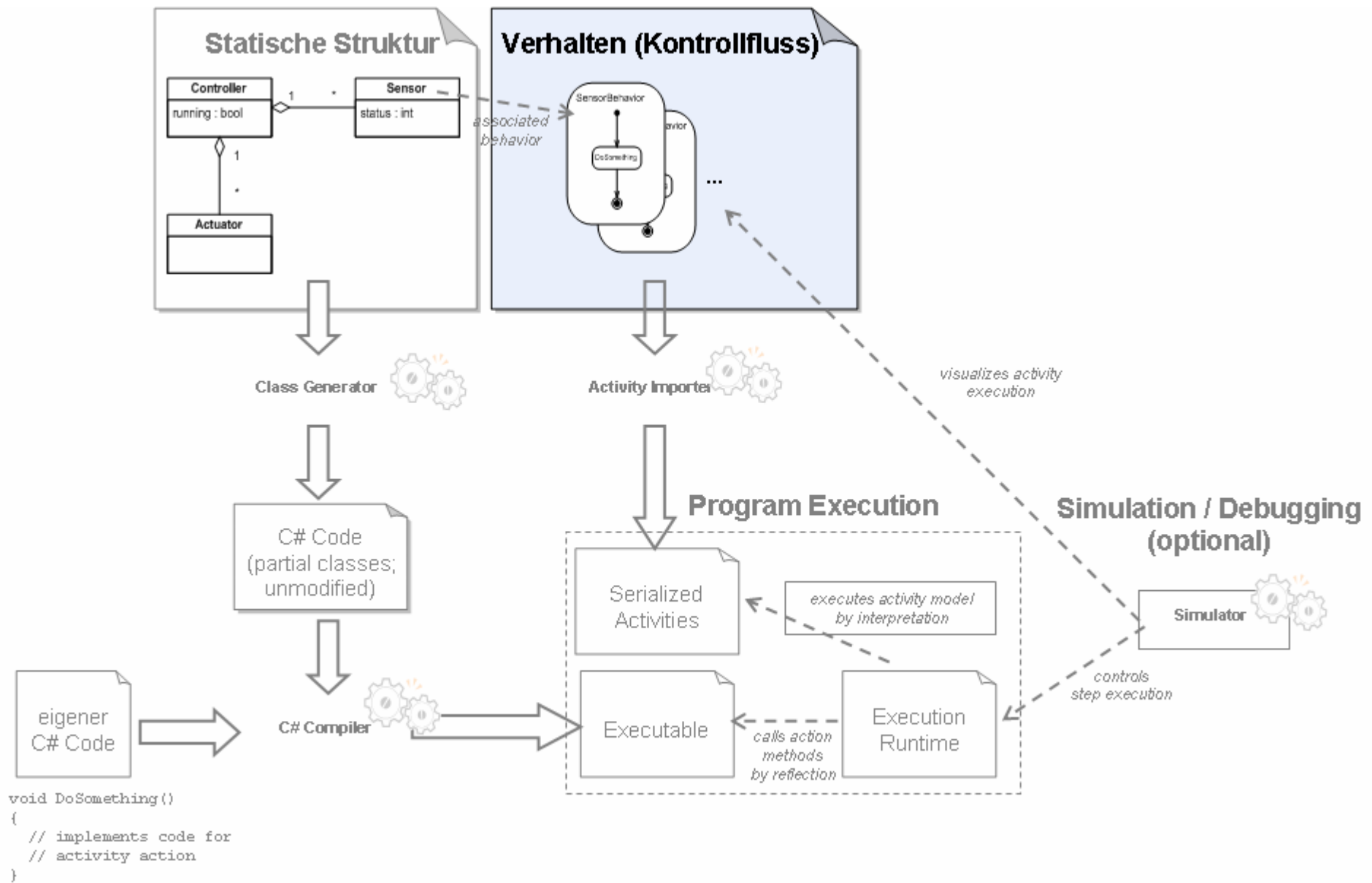
Generator

C# Code, der die Klassen,
Attribute und Assoziationen
implementiert

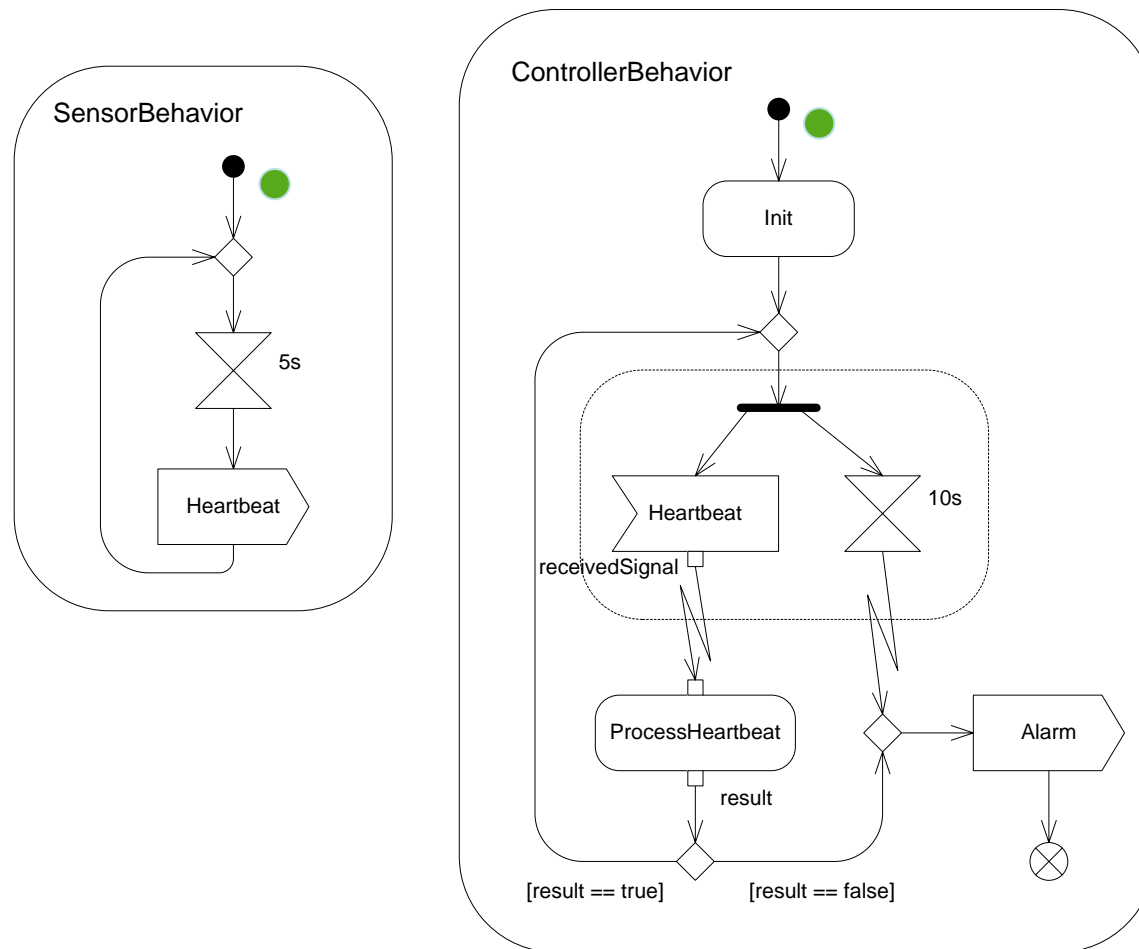


Verwendung in eigenem Code

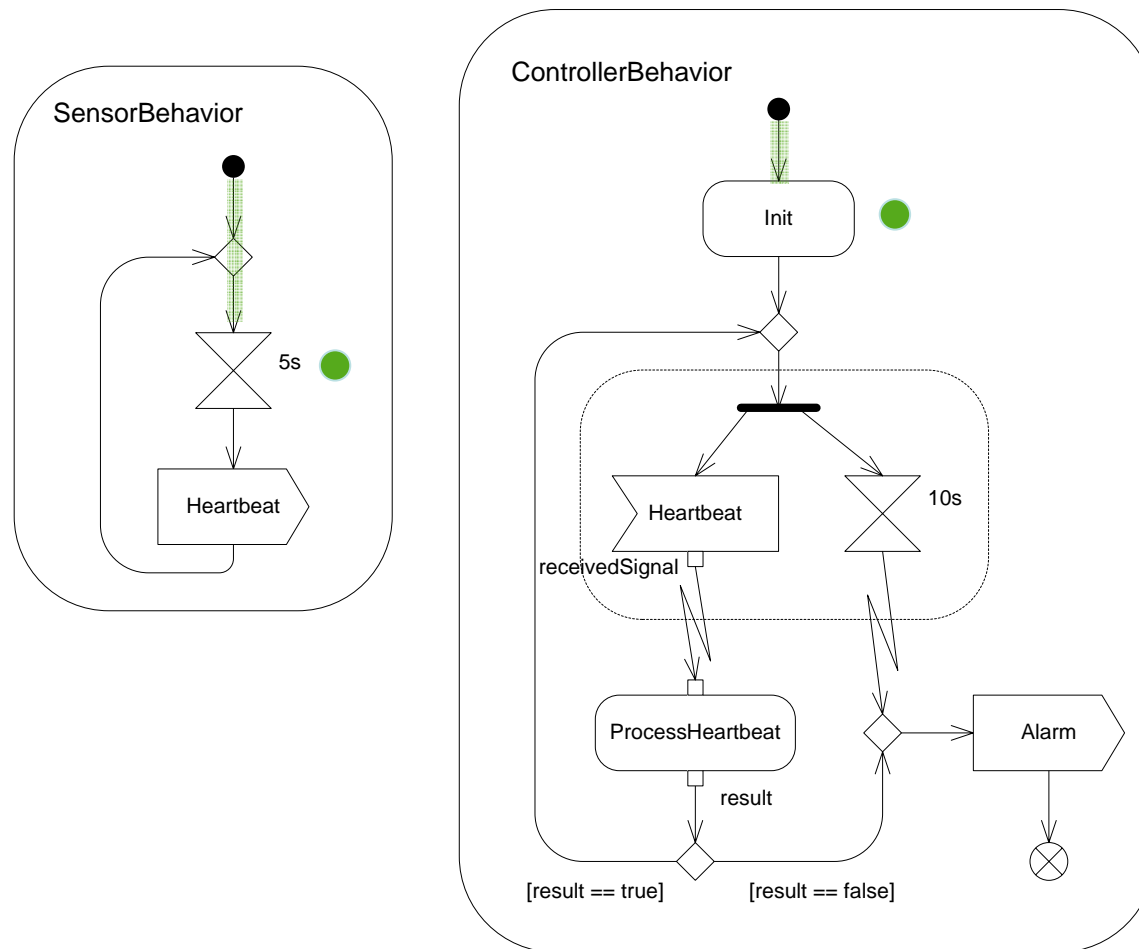
```
AlarmDevice alarmanlage = new AlarmDevice();
Siren sirene = new Siren();
alarmanlage.sirenen.Add(sirene);
```



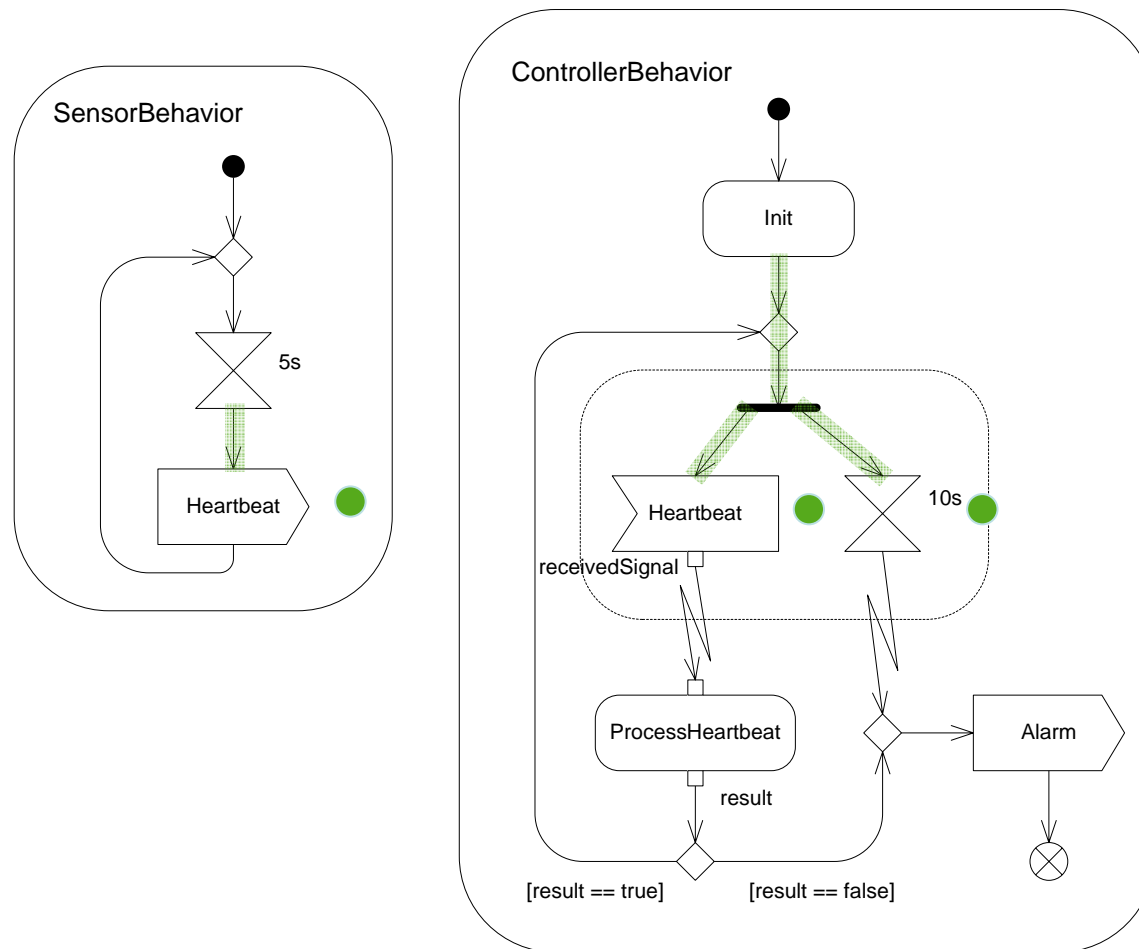
Alarmanlage - Kontrollfluss



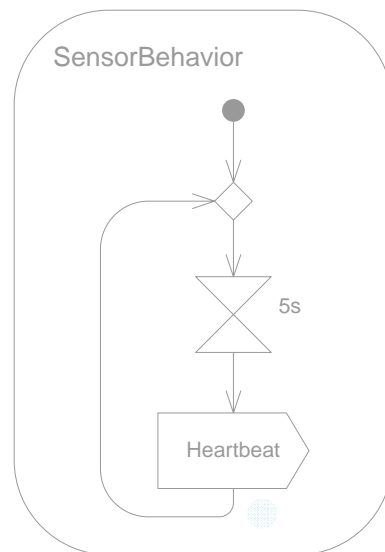
Alarmanlage - Kontrollfluss



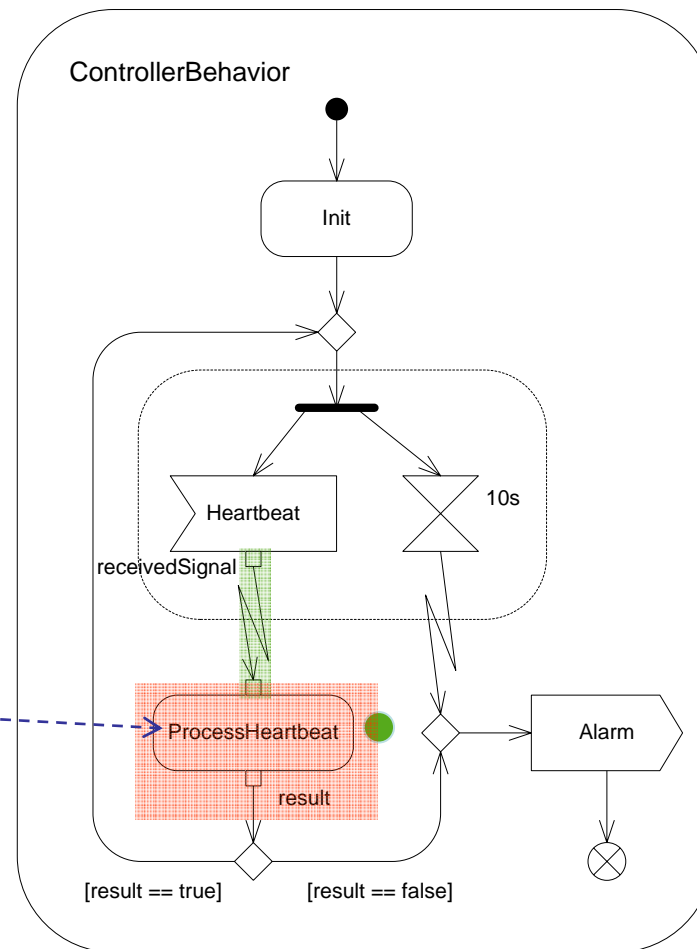
Alarmanlage - Kontrollfluss

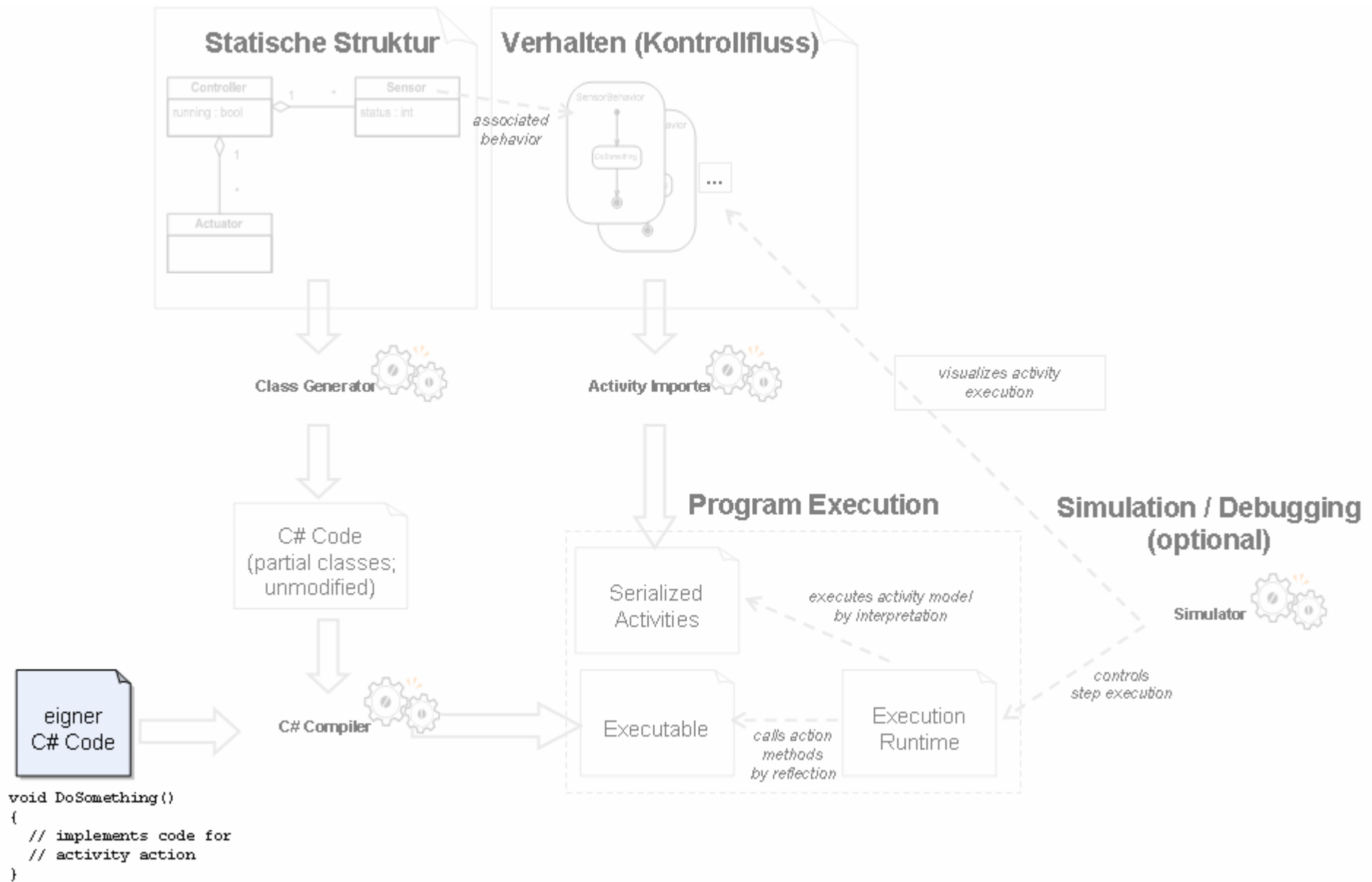


Alarmanlage - Kontrollfluss



Programmiert mit C# Code





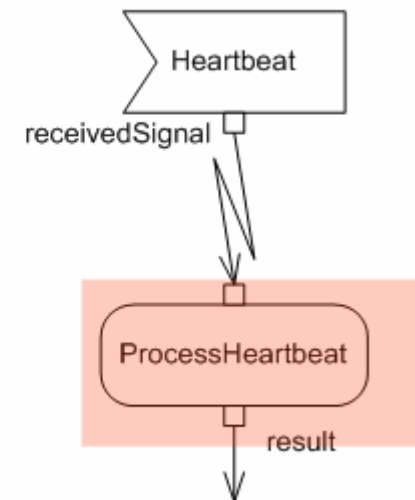
Handgeschriebener Code

ProcessHeartbeat muss überprüfen, ob ein Sensor fehlt

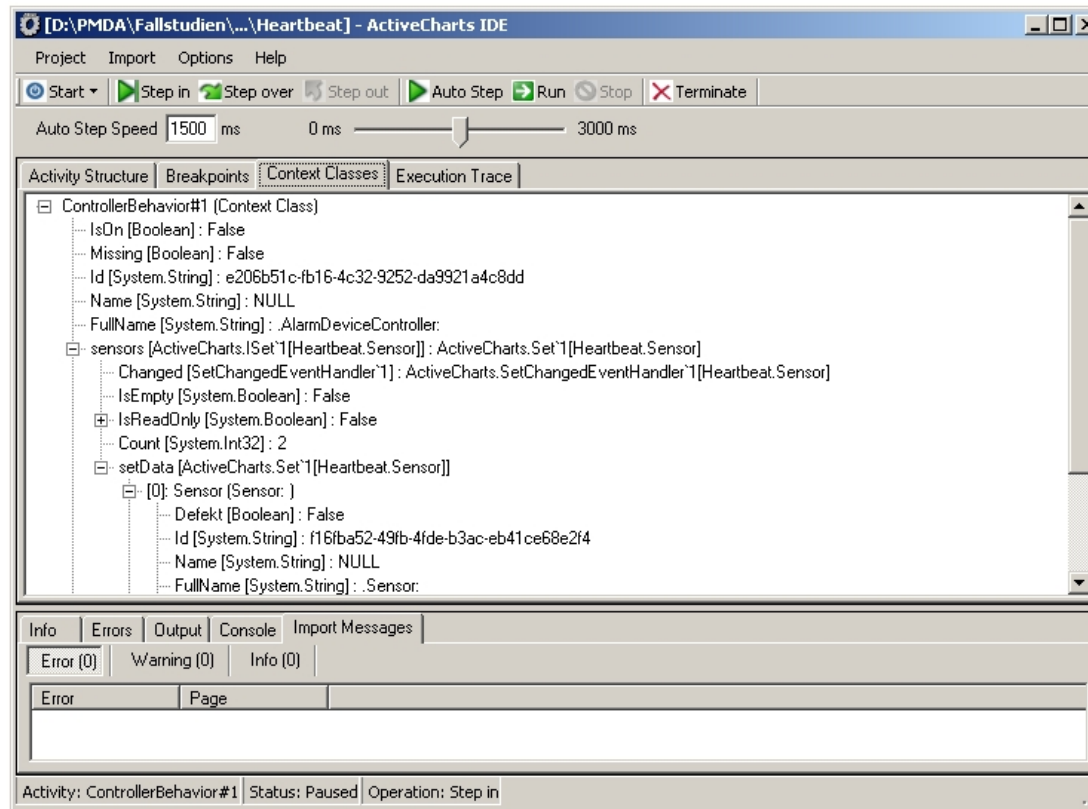
- Für alle existierenden Sensoren die Zeitstempel anschauen
- Falls ein Sensor keinen Heartbeat mehr sendet, Alarm auslösen
- Für den Sensor, der das Signal gesendet hat: Zeitstempel aktualisieren

```
bool ProcessHeartbeat( receivedSignal : heartbeat)
{
    Sensor triggeredSensor = receivedSignal.GetSensor();
    Time t_now = System.CurrentTime();
    bool result = true;

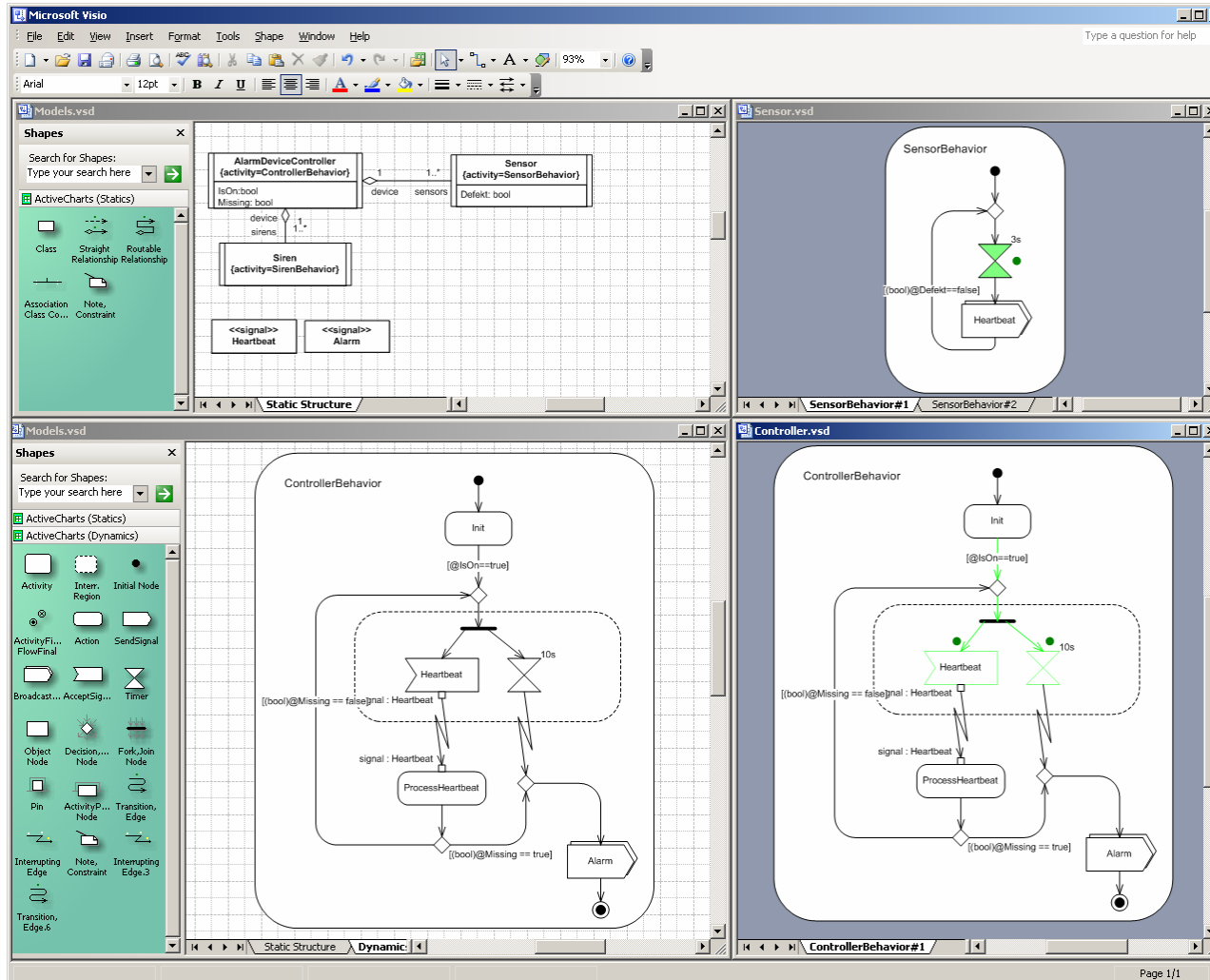
    for(int i=1; i<=CurrentTimestamps.GetCount(); i++)
    {
        if (sensors[i] == triggeredSensor)
            CurrentTimestamps[i] = t_now;
        else
            if (CurrentTimestamps[i] < t_now-6)
                result = false;
    }
    return result;
}
```



ActiveCharts – Screenshots (1/2)



ActiveCharts – Screenshots (2/2)



Diskussion

Die erstellten Analyse/Design-Dokumente sind Teil der Implementierung

- Wiederverwendung der Design-Artefakte
- Die graphische Dokumentation ist immer aktuell
- Kein unnötiges Nachprogrammieren des Kontrollflusses
- Die Modelle könne ausgeführt werden
- Wohldefinierte Semantik

Explizite Schnittstelle zwischen PIM und PSM/Code (Aktionen)

Verwendung einer herkömmlichen Programmiersprache (C#)

Verhältnis von Code und Modellen ist frei wählbar

- Erhöht die Akzeptanz beim Entwickler
- Vereinfacht den Übergang zu modellgetriebener Softwareentwicklung

Formale Beschreibung der Semantik von Aktivitätsdiagrammen der UML 2.0 mit Abstract State Machines

Aktuelle Forschungsarbeiten

- Einbindung von (nichtfunktionalen) Anforderungen in ActiveCharts
- Übersetzungsansatz: übersetzte Komponenten einbinden
- Persistierung des Interpreters (beispielsweise für Testsuite)
- Generierung von Testfällen
- Portierung auf Java
- Wohldefinierte Semantiken für weitere UML 2.0 Diagrammarten (Abstract State Machines)
- Verifizierung von Abstract State Machines

Zusammenfassung und Ausblick

- Unser Ansatz der modellgetriebenen Softwareentwicklung benutzt UML 2.0 Klassen- und Aktivitätsdiagramme
- ActiveChartsIDE unterstützt in diesem Ansatz:
 - Import der erstellten Diagramme
 - Codegenerierung aus Klassendiagrammen
 - Interpretation der Aktivitätsdiagramme zur Laufzeit
 - Visualisierung des Tokenflusses
 - Debugger Funktionalität
- Formale Beschreibung der Semantik mit Abstract State Machines

Weitere Informationen und ActiveChartsIDE-Download unter
www.activecharts.de

Vielen Dank für Ihre
Aufmerksamkeit!