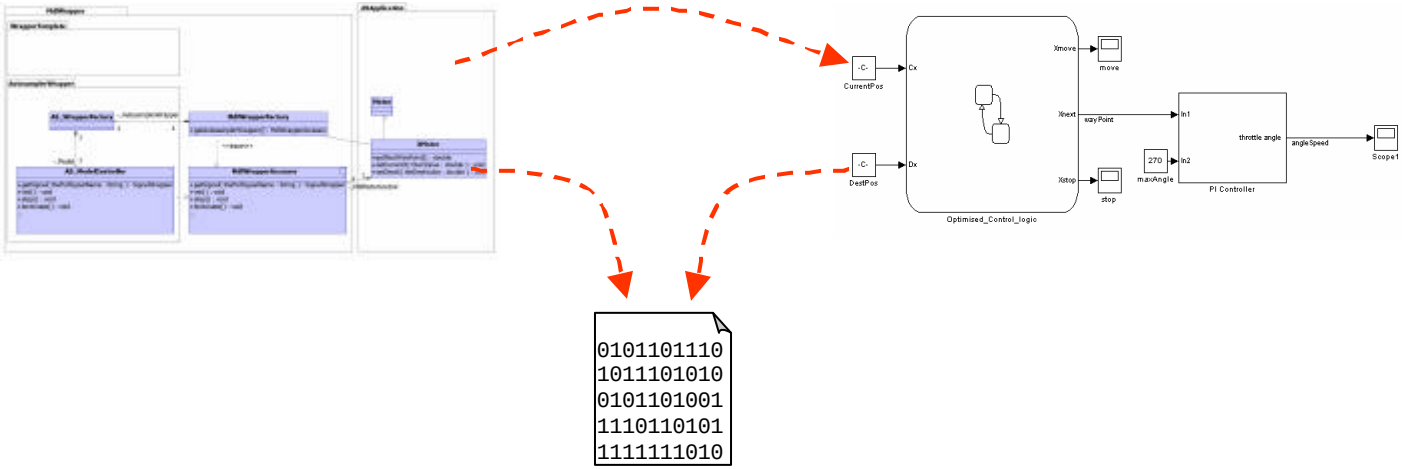
Automatisierte Modellkopplung heterogener eingebetteter Systeme



Clemens Reichmann, Philipp Graf, Klaus D. Müller-Glaser

Institut für Technik der Informationsverarbeitung
Universität Karlsruhe (TH)

Prof. Dr. K. D. Müller-Glaser
Prof. Dr. J. Becker

Gliederung

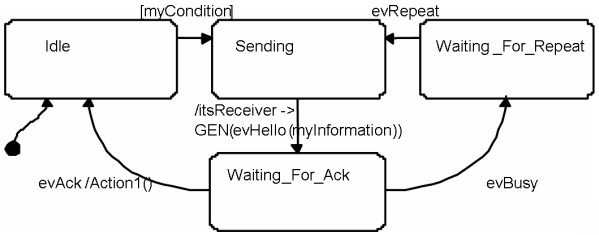
- Umfeld
  - Randbedingungen des Softwareentwurfs für eingebettete Systeme
- Grundlagen
  - Modellierungsdomänen
  - Integration durch Modelltransformation
- GeneralStore
  - Modellverwaltung
  - Werkzeugkopplung
  - Codeerzeugung
- Automatisierte Schnittstellengenerierung
  - Betrachtete Schnittstellen
  - UML Vorlagenmechanismus
  - Kopplungsmodellierung und -umsetzung
- Zusammenfassung und Ausblick

Umfeld und Herausforderungen

- Entwurf Software für eingebettete Systeme
  - Aufgaben: Messen, Steuern, Regeln
  - Software im Kontext der Hardware und der Umgebung (Aktoren, Sensoren)
  - Begrenzte Systemressourcen
  - Anwendungsbereiche:
    Automotive, Automatisierung, Medizintechnik

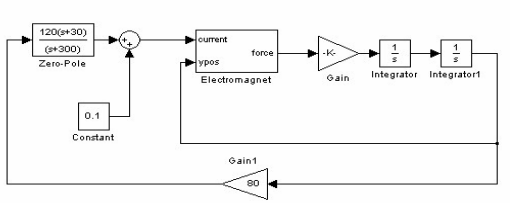
- Entwurfsrandbedingungen
  - Stark steigende Komplexität
  - Gegenläufig: Kostenbegrenzung, kurze Produktzyklen, Variantenreichtum, konstante Qualität und Sicherheit
  - Heterogenes Entwurfsumfeld, verstärkt Software-intensive Probleme
- Lösungsansatz
  - Graphische Modellierung
  - Verringerung der Entwurfskomplexität durch mächtigere Artefakte
  - Partitionierung des Entwurfs auf unterschiedliche Modellierungsdomänen

Ereignisdiskret
Modellbildung mit Statecharts



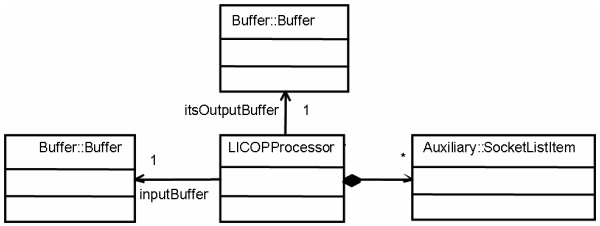
- Rhapsody in C++ (i-Logix)
- Statemate (i-Logix)
- Stateflow (The MathWorks)

Signalflussorientiert
Modellbildung mit Blockdiagrammen



- MATLAB/Simulink (The MathWorks)
- MATRIXx (National Instruments)
- ASCET-SD (ETAS)

Softwarekomponenten
Modellbildung mit UML



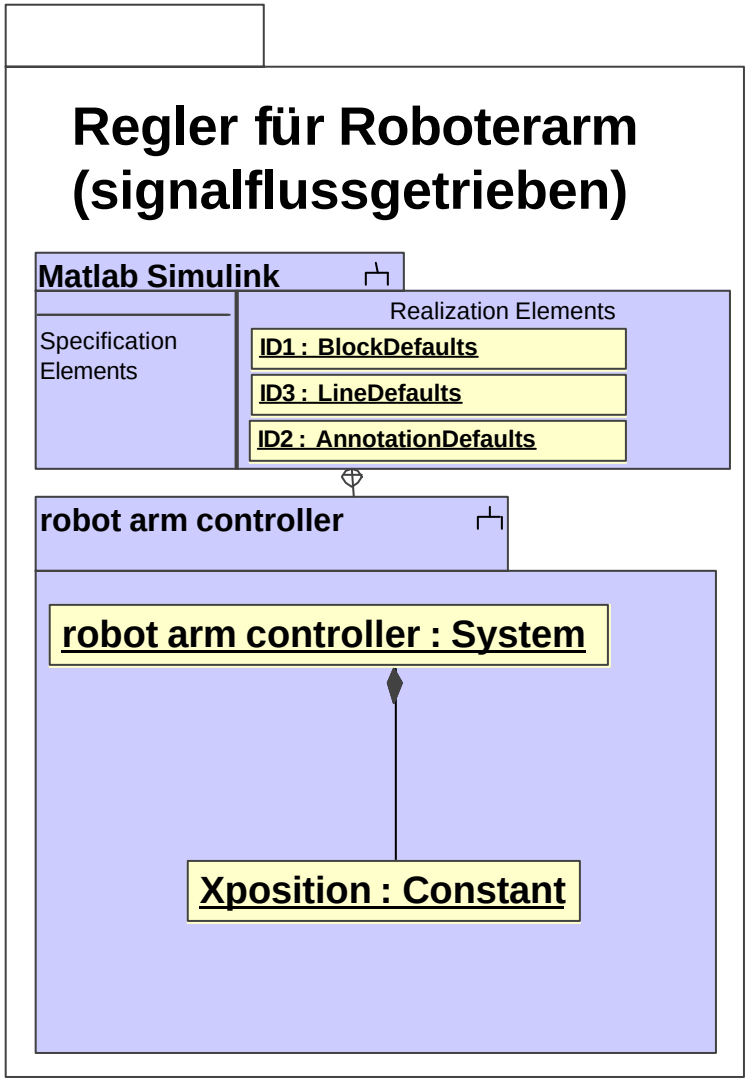
- Real-time Studio (ARTiSAN)
- Rhapsody in C++ (i-Logix)
- ObjectiF (MicroTOOL)
- Rose (Rational Software, IBM)
- Together (Borland)
- Poseidon (Gentleware)
- MagicDraw (NoMagic)
- Ameos (Aonix)
- TAU2 (Telelogic)

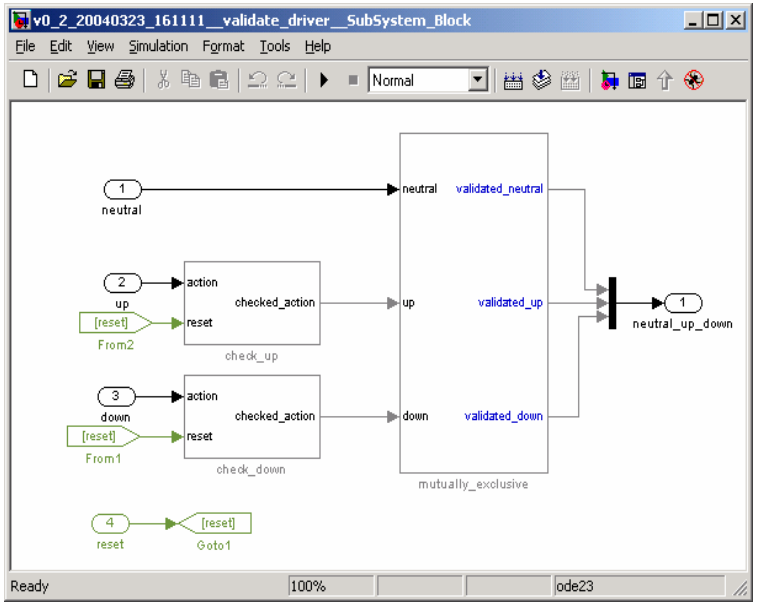
Unified Modeling Language als zentrale Notation

- **Unified Modelling Language (UML)**
  - Industriestandard der Object Management Group (OMG), Version 2.0
  - Wohldefinierte Sprache durch:
    - Abstrakte Syntax (MOF Metamodell)
    - Object Constraint Language (OCL)
    - Prosabeschreibung der Semantik

  - Trennung zwischen Modell und Ansicht
  - Beschreibung statischer und dynamischer Aspekte
  - Unabhängig von der Zielsprache (Action Language)
- **Modell des Gesamtsystems in einer Notation**
  - Modellverwaltung auf Basis des UML-Metamodells 1.5
  - Systemteile werden problemspezifisch in passender Notation beschrieben und mit bestmöglichen Werkzeugen bearbeitet
  - Bidirektionale Transformation in die Gesamtnotation

Transformation zwischen Simulink und UML





Übersicht: GeneralStore

Concurrent Engineering

Nutzerverwaltung

Datenhaltung des Gesamtmodells in Datenbank

Versionierung der Modellartefakte

Austausch von Modelldaten mit UML Werkzeugen

Modell-austausch

Transformation von und nach MATLAB/Simulink/ Stateflow

Austausch von Modelldaten mit Statemate

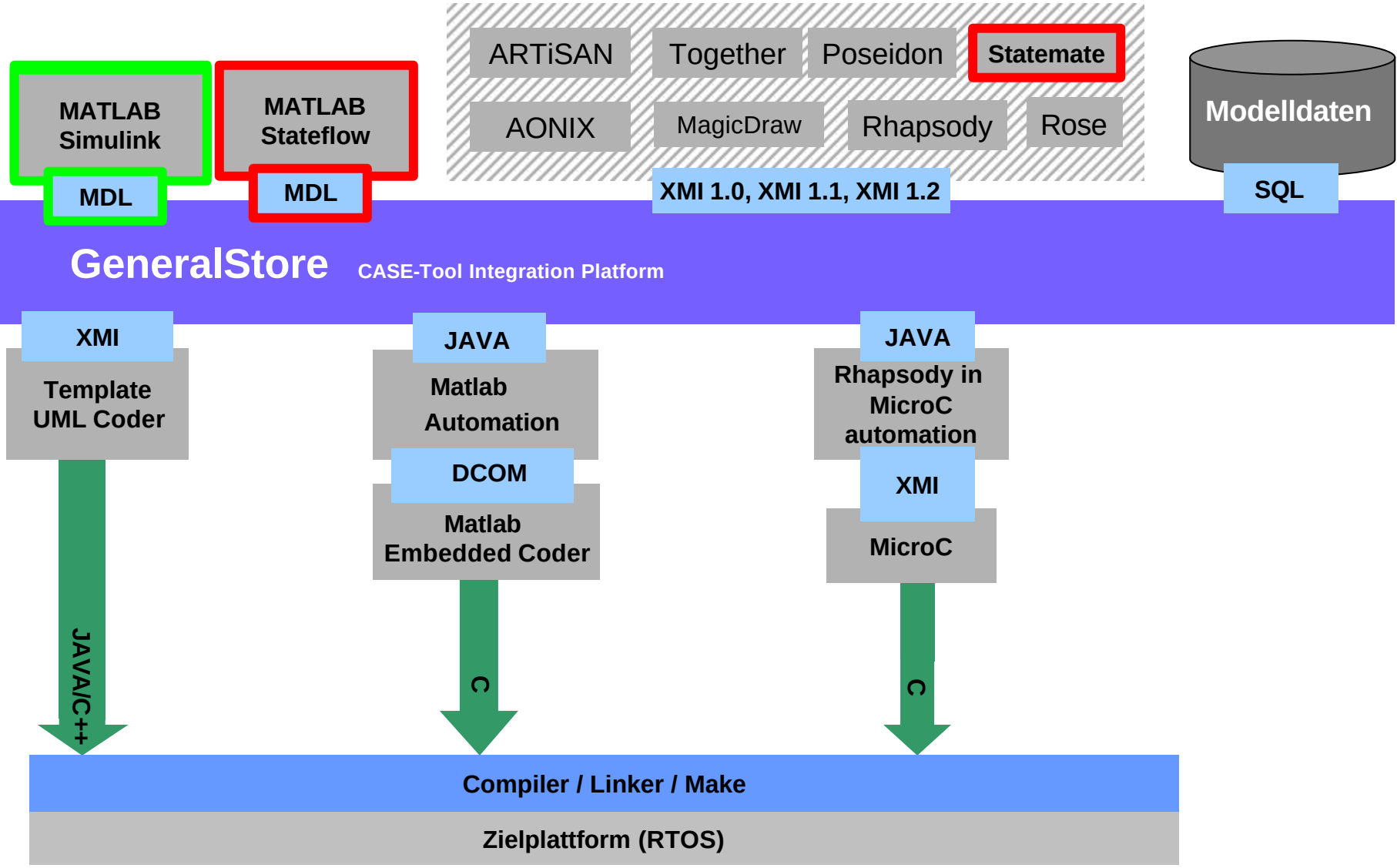
Codegenerierung

Generischer Codegenerator für JAVA/C++ aus UML

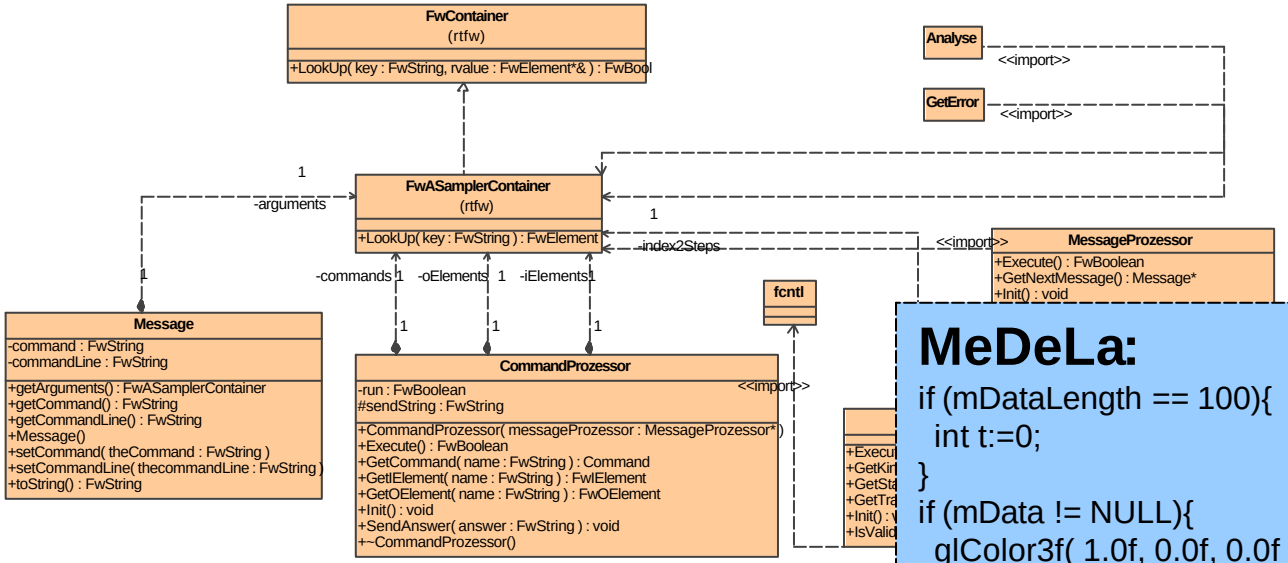
Anstossen von Code-generatoren (z.B. Embedded Coder)

Automatisierte Erzeugung von Codeschnittstellen

GeneralStore - Architektur und Codeerzeugung



Codeerzeugung

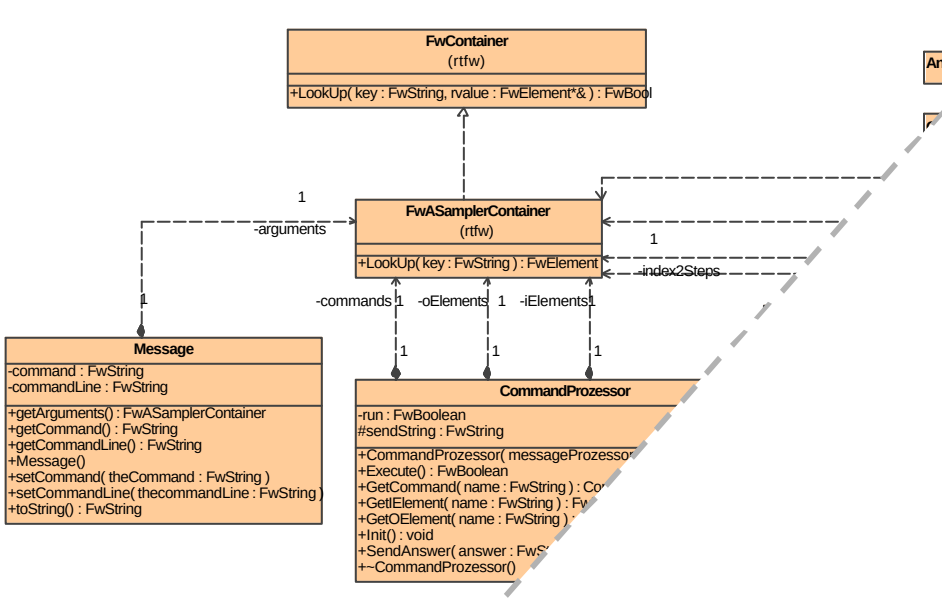


- **Code erzeugbar aus Klassendiagrammen**
- **Verhaltensbeschreibung mit MeDeLa**
- **Derzeit in Entwicklung:**
  - Sequenzdiagramme
  - Statecharts

MeDeLa:

```
if (mDataLength == 100){
  int t:=0;
}
if (mData != NULL){
  glColor3f( 1.0f, 0.0f, 0.0f );
  double prevX := mMin.getX();
  double prevY := mMin.getY();
  double step :=
(abs(mMin.getX())+abs(mMax.getX()))/mDataLength;
  int count := 0;
  for (double x := mMin.getX(); x < mMax.getX(); x +=
step){
    double y := mData[count];
    drawLine(prevX,prevY,x,y);
    prevX := x;
    prevY := y;
    count++;
  }
}
```

Codeerzeugung



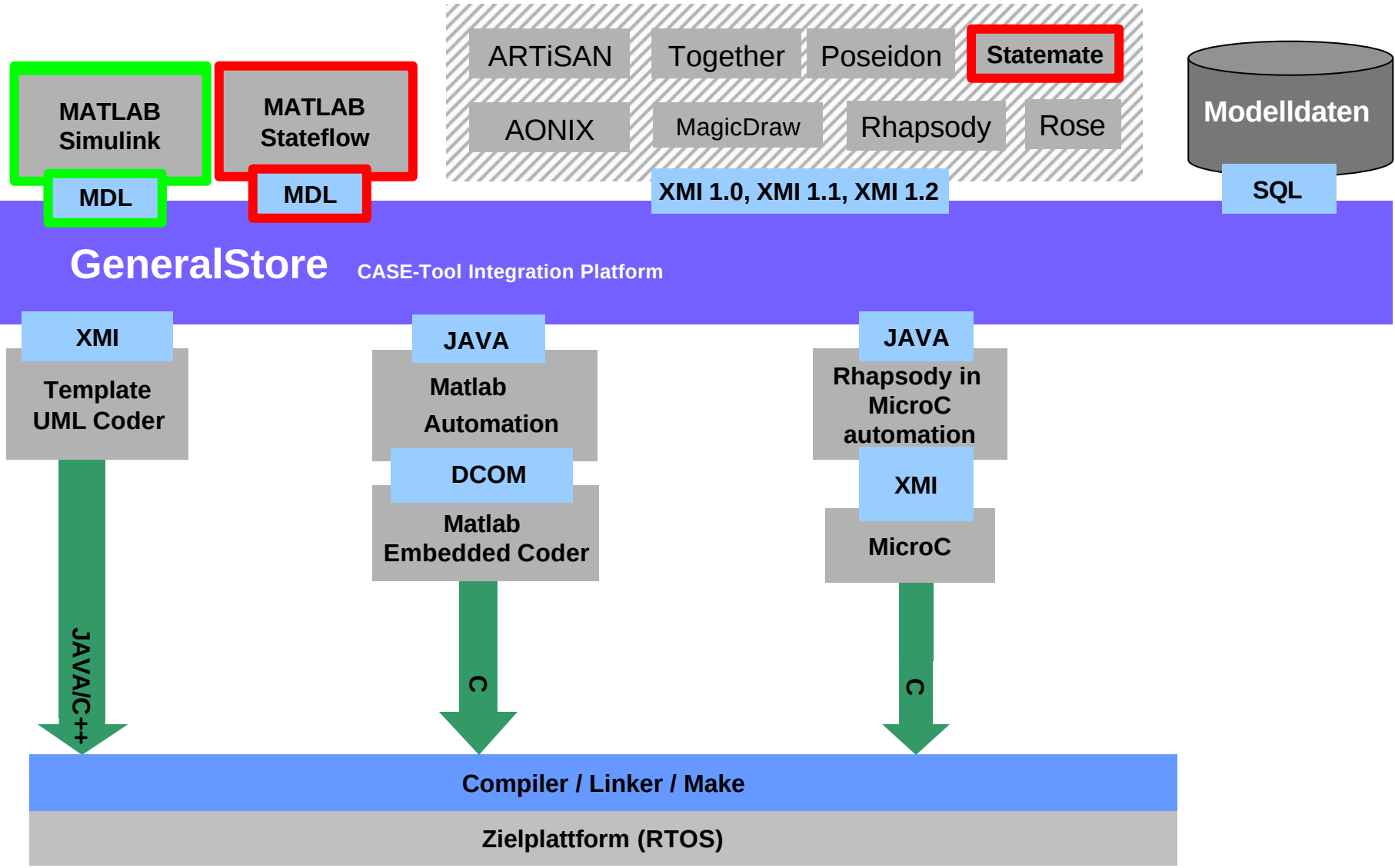
Velocity Template:

```
## ---------- Class, Parents, and Interfaces ----------
// class declaration
class $class.name##
#if($class.parents.size()+$class.interfaces.size() > 0)##
                    :
                    #foreach($parent in $class.parents)##
                                        #if($velocityCount > 1)##
                                        ,
                                        #end##if
                                           public $parent.name##
                    #end##foreach
                    #foreach($interface in $class.interfaces)##

                    #if($class.parents.size()+$velocityCount > 1)##
                                        ,
                                        #end##if
                                           public $interface.name##
                    #end##foreach
#end##if

// associations
```

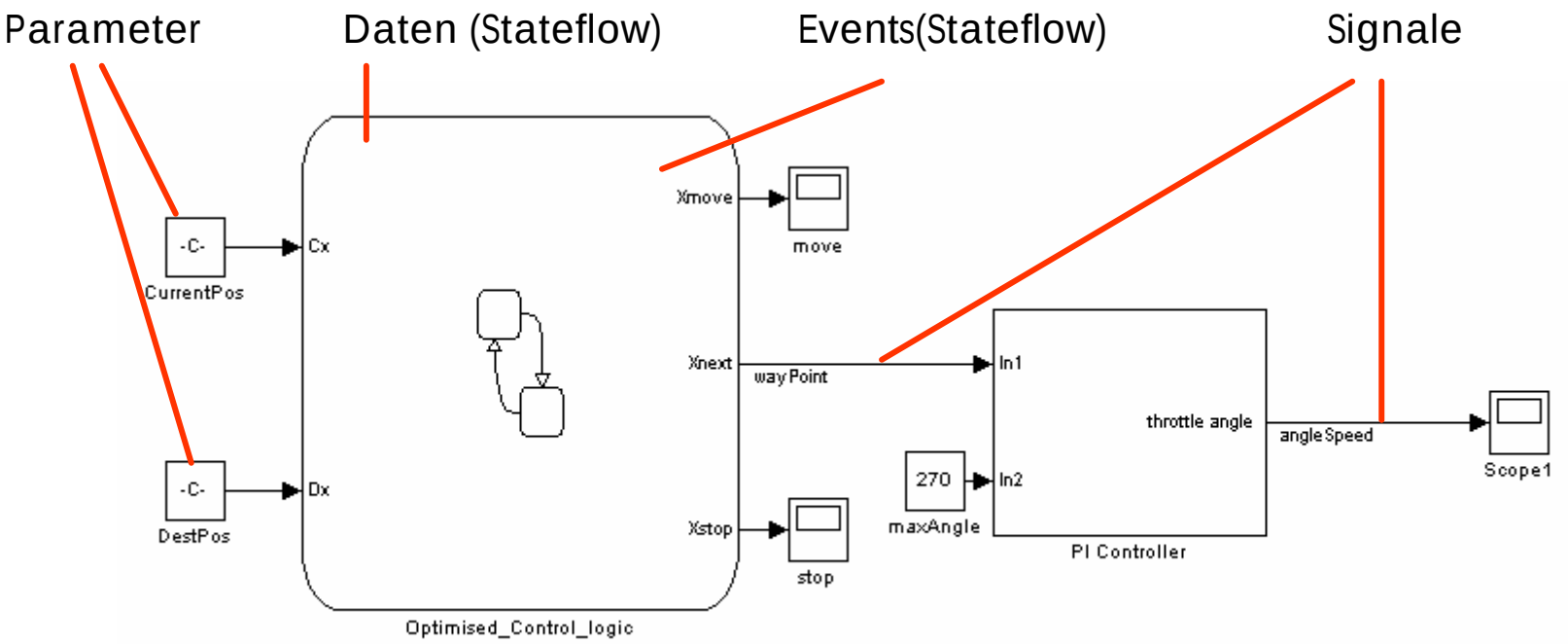
JAVA/C++ Quellcode:

```
// class declaration
class FwAsamplerConainer :
    public FwContainer {

// associations
```

GeneralStore - Architektur und Codeerzeugung



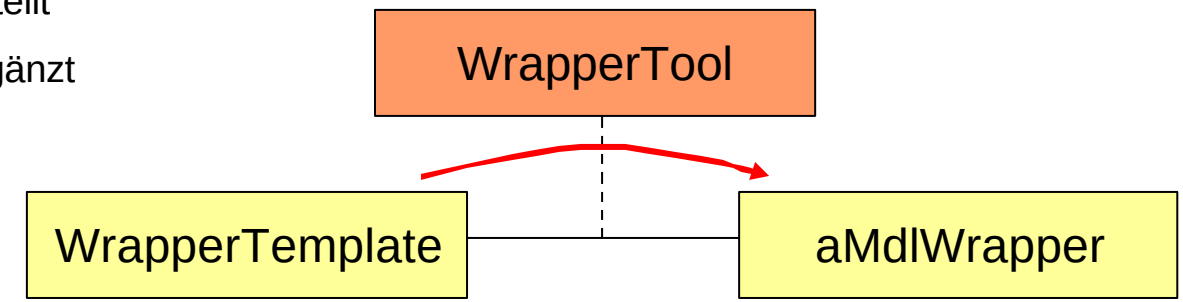
MATLAB Simulink / Stateflow Schnittstellen



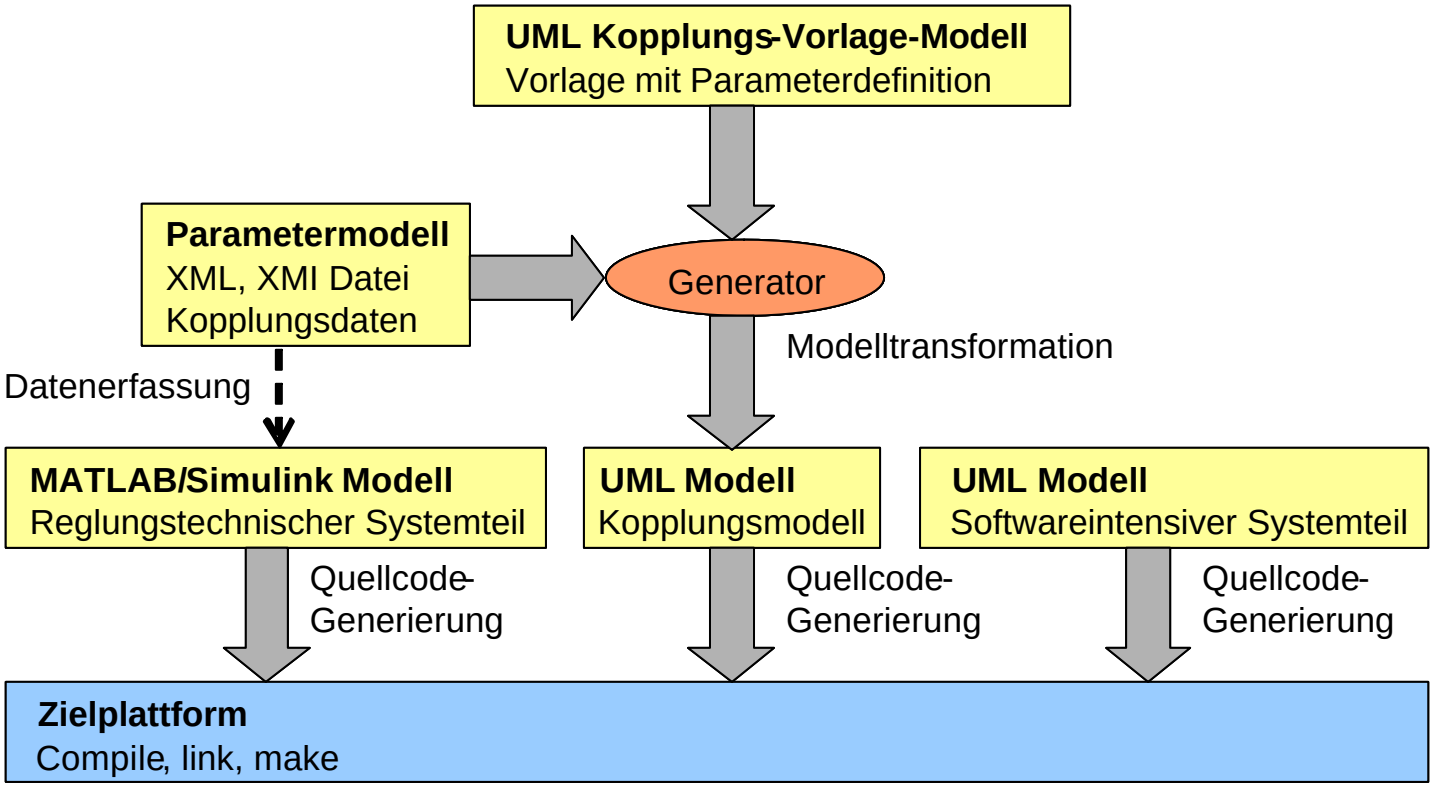
- Benötigte Schnittstellen müssen im Matlabmodell als solche gekennzeichnet werden
- Schnittstellen werden im generierten Code als globale Variablen, Arrays oder Funktionen definiert
- Sie können über `extern "C"` statements in eigene Applikationen integriert werden; spätestens beim Binden muss die Definition verfügbar sein

Wrapper Konzept

- Parametrisierbares UML Modell beschreibt Wrapper (WrapperTemplate)
  - Gibt Struktur und Umfang des MdlWrappers vor
  - Parametrisierung durch Template-Klassen
  - Methoden mit MeDeLa beschrieben
  - Kann durch ein UML-CASE-Tool vom Entwickler angepasst und verändert werden
- WrapperTool generiert den Wrapper
  - Das Matlab Modell wird nach benötigten Informationen untersucht
  - Das WrapperTemplate-Modell wird gelesen
  - → Aus diesen Informationen wird der MdlWrapper erstellt
    - Struktur wird erstellt
    - MeDeLa wird ergänzt

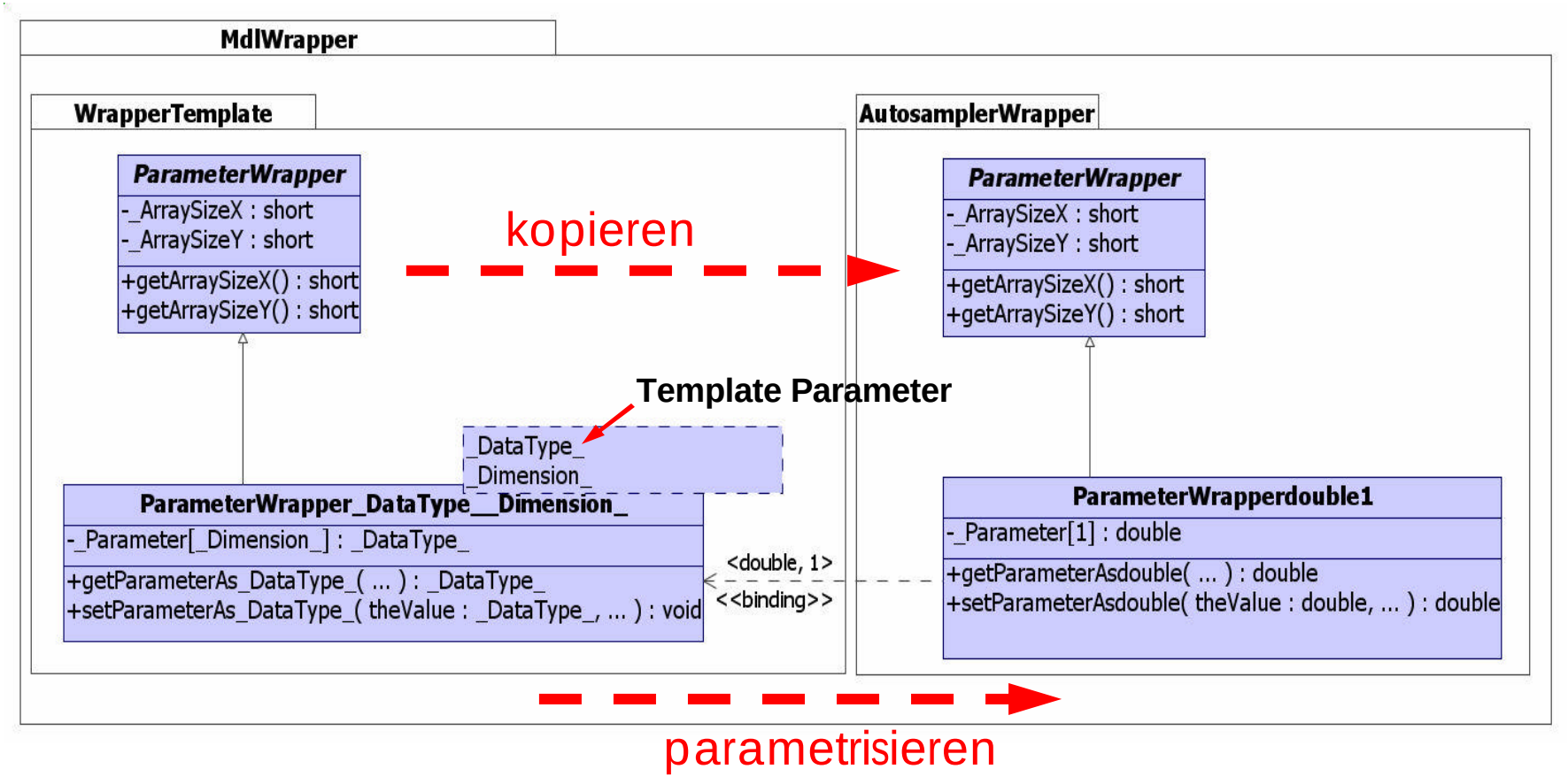


Workflow bei der Kopplung



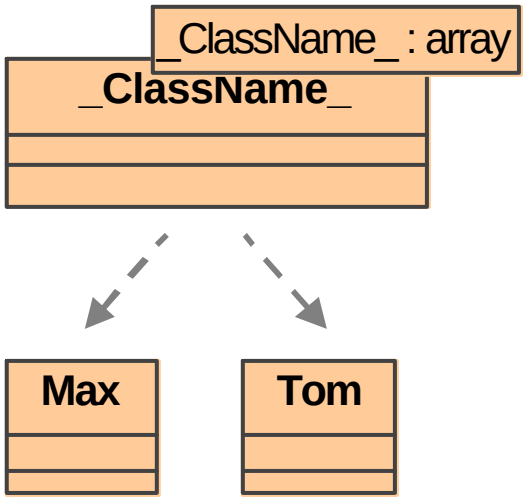
- Komplett modellbasiert
  - Vorlage des Kopplungsmechanismus („WrapperTemplate“) ist ein UML Modell
  - Zur Generierung der Kopplung werden nur Modellelemente benötigt
  - Ergebnis der Generierung ist ein UML Modell

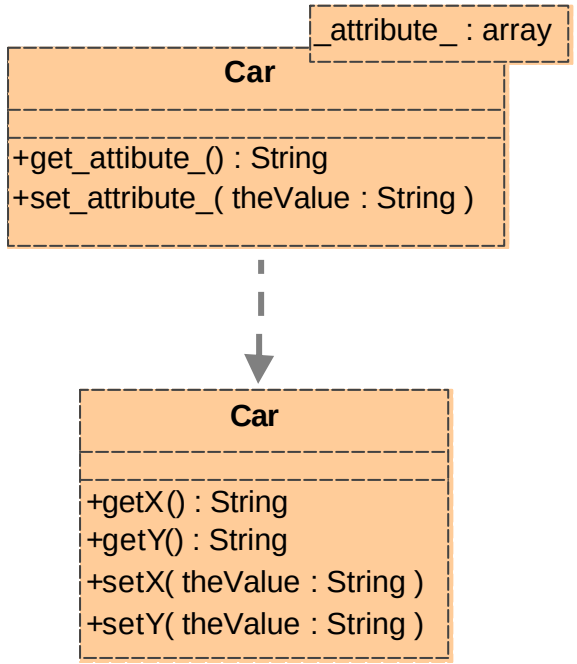
UML Template Mechanismus



UML Template Mechanismus - Konventionen

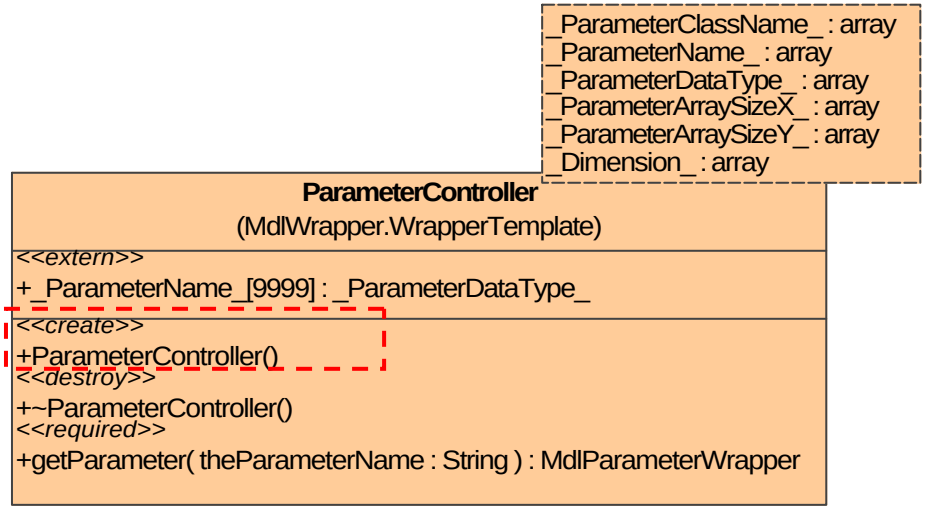
- Erweiterung des Templatemechanismus
  - Typ „array“ mit Dimension n
    - Für Klassennamen: ergibt n Klassen
    - Für Attribute/Operationen: ergibt n Attribute/Operationen





UML Template Mechanismus - Konventionen

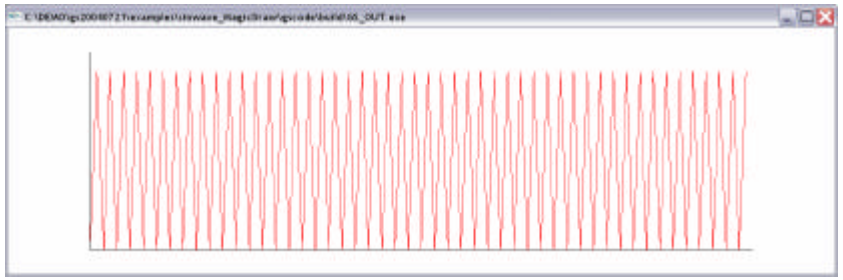
- Erweiterung des Templatemechanismus (Fortsetzung)
  - MeDeLa enthält spezielle Befehle, welche bei Parametrisierung aufgelöst werden:
    - Iterationen
    - Bedingungen



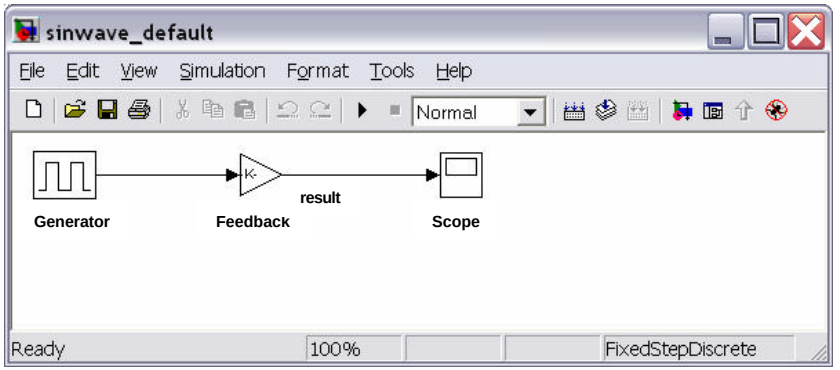

```
_parameter := new Vector();
MdlParameterWrapper aParam := null;

//#{
aParam := new _ParameterClassName_(
                                    _ParameterName_,
                                    String("_ParameterName_"),
                                    _ParameterArraySizeX_,
                                    _ParameterArraySizeY_
                                );
_parameter.add(aParam);
//#}
```

Anwendungsbeispiel

- **Softwarekomponenten**
  Modellbildung mit UML

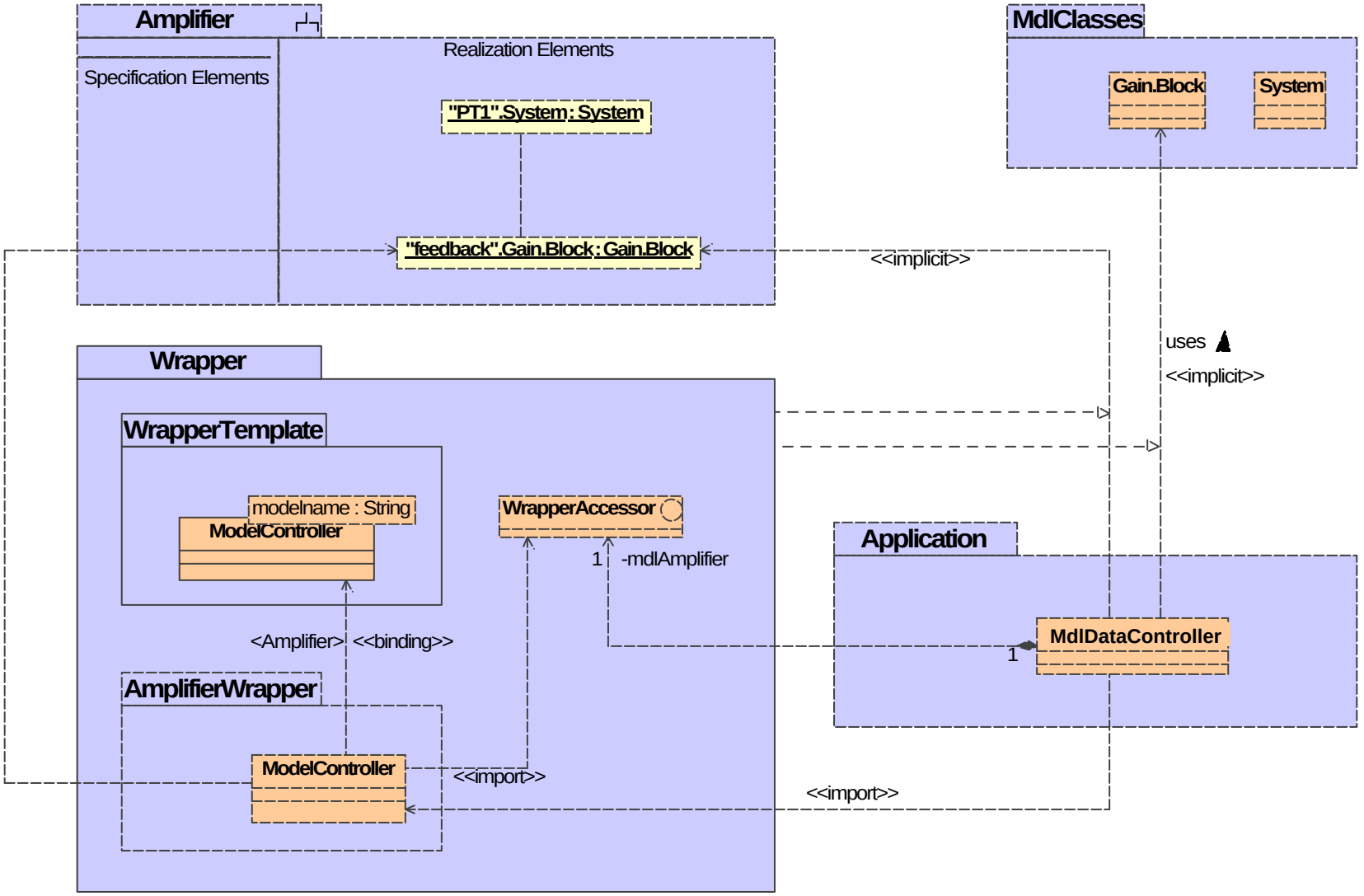


- **Signalflussorientiert**
  Modellbildung mit Blockdiagrammen

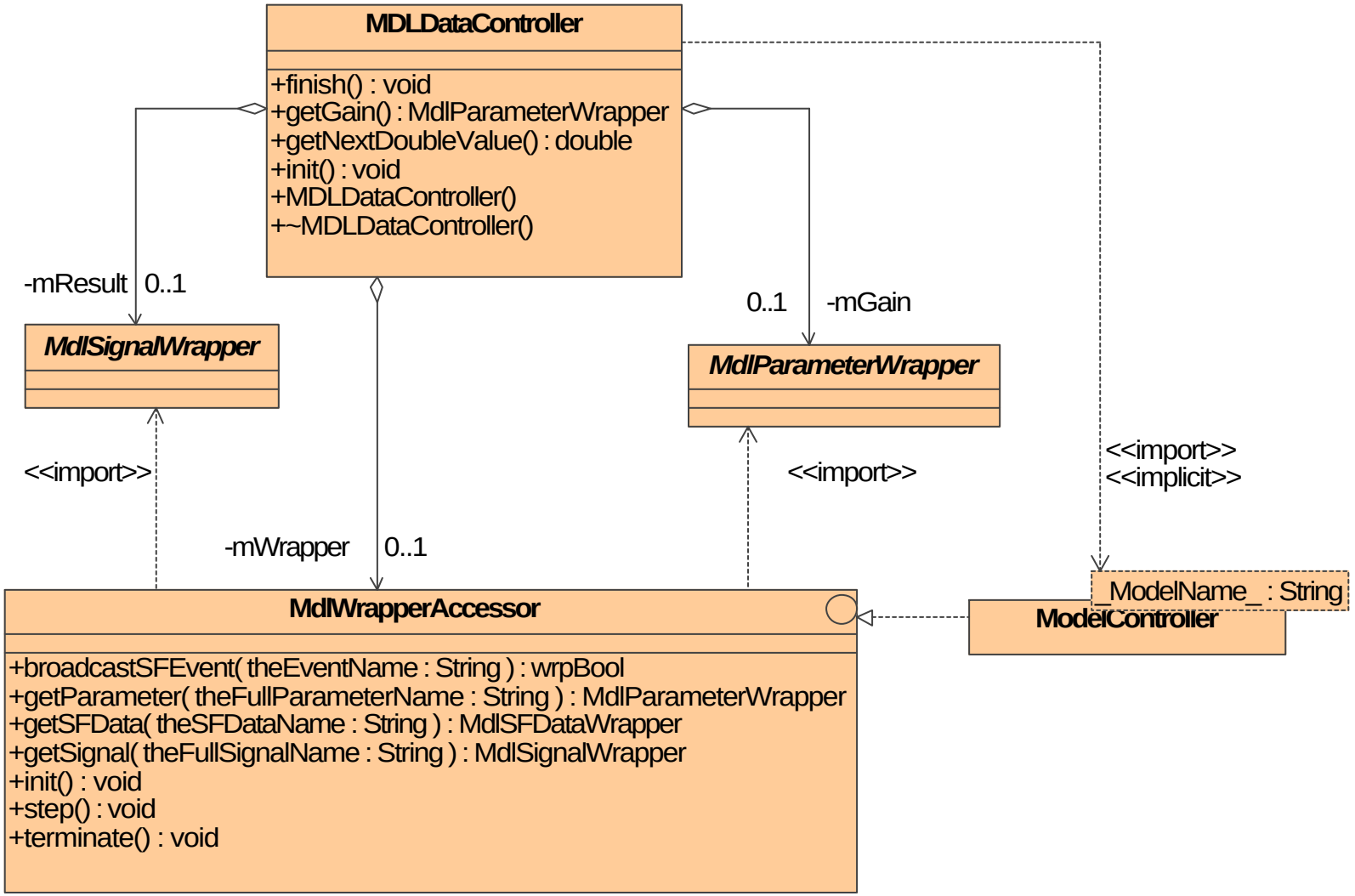


- Signal wird ausgelesen und Parameter kann verändert werden
- UML Modell
  - Kommuniziert mit MATLAB/Simulink
    - Scheduling des MATLAB Modells
    - Verstärkerfaktor zur Laufzeit verändern
    - Auslesen des Signals
  - Initialisierung/Herunterfahren
  - Nutzerinterface (Eingabe, OpenGL)

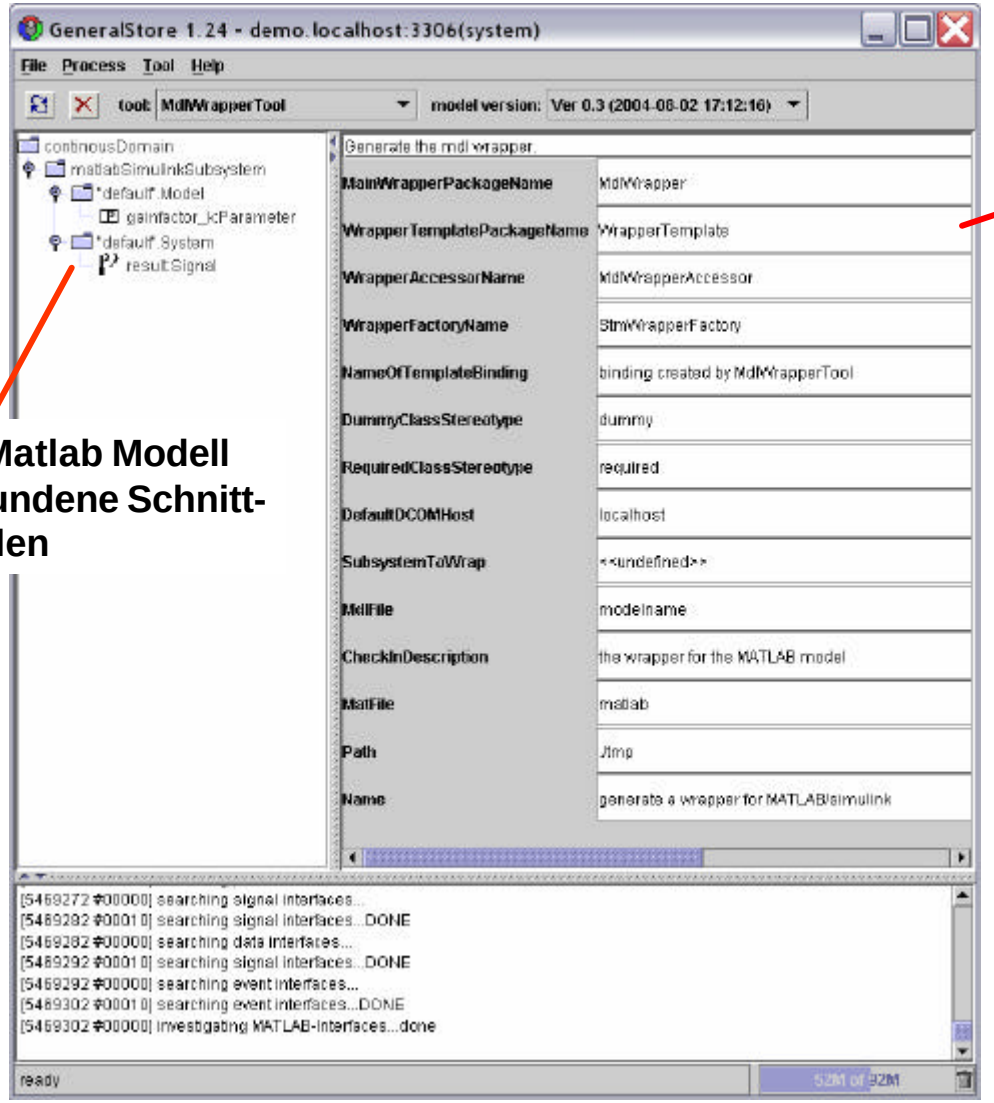
Architektur des Kopplungsmodells



Wrapper Anbindung



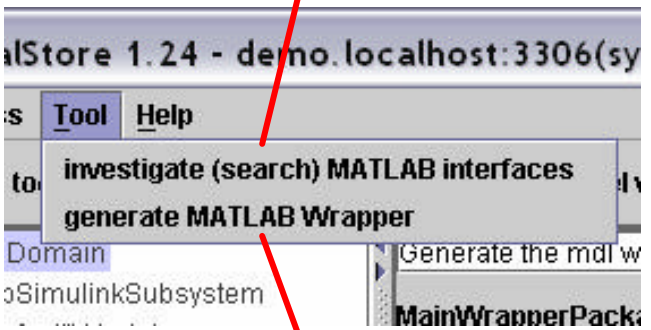
Implementierung als Plug-In von GeneralStore



Im Matlab Modell gefundene Schnitt-stellen

Zur Wrapper-Generierung benötigte Parameter

Schnittstellen im MATLAB-Modell suchen



Templatemodell in Kopplungs-modell transformieren

Zusammenfassung

- Heterogene Modellierung in unterschiedlichen Notationen nötig zur adäquaten Behandlung des Entwurfs eingebetteter Systeme
- Integrationsplattform GeneralStore
  - Modellverwaltung auf Basis des UML 1.5 Metamodells
  - Transformation aus anderen Notationen
  - Erzeugung eines lauffähigen Systems durch Codeerzeugung
- Werkzeugkopplung
  - Kopplung auf Modellebene
  - Modellierung der Kopplungsvorlage mit UML Templates
  - Automatische Umsetzung in eine UML-Kopplungsmodell, aus dem Code erzeugt werden kann
  - Implementiert in GeneralStore

Ausblick

- GeneralStore
  - Erweiterung der Codegenerierung auf C
  - Codegenerierung aus Sequenzdiagrammen und Statecharts
  - ETAS ASCET-SD Anbindung
  - UML 2.0 als zentrales Metamodell
  - Skalierung der Datenbank bei großen Modellen

- **http://gs.itiv.uni-karlsruhe.de**