Vorlesungsreihe „Diskrete Simulation“ (Masterkurs)

Das .NET-basierte diskrete Simulationssystem SpeedSim

Prof. Dr.-Ing. Thomas Wiedemann

email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Überblick

- **Bewertung aktueller diskreter Simulatorkonzepte**
  - Probleme und neue Anforderungen
  - mögliche Lösungsoptionen

- **Das Simulationssystem SpeedSim Version 4**

  - Historie und Ausgangsbasis bei der Entwicklung
  - grundlegende Anwendungsoptionen als Standalone-Version und gesamtheitliches Simulationssystem
  - Die Objekt-Hierarchie des Simulators und der Simulationskomponenten
  - Codebeispiele zur Anpassung der Simulation und der Simulationskomponenten

Alle Angaben und Abbildungen zum System Speedsim stammen aus Beschreibungen und Quellen der Fa. Dualis IT-Solutions Dresden.

Vergleich der aktuell verfügbaren diskreten Simulatorkonzepte

Rein sprachbasierte Werkzeuge

Bsp.: SLX – Compiler

Vorteile:

- volle Freiheit bei der Beschreibung von Simulationsaufgaben,
- kaum Einschränkungen bei der Modellierung der Systemobjekte und deren Interaktionen
- noch relativ günstige Kosten (im Vergleich zu Bausteinsystemen)

Nachteile:

- Mod. aufwändiger und fehleranfälliger
- Ergebnisauswertung meist nur textuell oder über externe Programme
- Animation nur extern realisierbar

Baustein - basierte Werkzeuge

Bsp.: Enterprise Dynamics

Vorteile:

- sehr schnelle und relativ komfortable Modellierung auch sehr komplexer Systeme, falls die Werkzeug-Bausteine auf die Systemobjekte passen
- sehr gute Ergebnisauswertung und Animation ohne expl. Programmierung

Nachteile:

- Probleme mit sehr spezifischen Skriptsprachen und vom Baustein-verhalten abweichenden System-objekten
- meist relativ teuer (durch sehr große integrierte Funktionalität)

Neue Anforderungen an diskrete Simulationssysteme

Modellierungsanforderungen : Durchgängigkeit und Offenheit

- Nach dem allgemeinen Erfolg der OO-Programmierung sollte diese auch im Simulationsbereich DURCHGÄNGIG von der Basisprogrammierung bis hin zur Anwendungsanpassung eingesetzt werden.
- Verfügbare und bei den Programmierern gut bekannte Entwicklungsumgebungen sollten durchgängig eingesetzt werden.

Integration in modernste IT-Architekturen:

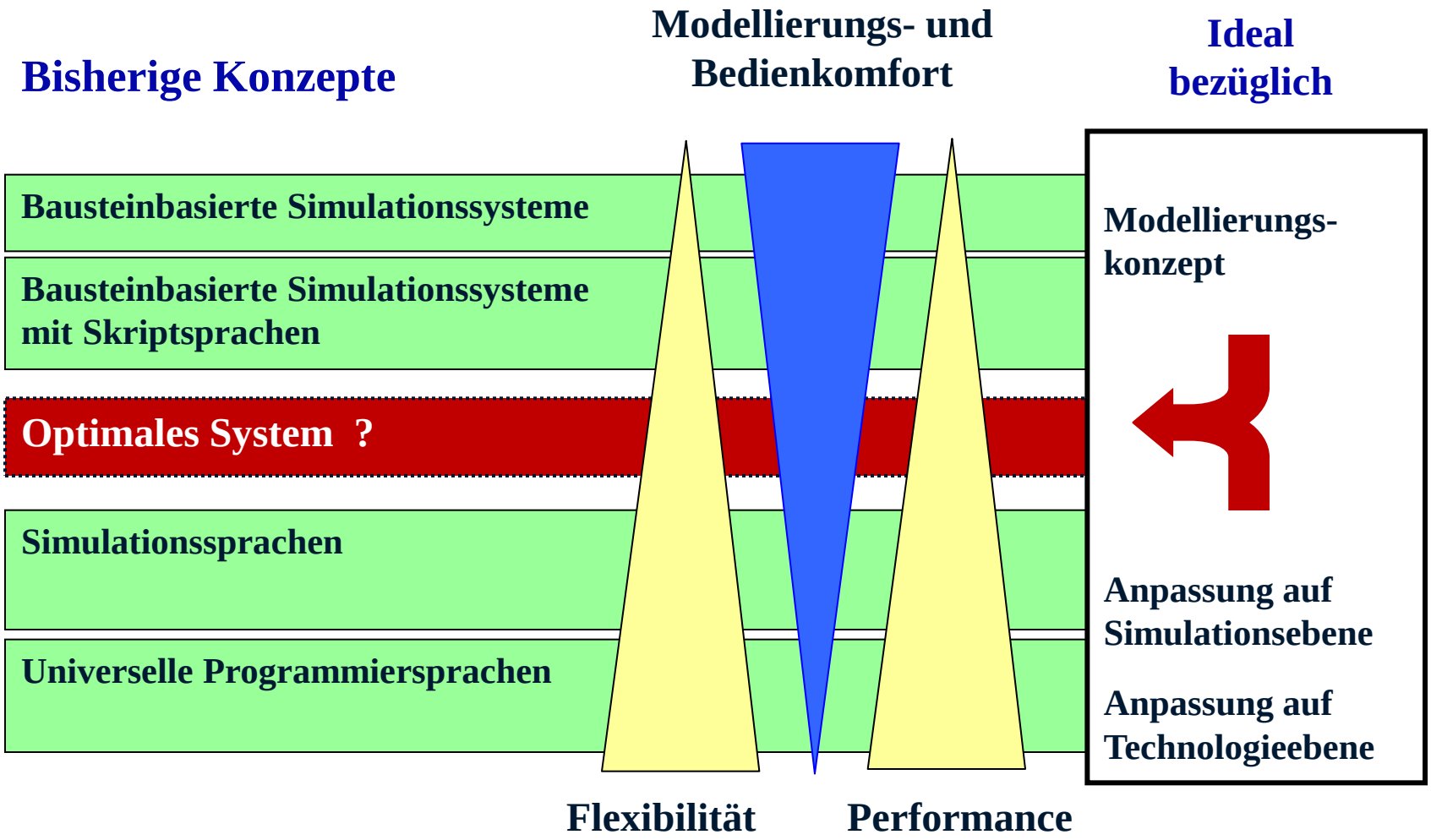
- Neue Integrationskonzepte im IT-Bereich wie SOA und SaaS fordern auch von Simulationssystemen eine Anbindung und Verwendung modernster Technologien, wie z.B. Web-Services oder andere Web-basierte Schnittstellen

Performanceanforderungen

- speziell bei sehr großen Modellen ist die Simulationsgeschwindigkeit von Bedeutung um Simulationsergebnisse effizient verwenden zu können
- Bei einer Optimierung auf der Basis von Simulationsmodellen verschärfen sich die Anforderungen aufgrund der tw. 100fachen Berechnung noch weiter !!

Neue Entwicklungsoptionen für die diskrete Simulation

Sinnvoll erscheint eine Kombination aus beiden Werkzeugklassen unter Wahrung der jeweiligen Vorteile :



Mögliche Lösungsoptionen

Aus der Sicht der Programmiersprache und Entwicklungsumgebung:

- **Option 1: .NET-basierte Sprachen wie C#**
  - + gute Anbindung an Microsoft-Technologien (auch Office-Programme wie Excel – stark eingesetzt im Industriebereich)
  - sehr gute Abdeckung aller modernen IT-Technologien (DB/Web/Grafik)
  - trotz Zwischencode (MSIL) noch relativ schnell
  - Visual-Studio sehr leistungsfähig und auch gut erweiterbar
  - Nachteil: keine portable Lösung für Nicht-Windows-Systeme
- **Option 2: Java-basierte Frameworks**
  - Hauptvorteil: portabel auf (fast) allen Betriebssystemen
  - ebenfalls gute Abdeckung aller modernen IT-Technologien
  - Nachteile bei Performance und direktem Zugriff auf Rechnerressourcen
  - Entwicklungsumgebungen wie Eclipse ebenfalls sehr leistungsfähig

Fazit: prinzipiell kommen beide (oder noch weitere Optionen wie Python-basierte Sprachen) in Betracht – die genauere Darstellung soll daher an einem Beispiel aus dem .NET-Bereich erfolgen

Das .NET-basierte Simulationssystem SpeedSIM

Ausgangsbasis / Historie :

- Entwickler / Hersteller ist die Firma Dualis IT Solutions aus Dresden
- erste Versionen (V 1.0 bis 3.x) des Simulationssystems SpeedSim waren auf der Basis einer C++-Bibliothek aufgebaut:
  - alle Änderungen und Anpassungen mussten im Sourcecode erfolgen.
  - auch die Steuerung der Simulation und deren Auswertung mussten manuell kodiert werden.
  - Die zugrundliegende Basisalgorithmen waren stark an konkrete Industrieprojekte ausgerichtet und tw. schwer erweiterbar/adaptierbar.

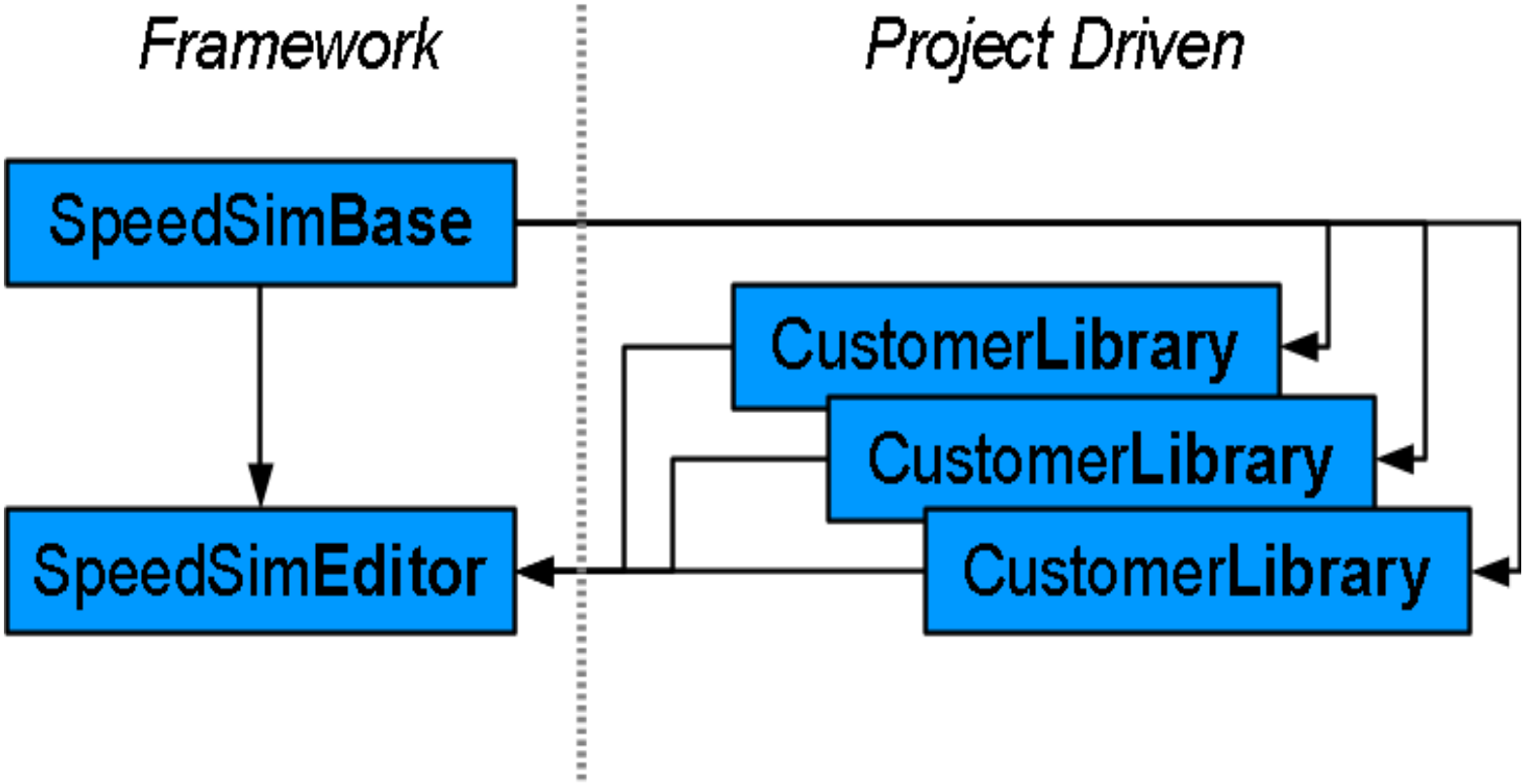
Eine Neuentwicklung auf der Basis von .NET sollte folgende Vorteile erreichen:

- komplette Überarbeitung des Simulations-Objektmodells
- maximale Ausnutzung der .NET-Funktionalität unter dem Fokus von
  - **Performance / Schlankheit / Modularität / Flexibilität**
- Integration bzw. Anbindung der Simulation in die VisualStudio-Umgebung

SpeedSIM –Basisarchitektur

SpeedSim – Frameworkarchitektur

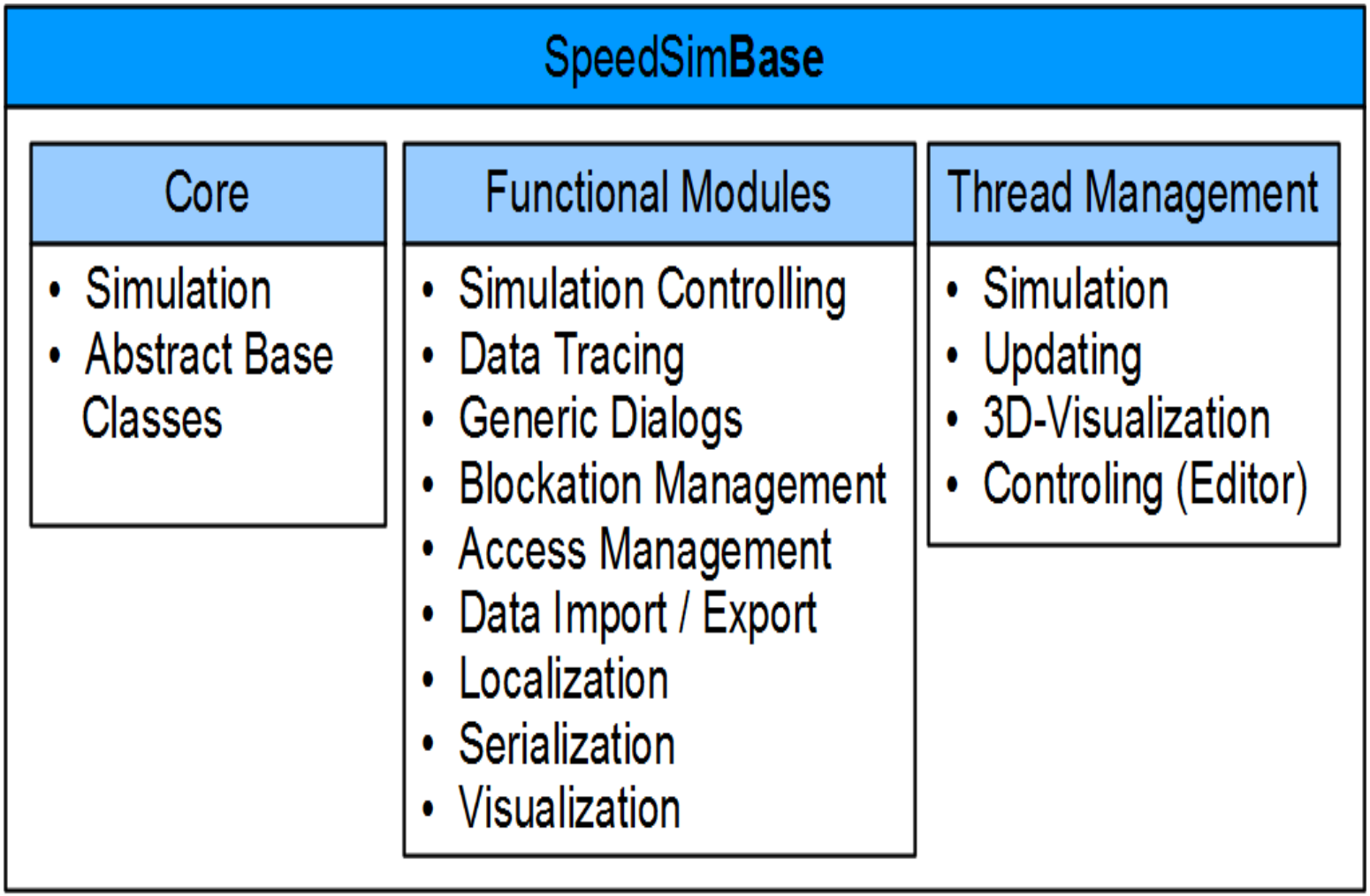
- Das Basis-Framework deckt die grundlegenden Basisfunktionen zur Modellierung und Simulation ab.
- Zusätzlich können spezielle kundenspezifische Bibliotheken angebunden bzw. integriert werden.



Die SpeedSIM –Base-Bibliothek

SpeedSimBase

- deckt die grundlegenden Simulatorfunktionen ab (vgl. Grafik).



Die SpeedSIMBase-Core-Funktionen

Aufgaben / Funktionen der SimBaseCore-Funktionen:

- **Simulationsorganisation**
  - Handling der Simulation-Events (Realisierung des Ereigniskalenders)
  - Controlling-Schnittstelle zur Steuerung der Simulation
  - Simulations-Objekt-Verwaltung (Liste statischer und dynamischer Objekte)
  - Continuous Simulation Mechanism (für die Nachbildung kontinuierlicher oder quasi-kontinuierlicher Abläufe oder auch zur Realisierung zeit-proportionaler Animationen)

- **Simulation Object Base – Classes**
  - Abstract Base Classes,
    - definieren die grundlegenden Objektattribute und Methoden aller Simulationsobjekte, welche dann später überschrieben werden (müssen)
  - **Specialized Base Classes**
    - dienen als erste Ebene für konkrete Simulationsobjekte

Die Aufgaben der SpeedSIMBase-Functional Modules

- **Simulation Controlling:**
  - Start, Stop, Pause, Reset, …
  - Zugriff auf Meta-Simulations-Daten wie Simulations-Zustand (Running, Paused, Terminated..), Simulations-Zeit, ..
- **Data Tracing**:
  - autom. Data-Tracing markierter Properties (Speichern von Wertverläufen)
  - Default: Deaktiviert (zugunsten Performance)
- **Generic Dialogs**:
  - Generalisierter Zugriff von einer GUI auf Simulations-Objekt-Eigenschaften (Properties)
  - Keine individuellen Dialoge nötig (aber möglich)
- **Blockation Management:**
  - Modellierung von blockierenden Simulations-Objekt-Logiken
  - Abbildung der Objekt-Interlogik (Objekt wartet auf 1-n Ressourcen (andere Objekte) und konkurriert mit 0-m anderen Objekten)

Die Aufgaben der SpeedSIMBase-Functional Modules (II)

- **Access Management:**
  - Benutzer-Zugriff auf Objekt-Daten (lesend, schreibend)
  - Festlegung, welche Änderungen an die Oberfläche propagiert werden
  - Erfolgt in Property-Settern, wird unterschieden in:
    - Property-Update (GUI reagiert auf Änderungen einzelner Eigenschaften)
    - Visual-Update (GUI visualisiert das Objekt neu (2D, 3D))
    - Default: keine (zugunsten Performance)
- **Data Import / Export:**
  - Excel-Schnittstelle
  - Layout-Erstellung / -Speicherung aus / in Excel-Dateien
- **Localization (zur Anpassung an andere Sprachen):**
  - Basierend auf externen code-unabhängigen XML-Dateien
- **Serialization:**
  - Speichern / Laden von Layouts oder Simulationen
  - Abbildung/Wiederherst. des akt. Speicherbildes durch Binary Serialization
- **Visualization:**
  - Grundmechanismen zur 2D-Darstellung (später optional auch 3D)

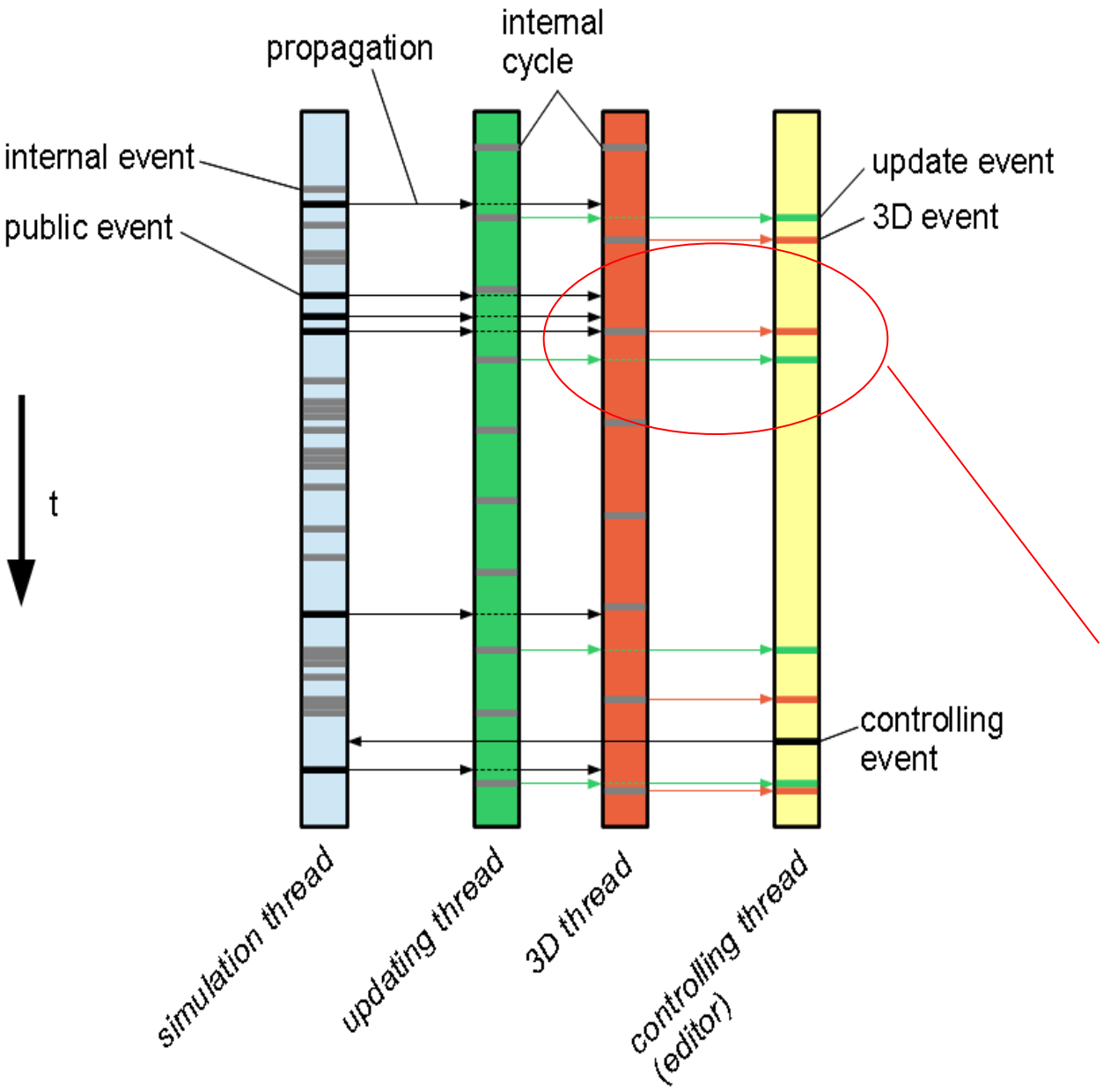
Die Aufgaben des SpeedSIMBase- Thread Management

- **Simulation Thread (Verwaltung der nebenläufigen Prozesse)**
  - Organisation der eigentlichen Simulation
  - asynchrone Abarbeitung der Simulations-Events
- **Updating Thread:**
  - Sammeln und Delegation von Änderungen an GUI, getaktet
- **3D-Visualization Thread:**
  - Sammeln und Delegation von Änderungen an 3D-Engine
  - getaktete Updates (zur Verbesserung der Gesamtperformance)
- **Controlling Thread für GUI-Interaktionen:**
  - GUI (ggf. auch speziell erstellt für Endkunden)
  - gewöhnlich mit dem SpeedSimEditor

Zeitliche Organisation des Thread-Management (siehe auch Abb. auf Folge-Folie)

- Der Simulations-Thread arbeitet unabhängig und asynchron die Events (internal und public Events) der eigentlichen Simulation ab.
- Der Simulation-Thread ist dabei meist um einige Größenordnungen schneller.

Die Abhängigkeiten im Thread Management



Zeitliche Organisation des Thread-Management

- Der Simulation-Thread ist meist um einige Größenordnungen schneller (vgl. Prakt. zur Monte-Carlo-Sim-Verh.: bis zu 1:1000) als die anderen Threads.

- GUI- und 2D/3D-Updates werden daher gesammelt und es wird ggf. nur die letzte Updateaktion ausgegeben (bei einem Default-Updateintervall von 40 ms.)

Die SpeedSIM-Objekthierarchie - ACSimObject

Abstraktes Basisobjekt ACSimObject

- **stellt Grundbaustein für alle Simulationsobjekte bereit**
- **Integration in die Komponentenverwaltung des VS-Editors**
  - GET- und SET-Methoden für Zugriff über Eigenschaftenfenster
- **Simulationsmethoden und Events (unvollst. Auszug – vgl. Codebibl.)**

```
public virtual void Action(CSimEvent SimEvent);//Callback für Sim-Ereignisse
public virtual void Delay(double DiffTime); // Zeitverzögerung
public virtual void Delay(double DiffTime, ACSimInfo SimInfo); // Verzög.
// mit Referenz auf externes SimInfo-Objekt (noch weitere Delay-Methoden)
```

- **Verweise auf**
  - zugehöriges Modell mit Zuordnung bei der Initialisierung mit

```
sim1object1 = new mySimObject(); // Anlage des Sim-Objektes
sim1object1.Initialise(model1, true, false);  // Initialisierung
```

  - optionale Liste von prozessierten Sim-Objekten

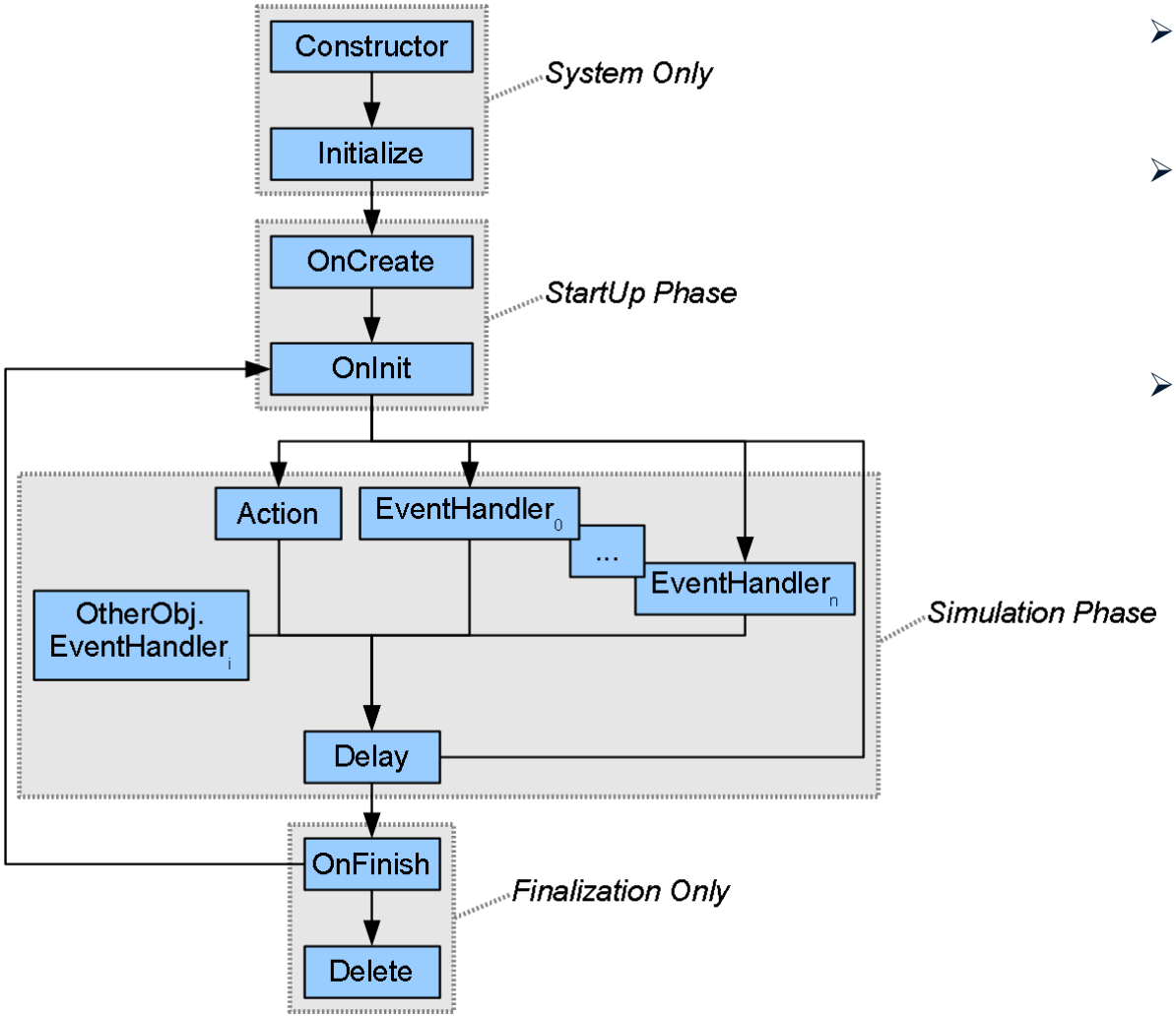
```
public List<ACSimFlowObject> ProcessedSimFlowObjects { get; }
```

  - Liste von Ein- und Ausgängen (StartingConnections/ EndingConnections)

```
public List<ACSimConnection> StartingConnections { get; set; }
```

Die SpeedSIM-Objekt-Eventhandler

Das abstrakte Basisobjekt ACSimObject definiert auch die Methoden und Eventhandler für die allg. und simulationsspezifische Steuerung verbindet diese im Simulation Objekt Life Cycle:



- Initialize und Delete sind systembedingte Methoden.
- Alle anderen Methoden sind simulationsspezifisch
- Nach dem Start der Simulation „bewegt“ sich der Simulationsprozess innerhalb des grossen Blocks in einer Schleife.

Die SpeedSIM-Objekthierarchie - ACSimObject

Abstraktes Basisobjekt ACSimObjViewable : ACSimObject

erweitert das Basisobjekt um visuelle Attribute und Methoden

- Basismethode zum Zeichnen (ggf. zu überschreiben)
  public virtual void Draw(Graphics g, double Zoom, bool IsSelected = false);
- Attribute zur Steuerung der Darstellung
  public bool Visible { get; set; }
  public virtual double Height { get; set; } // analog für width, x, y Methoden)
- Verweise auf 3D-Objektdarstelllungen
  OpenGL.DEMatrixObject ThreeDObject { get; set; }

- **Methoden zum Updaten bzw. Updatesignalisierungen**

  public void PerformVisualUpdate();

  public void NotifyVisualUpdate();

Die SpeedSIM-Objekthierarchie – CSimModel

Basisobjekt CSimModel

Definiert die Gesamtheit des Simulationsmodels mit

- **Liste der statischen und dynamischen Simulationsobjekte**

  public Dictionary<string, ACSimObject> **StaticSimObjects** { get; }

  public Dictionary<string, ACSimObject> **DynamicSimObjects** { get; }

- **Methoden zum Registrierung von Simulationsobjekten**

  public bool RegisterSimObj(ACSimObject SimObjToRegister);

  public bool RegisterSimObjNameChange(ACSimObject SimObj, string NewName);

- **Methoden zum Steuern der Simulation**

  public void InterruptSimulation();

  public void ResetSimulation();

  public void ResumeSimulation();

  public void StepSimulation();

  public void StopSimulation();

Die SpeedSIM-Objekthierarchie – CSimulation

Basisobjekt CSimulation

definiert die Simulationslaufzeitumgebung

- **aktueller Simulationsstatus und allgemeine Einstellungen**

```
public double CurrentSimTime { get; }   // aktuelle Simulationszeit
public double MaxSimTime { get; set; } // maximal Simulationszeit
public CSimulation.SimulationState CurrentSimState { get; }
```

mit dem Statuswerten

```
public enum SimulationState
    {   Running = 0,  Paused = 1,   Terminated = 2  }
```

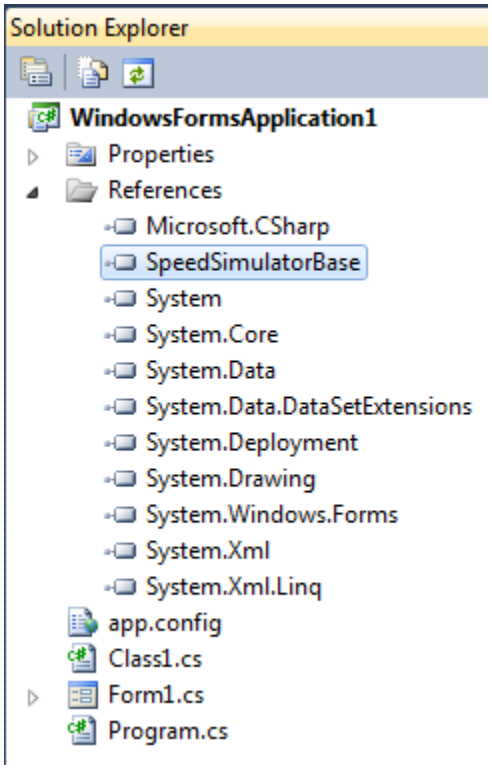
- **Methoden zum Steuern der Simulationanalog zum Modell**

```
public void InterruptSimulation();   public void ResetSimulation();
public void ResumeSimulation();     public void StepSimulation();
public void StopSimulation();
```

Die SpeedSIM-Laufzeitkonfigurationen

Option A.) Stand-alone Anwendung

- Einbindung der DLL „SpeedSimulatorBase“ als Referenz in die Windows-Applikation
- damit wird das gesamte SpeedSim-Objektmodell in die Applikation eingebunden
- Die gesamte Simulationskonfiguration und –steuerung sind dann entsprechend zu programmieren (vgl. Programmierbeispiel auf Folgeseite)



Vorteile / Anwendungsfälle

- Anwendungen mit voller Kontrolle über die Visualisierung der der Simulation
- ggf. auch Einbindung anderer Komponenten aus dem VisualStudio-Toolset

Nachteile

- prinzipbedingt höherer Aufwand bei der Steuerung

Die SpeedSIM-Laufzeitkonfigurationen – „Stand-alone“

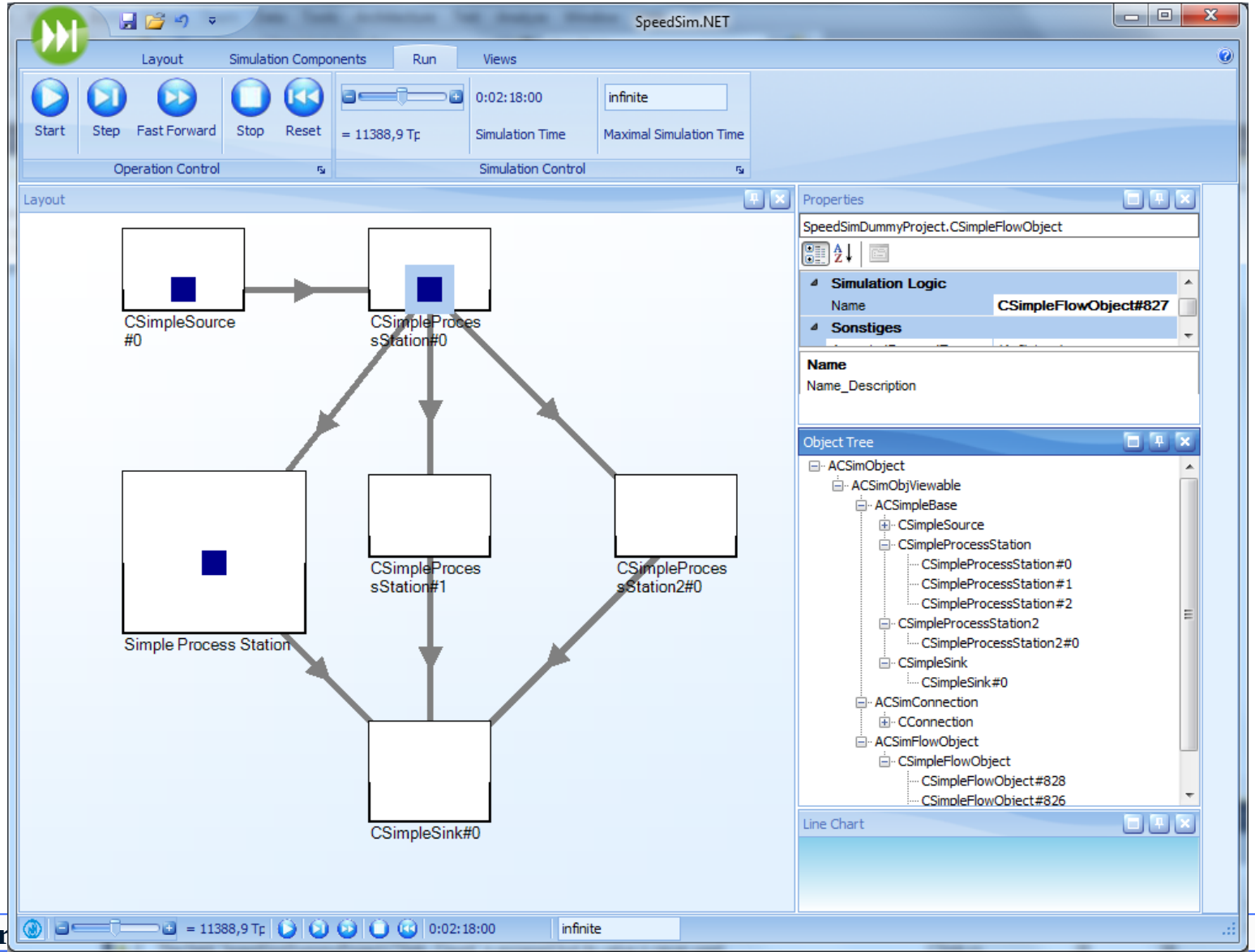
Stand-alone-Programmierbeispiel

```
public Form1()
{   InitializeComponent();  // Init Form
    // Define Model for Simulation
        model1 =  new SpeedSimulatorBase.Simulation.CSimModel();
       model1.Initialize();
        // Register SimObjects in Model
    sim1object1 = new mySimObject();
       sim1object1.Initialise(model1,true,false);
    sim1object2 = new mySimObject2();
       sim1object2.Initialise(model1,true,false);
       // Define Simulation
            simulation1 = new CSimulation();
            simulation1.Initialize(model1);
            simulation1.MaxSimTime = 100; // Define Simulation Duration
            simulation1.SecoundsPerTick = 0.01;
  }
```

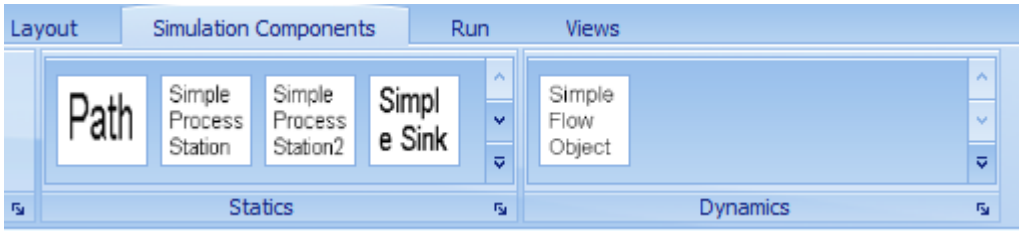
Die SpeedSIM-Laufzeitkonfigurationen

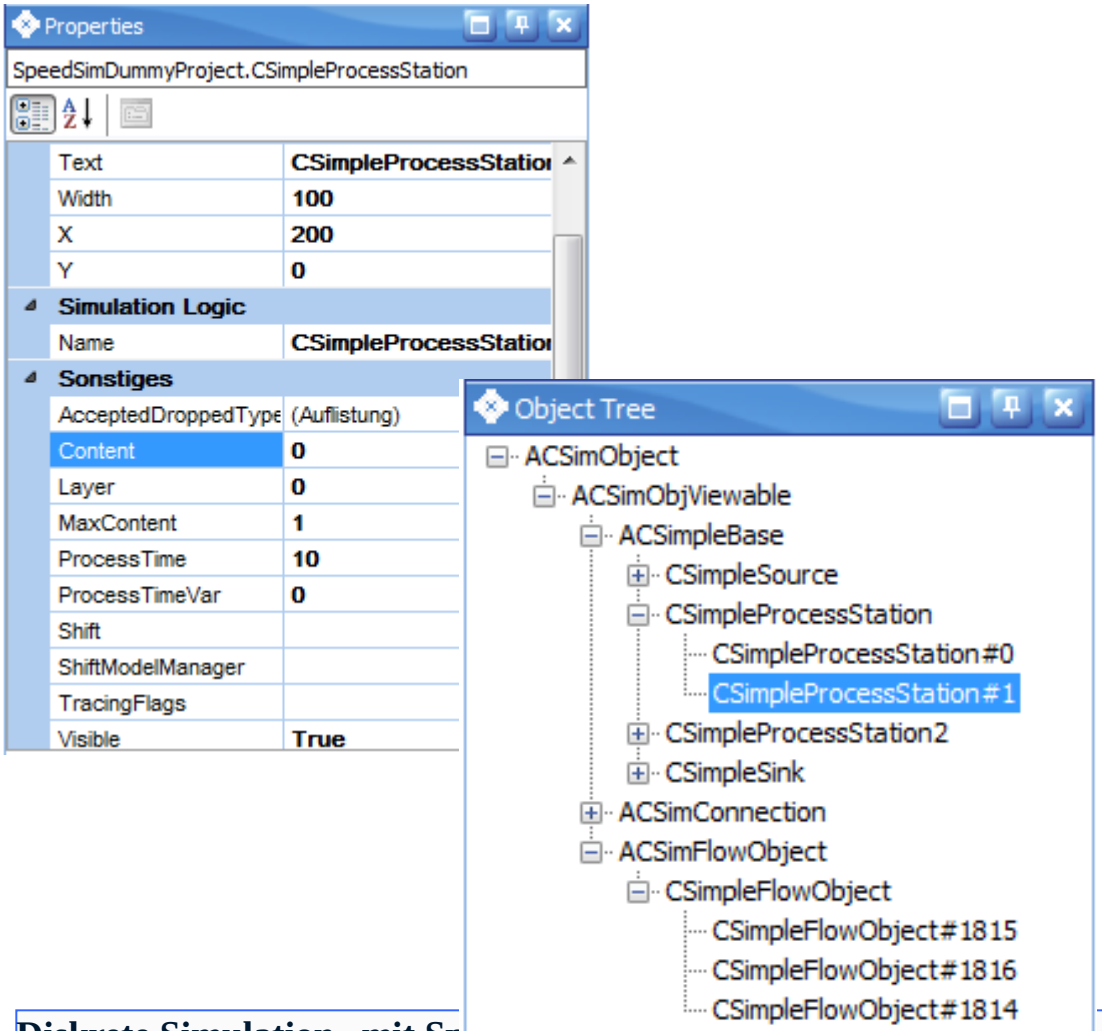
Option B.) Einbindung in SpeedSim-Editor

Mit vordefinierter GUI (Komponentenliste, Laufzeitsteuerung, Eigenschaften, Reporting,..)



Die SpeedSIM-Laufzeitkonfigurationen – Details





Die verfügbaren Simulationsobjekte werden in den Bereichen Statics und Dynamics angezeigt und können auf die Arbeitsfläche gezogen werden.

Alle Simulationsobjekte (auch die dynamischen) können über den Eigenschafteneditor (wie Visual-Studio) angezeigt und manipuliert werden.

Der Objektbaum zeigt alle zur Laufzeit aktiven Simulationsobjekte. Dabei sind teilweise auch die Vererbungslinien sichtbar und können Objekte auch für den Eigenschafteneditor selektiert werden.

Fazit zum SpeedSim-Simulator

Technologie-orientierte Bewertung

- sehr große Freiheiten bei Einbindung der gesamten Microsoft-Softwareumgebung
- Leistungsfähiges Objektmodell mit komplettem Zugriff auf alle Objektattribute und -methoden
    - Generierung neuer Simulationsobjekte mit abweichenden Verhalten durch einfache Vererbung und Überschreiben vorhandener Methoden

Anwendungsoptionen

- Stand-alone-Ansatz und komplexer SpeedSim-Editor erlauben sehr flexible Anpassungen an jeweilige Kundenanforderungen

Hierarchie der Simulationsobjekte

- noch in der Entwicklung (Feintuning der Objektabstufungen)
    - nur eine geringe Funktionalität in den abstrakten Objekten, um diese schlank und effektiv zu halten
    - stufenweiser Aufbau komplexer Funktionalitäten über Schichten von vererbten Simulationsobjekten

Gesamtfazit: Sowohl technologisch wie auch anwendungstechnisch sehr interessanter Ansatz. Noch große Freiheiten bei der Definition der aufsetzenden Objekthierarchie