SUMMER SCHOOL
Enabling Technologies – Part I
(Agents, Modeling, Notations for Automation)

Ordinarin
Automation and Information Systems (AIS)
Mechanical engineering,
Technische Universität München
www.ais.mw.tum.de; vogel-heuser@tum.de

TUM ASIA SUMMER SCHOOL
24TH – 30TH August 2017

Agenda – TUM Asia Summer School 29th August

<table>
<thead>
<tr>
<th>Time</th>
<th>29th August</th>
<th>30th August</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00AM – 10:30AM</td>
<td>Comparison of Industry 4.0, IoT, Smart Factory, Smart Data</td>
<td>Case Studies &amp; Successful Demonstrators: Applying Enabling Technologies</td>
</tr>
<tr>
<td>10:30AM – 11:00AM</td>
<td>MORNING TEA BREAK</td>
<td></td>
</tr>
<tr>
<td>11:00AM – 12:30PM</td>
<td>PART I: Enabling Technologies (Agents, Modelling Notations for Automation)</td>
<td>Smart Data Enabled Learning During Operation</td>
</tr>
<tr>
<td>12:30PM – 01:30PM</td>
<td>LUNCH BREAK</td>
<td></td>
</tr>
<tr>
<td>01:30PM – 03:00PM</td>
<td>PART II: Enabling Technologies (Agents, Modelling Notations for Automation)</td>
<td>Security and Human in the Loop</td>
</tr>
</tbody>
</table>

Complete Agenda: https://tum-asia.edu.sg/i4ss/

Univ.-Prof. Dr.-Ing. Birgit Vogel-Heuser| TUM Chair of Automation and Information Systems
Outline of part I

1. Principles and characteristics of agents
2. Application areas of software agents
3. Model Driven Engineering using Systems Modeling Language (SysML)
4. How to generate code out of a Model

Properties of technical agents

A technical agent is an encapsulated (hardware/software) entity with specified objectives. An agent endeavours to reach these objectives through its autonomous behaviour, in interacting with its environment and with other agents.

Source: VDI-Standard 2653 Sheet 1, 2010
### Terms of Agent Systems

<table>
<thead>
<tr>
<th>Encapsulation</th>
<th>State and behavior encapsulated in the agent</th>
</tr>
</thead>
</table>
| Target orientation     | • Agent-oriented behavior aiming a specific goal  
                          • Objectives either set by the developer (for example, to optimize the agent’s state permanently) or transferred from an user as an order at the execution time |
| Reactivity             | • Ability to perceive the environment and react appropriately |
| Autonomy               | • Control of internal state and behavior  
                          • Determines behavior mainly by agents themselves and not fixable from “outside”  
                          • Precondition for proactivity |
| Proactivity            | • Ability to act targeted and with foresight without direct influence from the outside (self-initiative)  
                          • Basis for proactivity is the existence of goals (see target orientation) |
| Interaction            | • Agents interact with each other at a high level of interaction (e.g. negotiations)  
                          • Basis for the interactions between agents is the underlying organizational context  
                          • Interactions of agents in order to achieve individual goals or to handle interdependencies |
| Persistence            | • Agent has continuous control flow, which depends on external activation  
                          • Ability to maintain its internal state during its life cycle |

### What are the possible problems and benefits of the control of the production by software agents? **Issues:**

### Advantages:
Morphological box for classifying agents—Which agent types are most appropriate for field level agents?

<table>
<thead>
<tr>
<th>Property</th>
<th>characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>autonomy</td>
<td>autonomous</td>
</tr>
<tr>
<td>architecture</td>
<td>reactive</td>
</tr>
<tr>
<td></td>
<td>memoryless</td>
</tr>
<tr>
<td>communication</td>
<td>synchronous</td>
</tr>
<tr>
<td>language</td>
<td>jointly defined</td>
</tr>
<tr>
<td>protocol definition</td>
<td>Petri nets</td>
</tr>
<tr>
<td>world model</td>
<td>together</td>
</tr>
<tr>
<td>adaptivity</td>
<td>Not adaptively</td>
</tr>
<tr>
<td>mobility</td>
<td>mobile</td>
</tr>
<tr>
<td>competition</td>
<td>cooperative</td>
</tr>
<tr>
<td>method of cooperation</td>
<td>negotiating</td>
</tr>
<tr>
<td>knowledge of their own abilities</td>
<td>Non resources</td>
</tr>
<tr>
<td>perception of the environment</td>
<td>through communication</td>
</tr>
<tr>
<td></td>
<td>through observation</td>
</tr>
</tbody>
</table>


Architecture of reactive agents

- Reactive agents (stimulus-response)
- No internal symbolic model of the environment
- No complex conclusion processes
- Compact, fault-tolerant and flexible agents
- Intelligence due to environment interactions
- Through continuous interaction intelligence arises and increases

Architecture of a deliberative agent

- Deliberative Agent
- Explicit, symbolic model of the environment
- Ability of logical conclusion as a basis for intelligent action
- Modeling the environment in first, knowledge base of agents

Outline of part I

1. Principles and characteristics of agents
2. Application areas of software agents
3. Model Driven Engineering using Systems Modeling Language (SysML)
4. How to generate code out of a Model
Fields of application for agents

Extended Enterprise
- Reaction time: 8 hours < x < 1 week
- Synchronized frequency: 1 hour < x < 1 day
- Any agent systems are applicable
- Programming languages: C++, C#, Java, ...

Merchandise management system
- Reaction time: 1 hour < x < 1 day
- Synchronized frequency: 60 seconds < x < 1 day
- Any agent systems are applicable
- Programming languages: C++, C#, Java, ...

Production planning and controlling systems
- Reaction time: 1 hour < x < 1 day
- Synchronized frequency: 60 seconds < x < 1 day
- Any agent systems are applicable
- Programming languages: C++, C#, Java, ...

Field control system
- Control of the production processes
- Reaction time: 1s < x < 60s
- Synchronized frequency: 1s < x < 60s
- Need for multi-agent systems
- Programming languages: IEC 61131, C++, C#, Java

Field control system
- Control of the machine physics
- Reaction time: partly < 1ms
- Synchronized frequency: partly < 1μs
- Need for multi-agent systems
- Programming languages: IEC 61131, C++, C#, Java

Source: See Lüder, A. Möglichkeiten und Grenzen Agentenbasierter Steuerungssysteme; 2006
Definition of an agent

A technical agent is an encapsulated (hardware/software) entity with specified objectives. An agent endeavours to reach these objectives through its autonomous behaviour, in interacting with its environment and with other agents.

Source: VDI-Standard 2653 Sheet 1, 2010

Guideline VDI/VDE 2653 – Part 1

Multi-agent systems in industrial automation

Part 1: Fundamentals
Terminology for agents used in automation, basic concepts and properties of agent systems
User groups: Operators and developers

Part 2: Development
Criteria for the selection and for the comparison of agent-oriented development methods and platforms
Analysis of existing agent-oriented development methods
User groups: Those who are concerned with the development of multi-agent systems

Part 3: Application
Reports (industry and universities) using multi-agent systems in different application fields
Definition of the problem and the problem solution with agents as well as the advantage of using agents
User groups: Those who are concerned with the development of multi-agent systems
Revision of the guideline VDI/VDE 2653 – Part 1

• New chapter „Learning“:
  – Integration of rule-based learning approaches, e.g. learning classifier systems, for critical but comprehensible control tasks
  – Classifier System reviews the applied rules. → Good rules are kept in rule set. Unusable rules are deleted in rule set
  – Initial learning process can be reduced through a priori rules
  – Using rules based on fuzzy logic improves the adaptation of the rules

• New chapter „Energy“:
  – New sub-chapters „Approach for controlling small distributed energy systems“, „Multimodal energy systems“, „Home automation“, „Microgrids“, „Virtual power plants“, „Energy management in vehicles“
  – Switch to renewable energies requires small and distributed energy conversion systems
  – Realisation through multi-agent system

• New chapter „Mobile transport robots“:
  – Using multi-agent systems for the control and communication of mobile transport robots
  – Collaboration between MAS to accomplish joint tasks

• New chapter „Data analysis“:
  – Use the resulting data from different systems to generate more information and knowledge for the expansion of business areas and models from the collected data
  – Horizontal and vertical coupling of heterogeneous systems
  – If damage was recorded, compensation strategies are applicable

• New chapter „Smart Environment“:
  – Complexes and hierarchical control structures are implemented by decentralized intelligent algorithms
  – Information about the building and the current situation are stored in a knowledge-based, semantic building model

Outline of part I

1. Principles and characteristics of agents
2. Application areas of software agents
3. Model Driven Engineering using Systems Modeling Language (SysML)
4. How to generate code out of a Model
Product-Service-Systems Innovations

- Distributed and Collaborative Development
- Cross-Domain Dependency
- High Flexibility and Variability in Innovation Processes

Innovation Process of the Product-Service System

- Manufacturing Planning
- Business Model
- Mechatronic Modeling
- Product Development

Overview on the CRC 768 Model Network
Visualization of the inconsistencies in coupled engineering models

Information boxes
- Provide information about the inputs and outputs
- description on link between models,

Arrow types
- **Red**: Input
- **Blue**: Output
- **Black**: Yet invalid connection
- **Yellow**: Zoom of part of the model
- **Dotted**: Indirect information flow

UML-based implementation of IEC 61131-3 code with SysML expansion for software agents

SysML-Plugin „KREA“ for TwinCAT 3 (CODESYS Family)
Tank with upper and lower filling level sensors, valve and pump


Level 0 - fault tolerant sensor (Model based with agent)
Redundancy Model of Tank according to [SWL+13] and mapping to PAR


Mapping of PAR onto initialization of the redundancy matrix

Agent-based self-aware, self-describing CPPS-module


Introduction of the small lab scale production system pick-and-place-unit (PPU)
Intelligent Software Components on PLCs

**Diagnosis Module**
- Evaluation of sensors values
- Execution of failure diagnosis

**Control Module**
Control of the plant module or other subagents

**Knowledge Base**
Models of the agents’ local knowledge

**Dynamic Reconfiguration**

**Tool-Supported Development of Agents Knowledge Base**

**AD: Technical Process**

**BDD: Software Structure**

**PD: Redundancy Model**

**PD: Tolerance Model**

**Agent Models**
- Redundancy Model
- Tolerance Model

**Self-Aware Sensor Agent**

**Main Routine**

**Analytical Dependency**

**Production Plant**

**Real-Time Capable Fault Tolerant Software**

Source: Frank et al. 2011, Schütz et al. 2012, DFG funded project KREAagentuse
Construction of SysML diagrams

SysML

Requirements diagram

Behavior diagrams

Use case diagram

Activity diagram

Sequence diagram

State machine diagram

Structure diagrams

Package diagram

Block definition diagram

Internal block diagram

Parametric diagram

UML 2

Not used by SysML

SysML

Extension of UML 2

UML reused by SysML

Source: http://omgsysml.org, 2007

Systems Modeling Language (SysML)
Relationship between model and diagrams

requirement diagram

block definition diagram

parameter diagram

SysML offers 9 types of diagrams (structure and behavior) for the representation of different system aspects for different stakeholders

→ Base, to develop mechatronic PSS model-based and consistently

→ Diagrams to support the modeling from the early requirement design to the realization

Univ.-Prof. Dr.-Ing. Birgit Vogel-Heuser| TUM Chair of Automation and Information Systems
Requirement diagram of the stamping process

Requirement in text form:
The workpiece must be stamped for at least 10s with a force of at least 10N.

Chart name in the tool

The workpiece must be stamped for at least 10s with a force of at least 10N.

```
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>stamping process</td>
<td>&quot;The workpiece must be stamped for 10s with a force of at least 10N&quot;</td>
</tr>
<tr>
<td>2</td>
<td>stamping force</td>
<td>&quot;The stamping force must be greater than 10N.&quot;</td>
</tr>
<tr>
<td>3</td>
<td>stamping time</td>
<td>&quot;The stamping process must take more than 10s.&quot;</td>
</tr>
</tbody>
</table>
```

«refine» : Refine relationship describes that a model element describes the properties of a requirement in more detail.

«satisfy» : Satisfy relationship describes design elements that fulfills a requirement.

«verify» : Verify relationship combines a test case with the requirement that is checked by the test case.
Relationship between requirements diagram and parametric diagram

Defined requirements, which are specified in requirements diagrams, are met in “satisfy” relationship of “constraint” blocks in parametric diagrams.

```
req [Requirement] StampProcess
  «constraint»
  MaximumStampForce
    values
    A_piston : m²
    p_max : Pa
    F_max : N
  «satisfy»

par [ConstraintBlock] MaximumStampForce
  p_max : Pa
  A_piston : m²
  «constraint»
  conversion: p_to_f
    \{ F = p*A \}
  F : N
  F_max: N
```

Structural modeling in SysML

Overview BDD and IBD

- The **Block Definition Diagram (BDD)** provides a "black box" view of a system block. It also describes the hierarchy of the sub-blocks of the system building block.
- The BDD can be compared with the first page of a building instruction for a piece of furniture. It shows the type and number of all parts in the package.

- The **Internal Block Diagram (IBD)** gives a "white box" or internal view to a system block. It shows how all parts of the system block shown are connected.
- The IBD can be compared with the last page of a building instruction for a piece of furniture. It shows how the individual parts are concretely connected.

The **structure** of a system block is described in the **Block Definition Diagram** and the role of its **system components** is described in the **Internal Block Diagram**.
Relationship between block definition diagram (BDD) and internal block diagram (IBD)

The relationships between parts of a block defined in the BDD are represented in the IBD.

Brainstorming

Which ports, which interfaces, which object flows have to be specified for the stamp of the PPU?
Structure model – Internal block diagram (IBD)

Example: Sorting plant: Block stamp

- Modelling power is neglected to retain the overview
- Air and workpiece are concrete objects, which are transmitted

SysML diagrams

Source: http://omgsysml.org, 2007
Notation for state diagram

A simple state is a state that does not have substates - it has no regions and it has no submachine. The compartments of the state are: name compartment, internal activities compartment and internal transitions compartment.

The transition is a directed relationship between a source state and a target state. Definition: 1. Trigger to execute state transition, 2. Condition to enter transition, 3. Behavior (action), which will be executed during the transition.

The initial pseudostate. It shows on the first state. Outgoing transitions can only contain actions, but not contains.

The final state.

The choice pseudostate enables to choose between transitions and to merge them.

Behavior model – state diagram
state description of the sorting plant

Events / Actions() in state diagram

entry / Action() is executed, while entering the state
do / Action() is executed as long as the condition is active, or until it is completed
exit / Action() is executed, while exiting the state

Transitions in state diagram:
Events [Condition]/Action

- In a state several methods of a class can be called (or other behavior diagrams)
- The state is exited if the subsequent transition is active: Then the Do-action is canceled and the Exit-action is executed once
State-based behavior—Important boundary conditions to model the crane

- Crane is able to move to retracted and extended positions
- A pneumatic cylinder is used to extend the crane

Behavior model: Notations of the activity diagram

- **An activity** describes processes, which consist of several elementary actions.
  - **<precondition>**: activity is started only when the precondition is satisfied.
  - **<postcondition>**: After completing the activity, post-condition must be fulfilled.

- **Action nodes** describe elementary process steps or subordinate activities. It is the most basic model element to describe behavior.

- Alternatively, **object nodes** can be represented by rectangles. They can be used as incoming or outgoing parameters in activities.

- **Activity edges** are directed relations between two nodes of an activity. A distinction is made between control and object flow. Both are graphically represented at the same way. Considering object flows, the arrow shows from or to an object node.

- **A start node** is a starting point of a sequence in an activity diagram.

- An **end node** ends the sequence, meaning all actions and control flows.
Outline of part I

1. Principles and characteristics of agents
2. Application areas of software agents
3. Model Driven Engineering using Systems Modeling Language (SysML)
4. How to generate code out of a Model
UML-based implementation of IEC 61131-3 code with SysML expansion for software agents

**UML-Plugin for CODESYS V3**

- Class diagram
- State diagram
- Activity diagram

**Code generation for IEC 61131-3 (bidirectional synchronization)**

- Block definition diagram
- Parametric diagram

**SysML-Plugin „KREA“ for TwinCAT 3 (CODESYS Family)**

Construction of SysML diagrams

- Requirements diagram
- Behavior diagrams
  - Use case diagram
  - Activity diagram
- Structure diagrams
  - Package diagram
  - Block definition diagram
  - Internal block diagram
  - Parametric diagram

**Source:** http://omgsysml.org, 2007
Relationship between requirements diagram and parametric diagram

Defined requirements, which are specified in requirements diagrams, are met in "satisfy" relationship of "constraint" blocks in parametric diagrams.

Defined requirements, which are specified in requirements diagrams, are met in "satisfy" relationship of "constraint" blocks in parametric diagrams.

Exemplary mapping between SysML and IEC 61131-3: block definition diagram

Example: energy calculation for the vertical fall

SysML

IEC 61131-3

(Block → Function Block)

FUNCTION_BLOCK EnergyCalculation
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR

Acceleration: REAL;
StartHeight : REAL;
Velocity: REAL;
Mass : REAL;
Height : REAL;
PotentialEnergy: REAL;
KineticEnergy: REAL;
END_VAR
Exemplary mapping between SysML and IEC 61131-3: Parametric diagram

Parametric diagram
- Calling of IEC 61131-3 functions and function blocks
- Visualization and structuring of parametric relationships

**SysML**

- Acceleration:REAL
- Mass:REAL
- PotentialEnergy:REAL
- KineticEnergy:REAL

**IEC 61131-3**

- **Function**
  ```plaintext
  FUNCTION Ekin : REAL
  VAR_INPUT
  m : REAL;
  v : REAL;
  END_VAR
  VAR_OUTPUT
  E : REAL;
  END_VAR
  Ekin := m * v;
  ```

- **Function Block**
  ```plaintext
  FUNCTION_BLOCK Height
  VAR_INPUT
  v : REAL;
  h0 : REAL;
  END_VAR
  VAR_OUTPUT
  h : REAL;
  END_VAR
  /*...*/
  ```

Univ.-Prof. Dr.-Ing. Birgit Vogel-Heuser| TUM Chair of Automation and Information Systems
Exemplary mapping between SysML and IEC 61131-3: Parametric diagram

**Parametric diagram**
- Calling of IEC 61131-3 functions and function blocks
- Visualization and structuring of parametric relationships

**SysML**

```
ActualSpeed (a := Acceleration, h0 := StartHigh);
Speed := ActualSpeed.h;
ActualHeight (v := Speed, h0 := StartHigh);
Height := ActualHeight.h;
PotentialEnergy := Epot(m := Mass, h := height);
KineticEnergy := Ekin(m := Mass, v := Speed);
```

**Problem**
What information is missing in pure SysML parametric diagram to automatically generate code?

**IEC 61131-3**

Advantage
- inherent documentation of entire code, readability, maintainability ("Model is Code")

Improvement of reuse of standard library elements and -functions

Graphical representation improves understandability

**What information is missing in pure SysML parametric diagram to automatically generate code?**

---

How can knowledge of software agents be modelled?
Which kinds of knowledge have to be modelled?

Which kinds of knowledge can be modelled with SysML?

How can this knowledge be used during runtime?

What is missing?
Thank you for your attention!

Univ.-Prof. Dr.-Ing. Birgit Vogel-Heuser

Ordinaria
Automation and Information Systems (AIS)
Mechanical engineering,
Technische Universität München
www.ais.mw.tum.de; vogel-heuser@tum.de