23. Action-Oriented Design Methods

Prof. Dr. Uwe Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

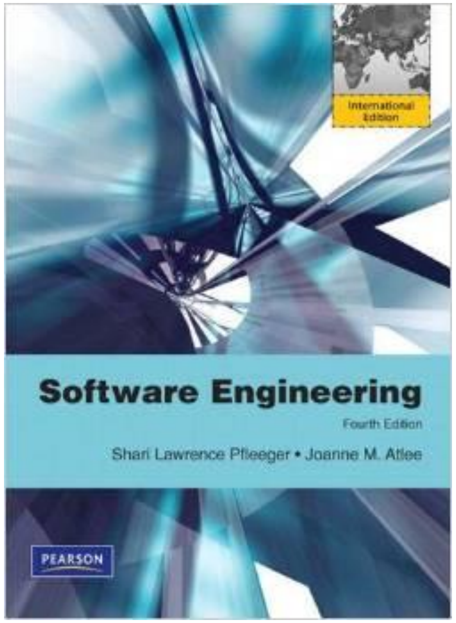
http://st.inf.tu-dresden.de/teaching/swt2

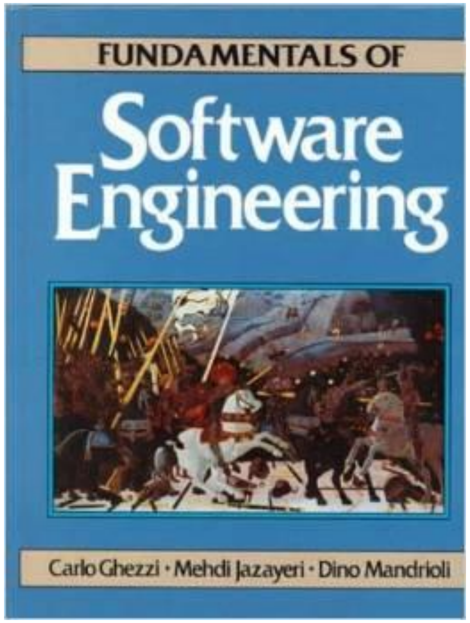
WS 17/18 15.01.2018

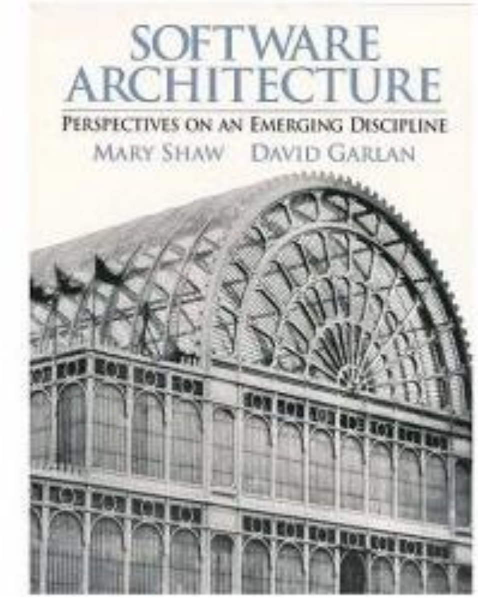
Lecturer: Dr. Sebastian Götz

Obligatory Reading

- S. L. Pfleeger and J. Atlee:
  Software Engineering: Theory and Practice.
  Pearson. 2009.
  - Chapter 5 (Designing the Architecture)
- C. Ghezzi, M. Jazayeri and D. Mandrioli:
  Fundamentals of Software Engineering.
  Prentice Hall. 1992.
  - Chapter 4 (Design and Software Architecture)
- M. Shaw and D. Garlan:
  Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.







23.1 Action-Oriented Design

- Action-oriented design is similar to function-oriented design, but admits that the system has states.
- It asks for the internals of the system
- Actions require state on which they are performed (imperative, state-oriented style)
- Actions are running in parallel
- **Decomposition strategy:**
  - Divide: finding subactions
  - Conquer: grouping to modules and processes
- Result: reducible action system
- Example: all function-oriented design methods can be made to action-oriented ones, if state is added
  - State machine based design for embedded systems; Petrinet based design (with distributed state)
  - Imperative programming

What are the actions the system should perform?

What are the subactions of an action?

Which state does an action change?

Structured Analysis and Design (SA/SD)

- **Structured Analysis (SA)** is a specific variant of action-oriented design with *processes (process-oriented design, data-flow based design)*

  [DeMarco, T. Structured Analysis and System Specification, Englewood Cliffs: Yourdon Press, 1978]

- Notations of SA:
  - Function trees (action trees, process trees): decomposition of system functions
  - Data flow diagrams (DFD), in which the actions are called *processes*
  - Data dictionary (context-free grammar) describes the structure of the data that flow through a DFD
    - Alternatively, class diagrams can be used
  - Pseudocode (minispecs) describes central algorithms (state-based)
  - Decision Table and Trees describes conditions (see later)

Why SA is Important

- Usually, action-oriented design is *structured*, i.e., based on hierarchical stepwise refinement.
- Resulting systems are
  - *reducible*, i.e., all results of the graph-reducibility techniques apply.
  - *parallel*, because processes talk with streams
  - *local*, because processes write to local shared memory
  - *easy to distribute*, because no global memory exists

- On the highest abstraction level, define the **context diagram**:
  - **Elaboration**: Define interfaces of entire system by a top-level action tree
  - **Elaboration**: Identify the input-output streams most up in the action hierarchy
  - **Elaboration**: Identify the highest level processes
  - **Elaboration**: Identify stores
- **Refinement**: Decompose action tree hierarchically
- **Change Representation**: transform action tree into process diagram (action/data flow)
- **Elaboration**: Define the structure of the flowing data in the Data Dictionary
- **Check** consistency of the diagrams
- **Elaboration**: Minispecs in pseudocode

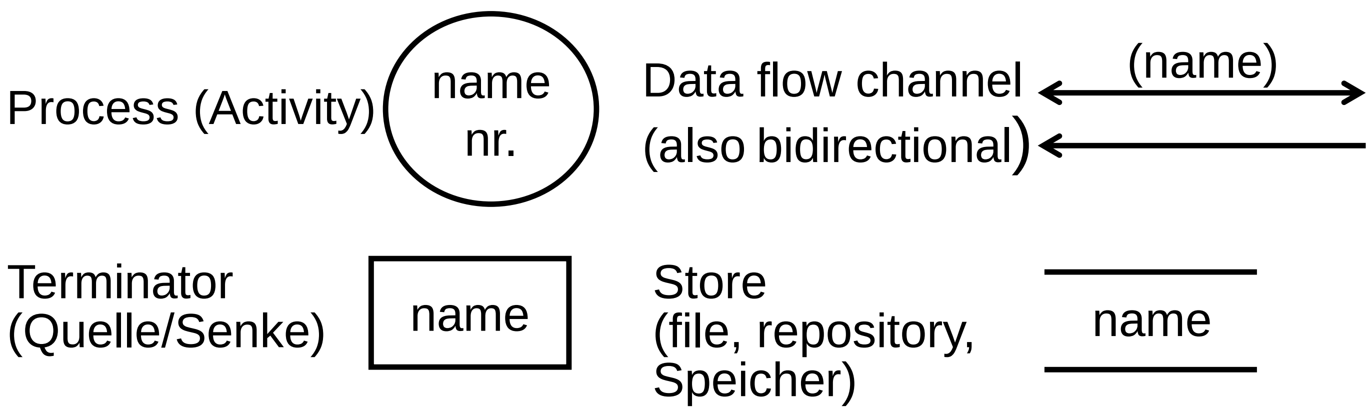
Data-Flow Diagrams (Datenflussdiagramme, DFD)

- DFD are a special form of Petri nets
- They are also special workflow languages without repository and global state
  - DFD use local stores for data, no global store
  - Less conflicts on data for parallel processes
- Good method to model parallel systems

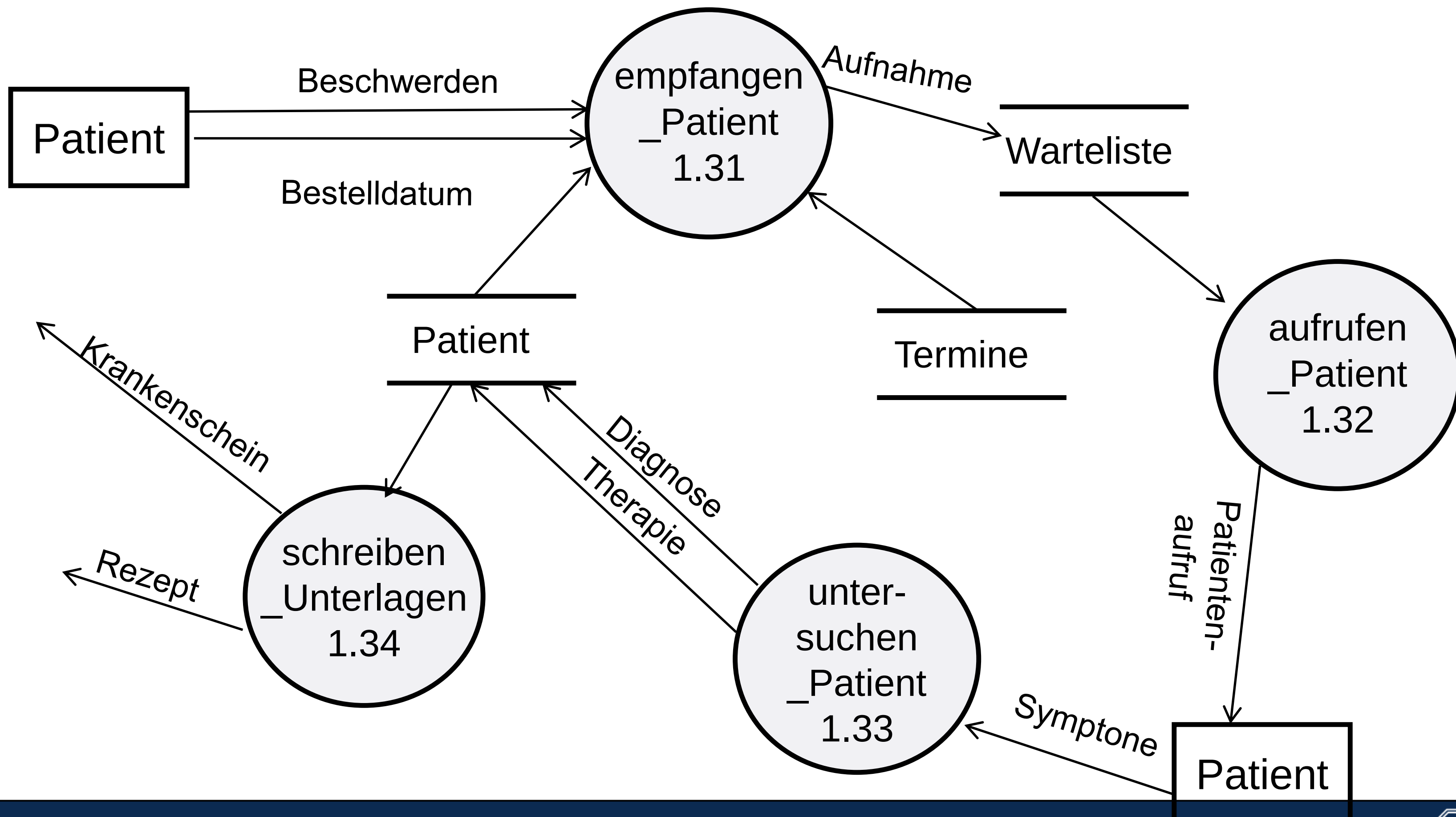
Modeling with DFD

- A **data-flow diagram** is a reducible (hierarchic) net of processes linked by channels (streams, pipes)
- Context diagram: top-level, with terminators
- Parent diagrams, in which processes are point-wise refined
- Child diagrams are refined processes
- Refinement can be syntactic or semantic
- **Data dictionary** contains types for the data on the channels
- **Mini-specs** (Minispezifikationendienen) specify the atomic processes and their transformationen
  - with Pseudocode or other high-level languages

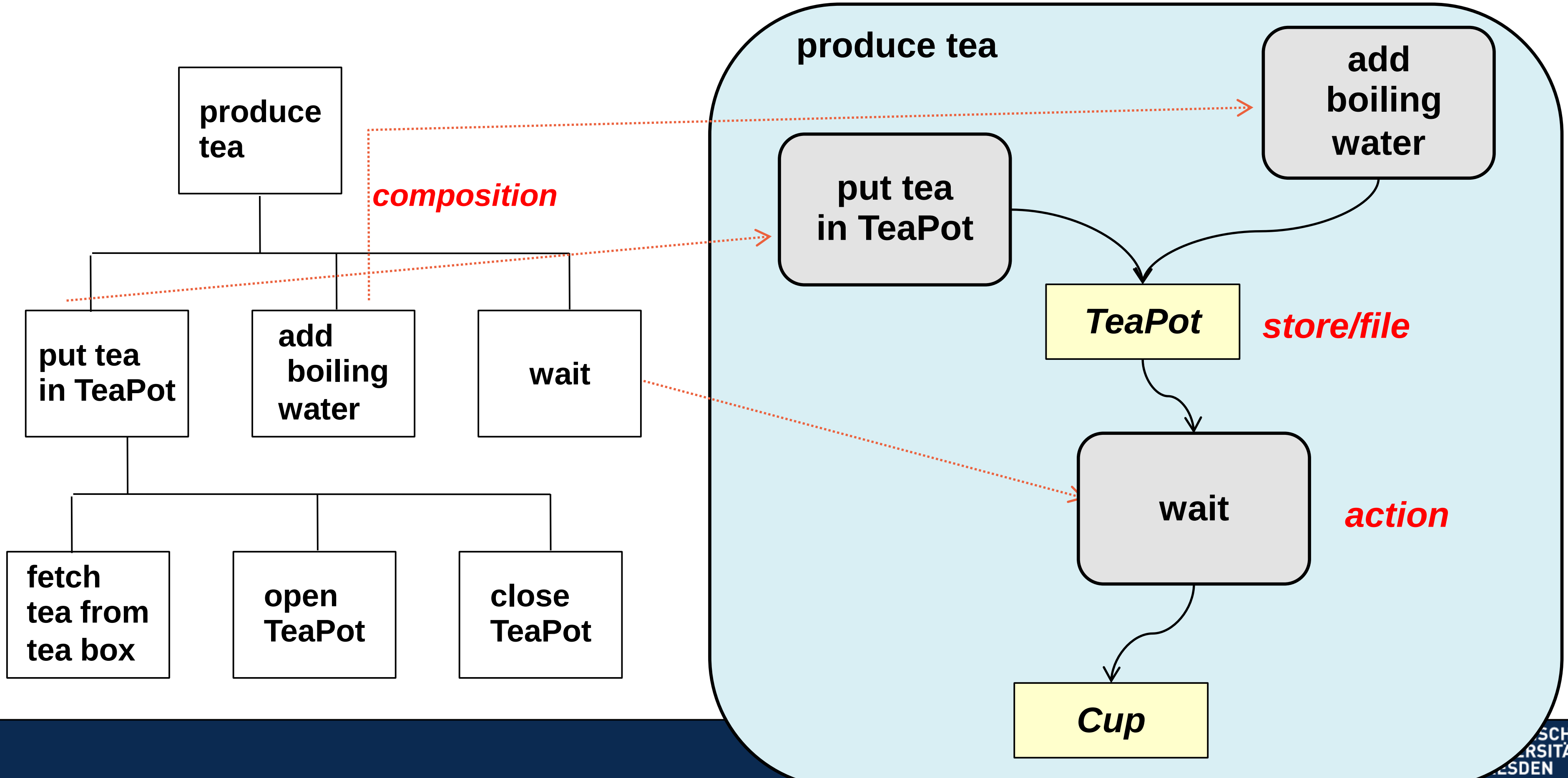
Symbols (SA/Balzert):



Ex.: DFD "treat_Patient" (Original SA Notation)

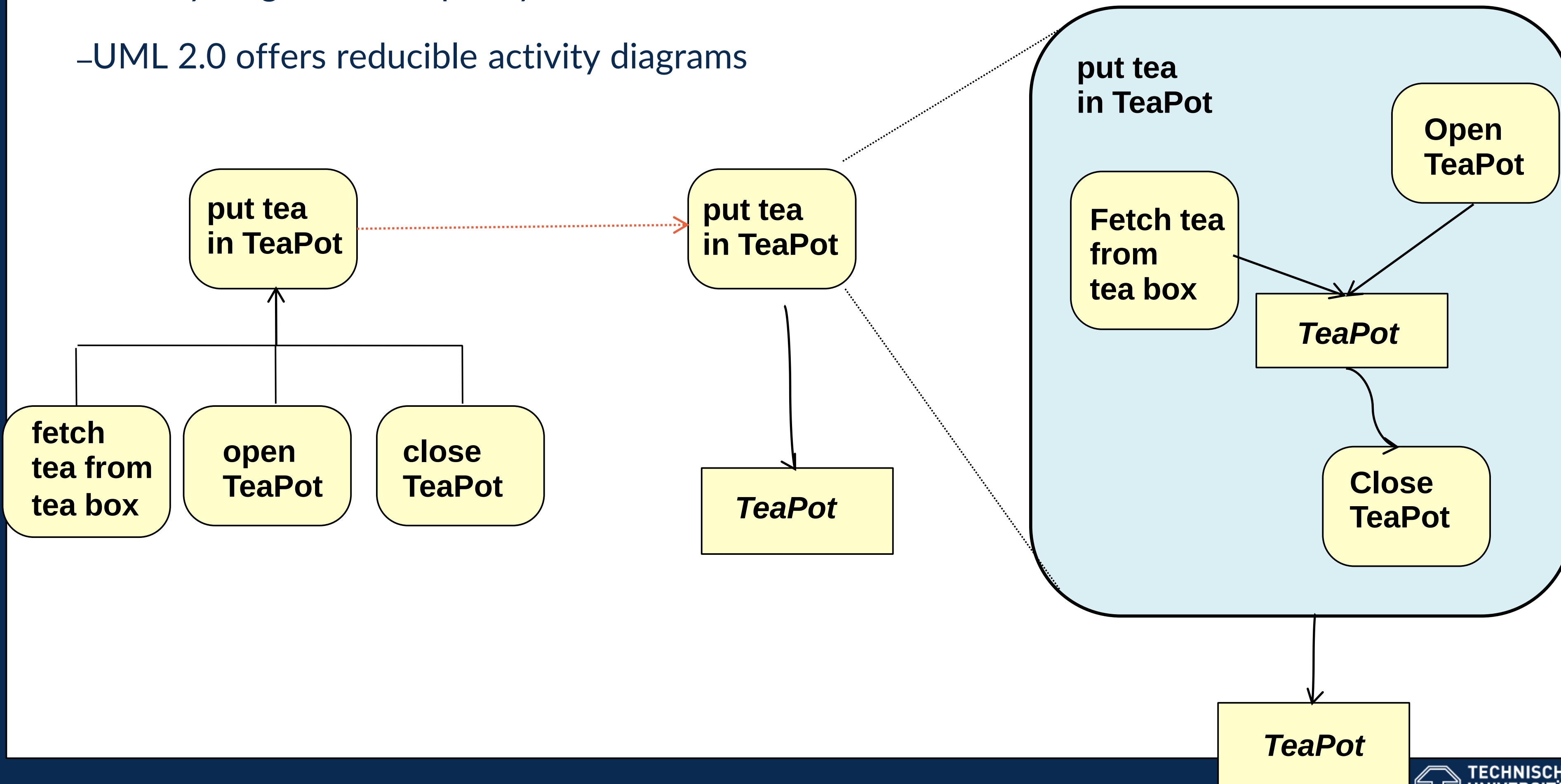


- **Action trees can be derived from function trees and function nets**
- DFD are homomorphic to Action trees, but add stores and streams
- **RepresentationChange**: Construct an action tree and transform it to the processes of a DFD



Pointwise Refinement of Actions

- Subtrees in the function tree lead to reducible subgraphs in the DFD
- UML action trees can be formed from activities and aggregation
- Activity diagrams can specify dataflow
  - UML 2.0 offers reducible activity diagrams



Typing Edges with Types from the Data Dictionary

- In an SA, the **data dictionary** collects data types describing the context free structure of the data flowing over the edges. To this end, a **data definition language (DDL)** is required:

- **Grammar**: For every edge in the DFDs, the context-free grammar contains a non-terminal that describes the flowing data items

- **Entity-Relationship Diagram** (or its object-oriented variant MOF)

- **UML class diagram**: classes describe the data items

- Grammars are written in **Extended Backus-Naur Form (EBNF)** with the following rules:

<table>
<tr><th></th><th>Notation</th><th>Meaning</th><th>Example</th></tr>
<tr><td></td><td>::= or =</td><td>Consists of</td><td>A ::= B.</td></tr>
<tr><td>Sequence</td><td>+</td><td>Concatenation</td><td>A ::= B+C.</td></tr>
<tr><td>Sequence</td><td><blank></td><td>Concatenation</td><td>A ::= B C.</td></tr>
<tr><td>Selection</td><td>I or [|]</td><td>Alternative</td><td>A ::= [B | C].</td></tr>
<tr><td>Repetition</td><td>{ }^n</td><td></td><td>A ::= { B }^n.</td></tr>
<tr><td>Limited repetition m</td><td>{ } n</td><td>Repetition from m to n</td><td>A ::= 1{ B }10.</td></tr>
<tr><td>Option</td><td>( )</td><td>Optional part</td><td>A ::= B (C).</td></tr>
</table>

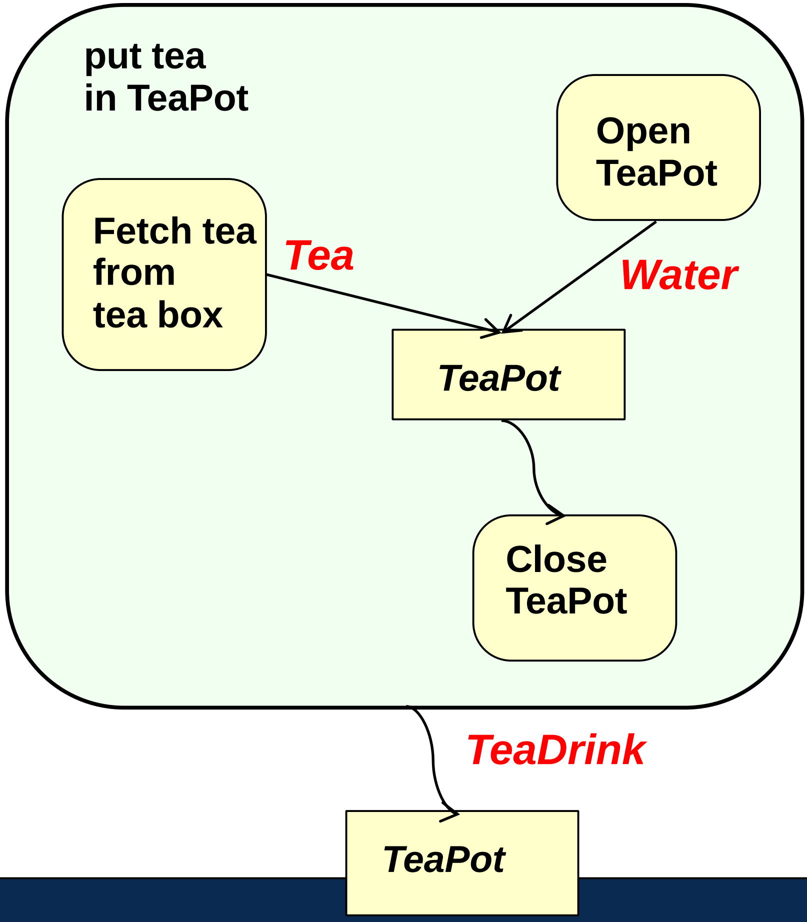
Example Grammar in Data Dictionary

- Describes types for channels

```
DataInPot ::= TeaPortion WaterPortion.
TeaAutomatonData ::= Tea | Water | TeaDrink.
Tea ::= BlackTea | FruitTea | GreenTea.
TeaPortion ::= { SpoonOfTea }.
SpoonOfTea ::= Tea.
WaterPortion ::= { Water }.
```

Exc. Read every rule aloud!

- Nonterminals from the data dictionary become types on flow edges
- Alternatively, classes from a UML class diagram can be annotated



Minispecs in Pseudocode

- **Minispecs** describes the processes in the nodes of the DFD in pseudo code. They describe the data transformation of every process
- Here: specification of the minispec attachment process:

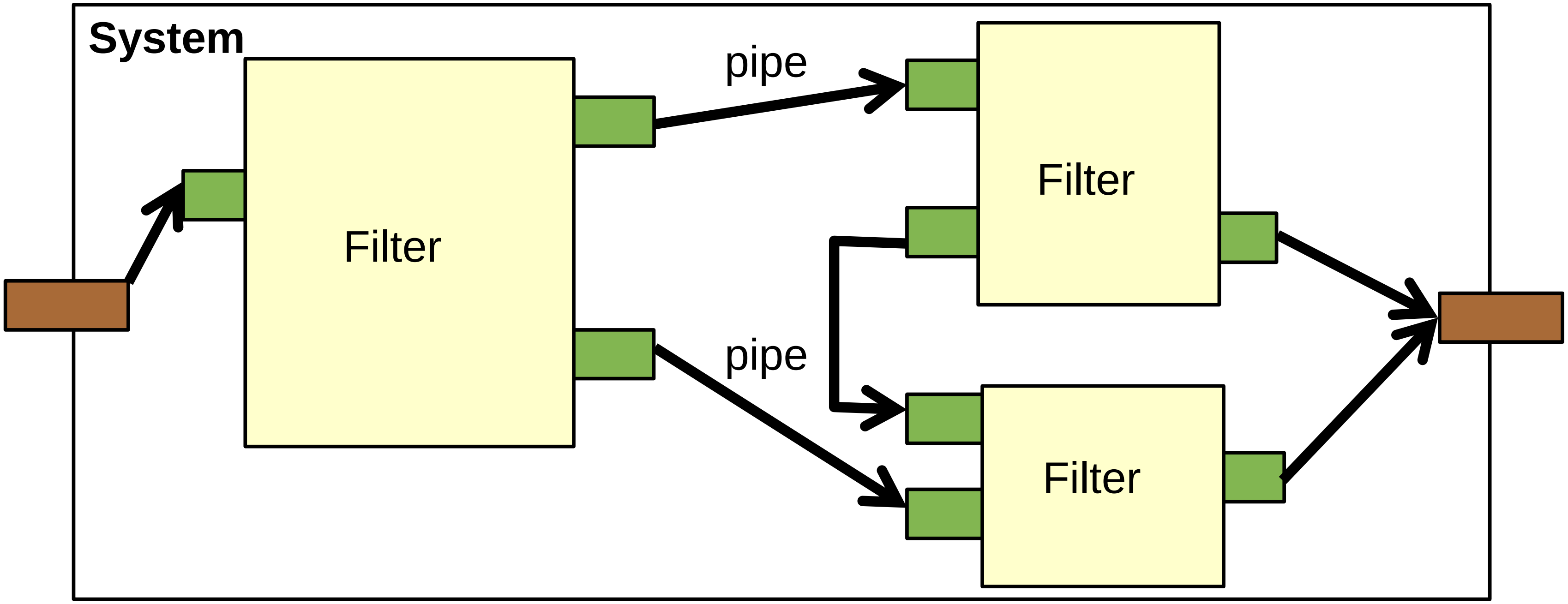
```
procedure: AddMinispecsToDFDNodes
target.bubble := select DFD node;
do while target-bubble needs refinement
   if target.bubble is multi-functional
          then decompose as required;
                  select new target.bubble;
                       add pseudocode to target.bubble;
          else no further refinement needed
   endif
enddo
end
```

Difference to Functional and Modular Design

- SA focusses on actions (parallel activities, processes), not functions
  - Describe the *continuous* data-flow through a system
  - Describe stream-based systems with pipe-and-filter architectures
- Actions are parallel processes
  - SA can easily describe parallel systems
- Function trees are interpreted as action trees (process trees) that treat streams of data

Result: Data-Flow-Based Architectural Style

- *SA/SD design leads to dataflow-based architectural style with *continuous data flow forward* through the system*
- Processes exchanging streams of data via *ports*
- Components are called **filters**, connections are **pipes (channels, streams)**



Application Areas are Manifold

- Shell programming with pipes-and-filters
  - tcsh, bash, zsh (Linux)
  - Microsoft Powershell
- LabView programming for engineers
  - Integration and differenciation possible, simulation of continuous variables
- Image processing systems
  - Image operators are filters in image data-flow diagrams
- Signal processing systems (DSP-based embedded systems)
  - The satellite radio
  - Video processing systems
  - Car control
  - Process systems (powerplants, production control, ...)
- Content management systems (CMS)
  - Content data is piped through XML operators until a html page is produced
- Stream-based business workflows for data-intensive business applications

What Have We Learned

- Besides object-oriented design, structured, action-oriented design is a major design technique
  - It will not vanish, but always exist for certain application areas
  - If the system will be based on stream processing, system-oriented design methods are appropriate
  - System-oriented design methods lead to reducible systems
- Don't restrict yourself to object-oriented design