23. Action-Oriented Design Methods

1

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de
Version 12-1.1, 09.01.13

1) Action-Oriented Design
2) Structured Analysis/Design (SA/SD)
3) Structured Analysis and Design Technique (SADT)
4) Workflow nets

Obligatory Reading

2

- Balzert, Kap. 14
- Ghezzi Ch. 3.3, 4.1-4, 5.5
- Pfleeger Ch. 4.1-4.4, 5

Why SA is Important

3

- 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.
  - Often *parallel*, because processes talk with streams
- SA and SADT are important for *embedded systems* because resulting systems are parallel and hierarchic
- **Mashups** are web-based data-flow diagrams and can be developed by SA (see course Softwarewerkzeuge)

23.1 Action-Oriented Design

4

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)
- 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

What are the actions the system should perform?
What are the subactions of an action?
Which state does an action change?

23.2 Action-Oriented Design with SA/SD

Data-flow connects processes (parallel actions)

State is implicit in the atomic processes, not explicit in the global, architectural specifications

Structured Analysis and Design (SA/SD)

- A specific variant of action-oriented design is *process-oriented design (data-flow based design)*
  - [DeMarco, T. Structured Analysis and System Specification, Englewood Cliffs: Yourdon Press, 1978]
- Representation
  - 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)

Structured Analysis and Design (SA/SD) – The Process

- On the highest abstraction level, on 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 function 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

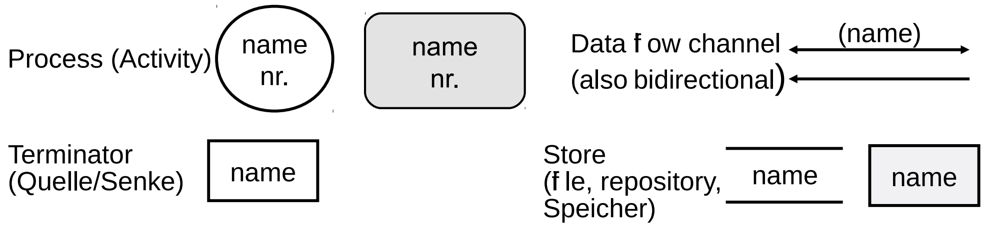
Data-Flow Diagrams (Datenflussdiagramme, DFD)

- DFD are special 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

DFD-Modeling

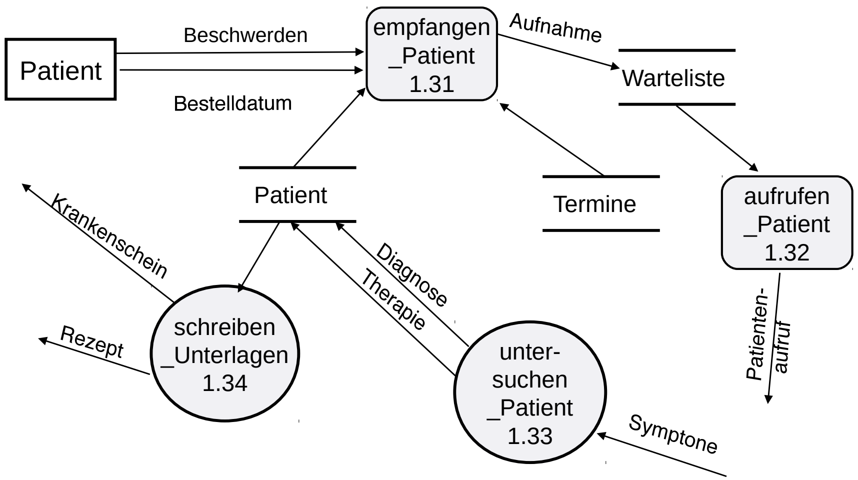
- Reducible (hierarchic) nets of processes linked by channels (streams, pipes)
  - **Context diagram:** top-level, with terminators
  - Parent diagrams, in which processes are point-wise refined
  - Child diagrams are refined processes
  - Refinement can be syntactic or semantic
- Data dictionary contains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

Symbole (Balzert/UML):



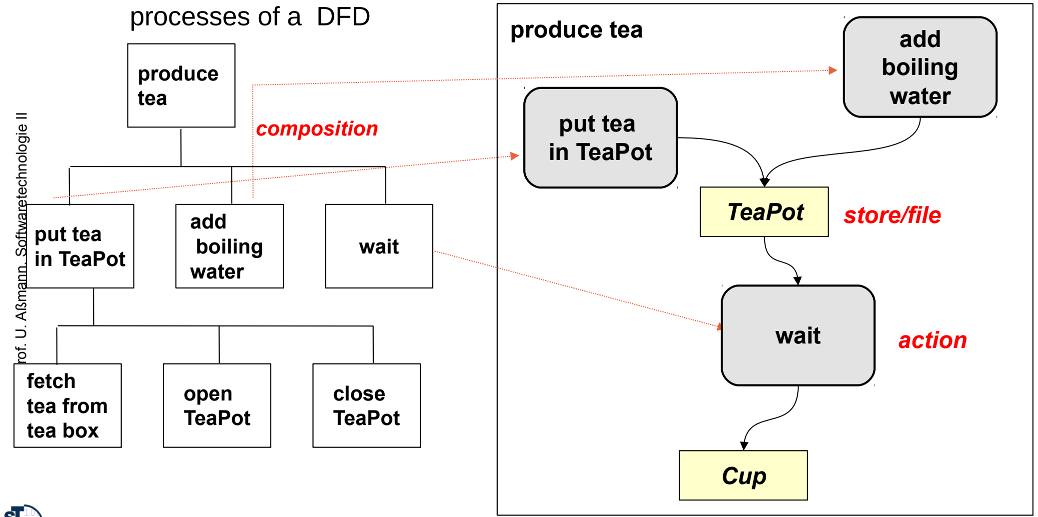
Ex.: DFD "treat_Patient"

- UML uses ovals for activities; SA uses circles



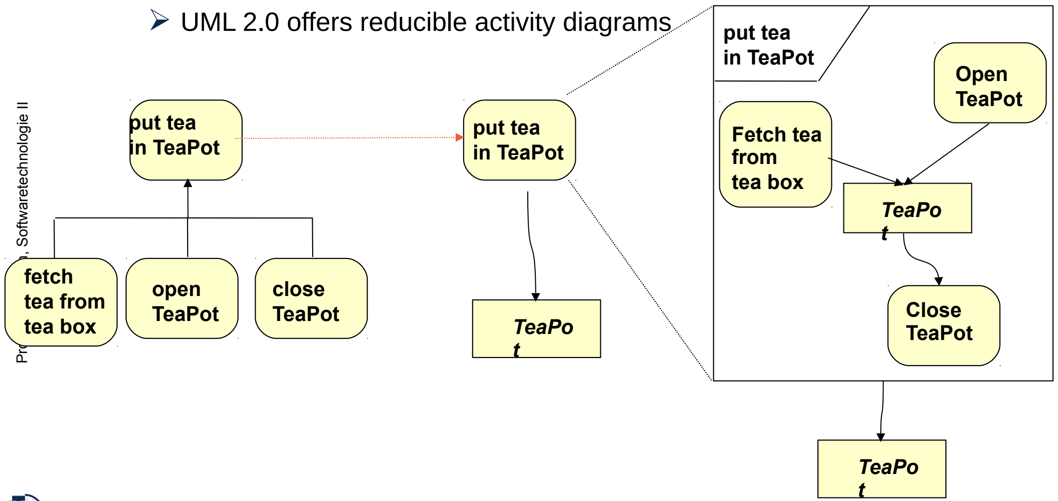
Action Trees and DFDs

- Action trees can be derived from function trees
- Action trees are homomorphic to DFD, except containing activities
- 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
  - Grammar: For every edge in the DFDs, the context-free grammar contains a non-terminal that describes the flowing data items
  - UML class diagram: classes describe the data items
- Grammars are written in Extended Backus-Naur Form (EBNF) with the following rules:

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

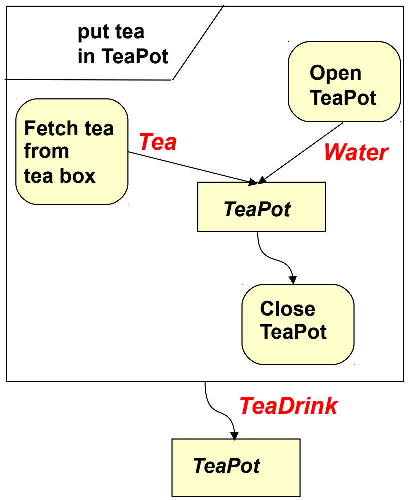
Example Grammar in Data Dictionary

- Describes types for channels

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


Adding Types to DFDs

- 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
```


Good Languages for Pseudocode

- SETL (Schwartz, New York University)
  - Dynamic sets, mappings, Iterators
  - http://en.wikipedia.org/wiki/SETL
  - http://randoom.org/Software/SetlX
- PIKE (pike.ida.liu.se)
  - Dynamic arrays, sets, relations, mappings
  - Iterators
- ELAN (Koster, GMD Berlin)
  - Natural language as identifiers of procedures
  - http://en.wikipedia.org/wiki/ELAN_(programming_language)
  - One of the sources of our TUD OS L4: http://os.inf.tu-dresden.de/L4/l3elan.html
- Smalltalk (Goldberg et.al, Parc)
- Attempto Controlled English (ACE, Prof. Fuchs, Zurich)
  - A restricted form of English, easy to parse

Structured Analysis and Design (SA/SD) - Heuristics

- Consistency checks
  - Isomorphism rule between diagrams (e.g., between function trees and DFD)
  - Corrections necessary in case of structure clash between input and output formats
- Verification
  - Point-wise refinement can be proven to be correct by bisimulations of the original and refined net
- Advantage of SA
  - Hierarchical refinement: The actions in the DFD can be refined, I.e., the DFD is a reducible graph
  - SA leads to a hierarchical design (a component-based system)

Difference to Functional and Modular Design

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

Implementation Hints

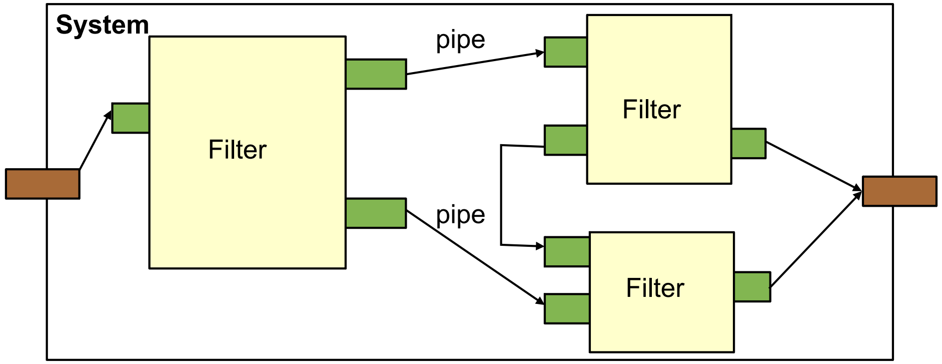
21

- Channels (streams): implement with Design Pattern Channel (ST-1)
- Processes: Ada-95 has parallel processes
- If actions should be undone (in interactive editing), or replayed, they can be encapsulated into Command objects (see design patterns Command and Interpreter)
- If actions work on a data structure, design pattern Visitor allows for extensible action command objects

Result: Data-Flow-Based Architectural Style

22

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



Examples

23

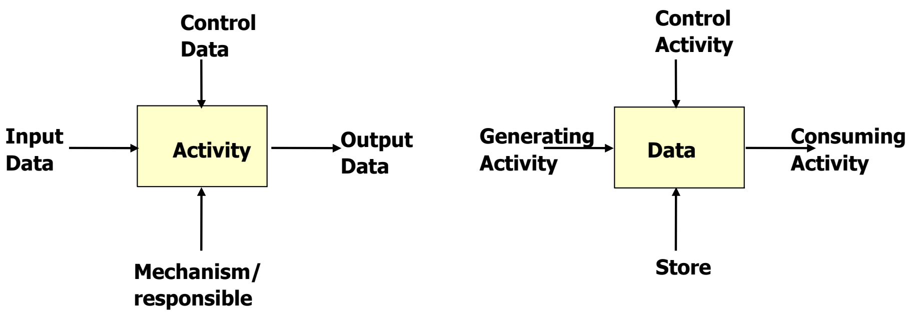
- Shell programming with pipes-and-filters
  - zsh
  - Microsoft Powershell
- Image processing systems
  - Image operators are filters in image data-flow diagrams
- Signal processing systems (DSP-based embedded systems)
  - The satellite radio
  - Video processing systems
  - Car control
  - Process 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

23.3 SADT

24

Structured Analysis and Design Technique (SADT)

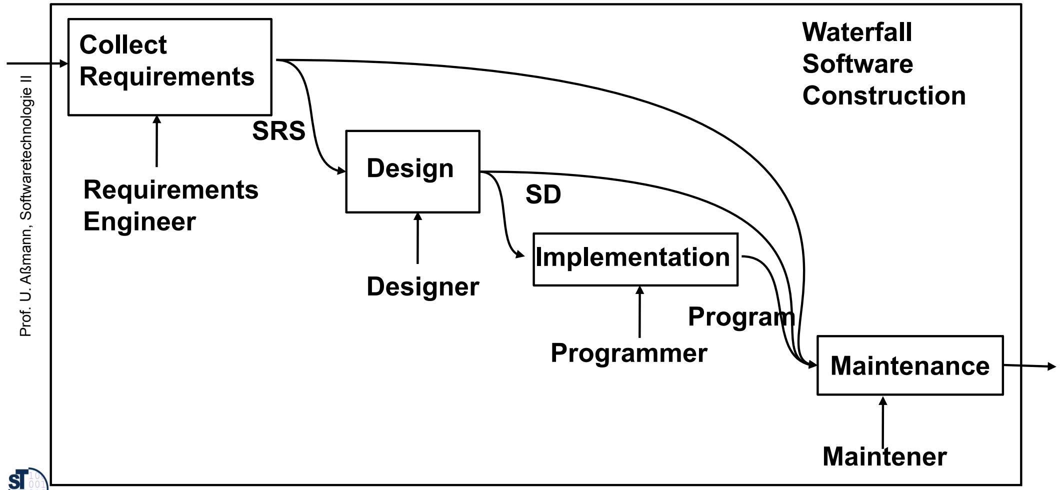
- SADT is a action- and data-flow-oriented method
- Reducible graphs with 2 main forms of diagrams:
  - Activity diagrams: Nodes are activities, edges are data flow (like DFD)
    - Data flow architectures result
  - Data diagrams: Nodes are data (stores) and edges are activities
  - Layout constraint: edges go always from left to right, top to bottom
- Companies used to have standardized forms, marked with author, date, version, name, etc..



Example: The Waterfall Software Model with Activity Diagram of SADT

Prof. U. Aßmann, Softwaretechnologie II

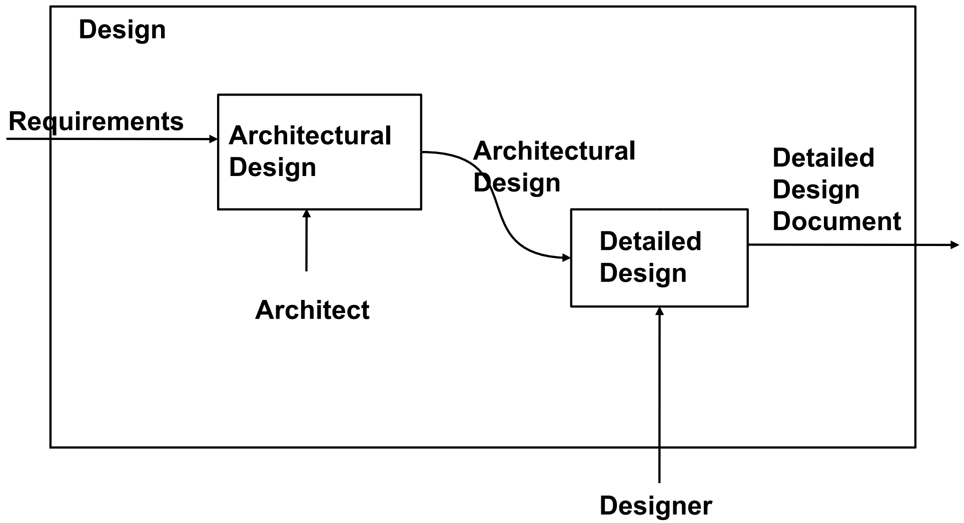
- Activity Diagrams SADT – Special DFD, with read direction left to right, top to bottom
- With designation of responsible



Point-Wise Refinement of Nodes

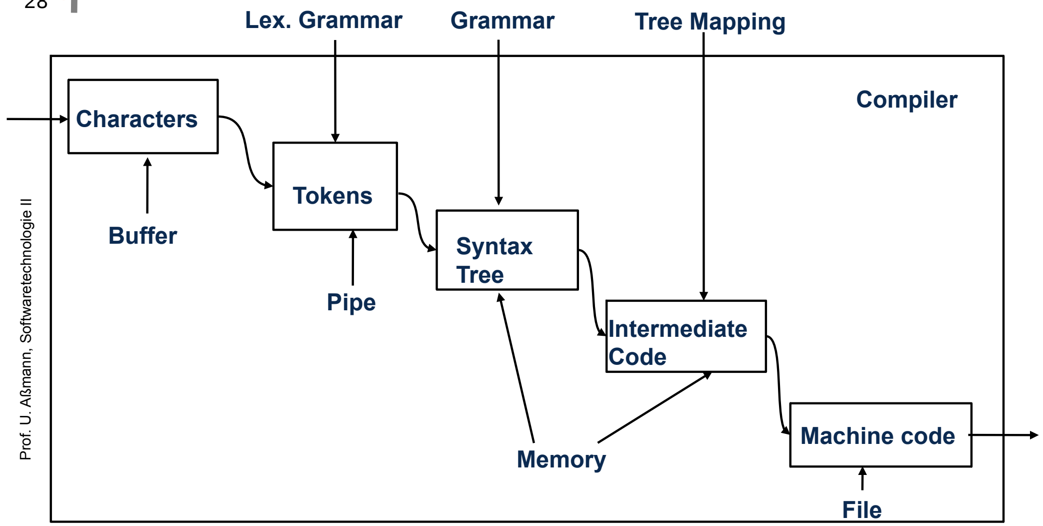
Prof. U. Aßmann, Softwaretechnologie II

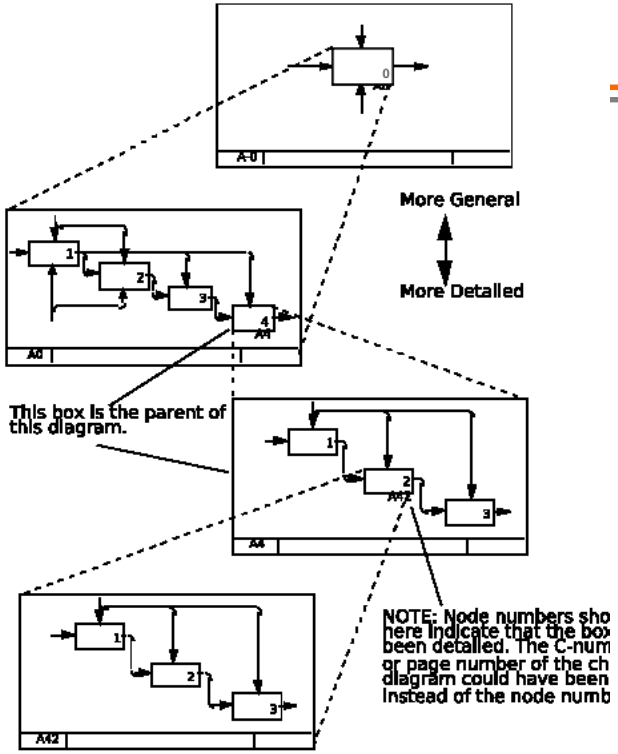
- Refinement must preserve input-output channels



SADT Data Diagram of a Copmiler

Prof. U. Aßmann, Softwaretechnologie II





Comparison SADT vs SA/SD

- SADT, SA/SD are system-oriented methods, known in other disciplines
  - *Action-oriented methods*
    - they only distinguish between actions (processes) and data
  - *Stream-oriented,* i.e., model streams of data flowing through the system
  - *System-oriented,* know the concept of a *subsystem*
- SA-DFDs are more flexible as SADT actitity diagrams, since the layout is not constrained
  - Function trees and DDs may be coupled with SADT

Why are the Data-Flow Methods SA and SADT Important?

- They lead to component-based systems (hierarchical systems)
  - Component-based systems are ubiquituous for many areas
  - Object-orientation is not needed everywhere
  - Other engineers use SADT also
- SA and SADT can easily describe parallel systems in a structured way
- SA and SADT are *stream-based*, i.e., for stream-based applications. When your context model has streams in its interfaces, SA and SADT might be applicable
- Use case actions can be refined similarly as SA and SADT actions!
- **Mashups** are web-based data-flow diagrams (see course Softwarewerkzeuge)

TECHNISCHE UNIVERSITÄT DRESDEN

23.4 Workflow Nets

Obligatory Readings

- W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of workflow task structures: A petri-net-based approach. Information Systems, 25(1): 43-69, 2000.
- Web portal "Petri Net World" http://www.informatik.uni-hamburg.de/TGI/PetriNets/

Relationship of PN and other Behavioral Models

- P.D. Bruza, Th. P. van der Weide. The Semantics of Data-Flow Diagrams. Int. Conf. on the Management of Data. 1989
  - http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.9398
- Matthias Weske. Business Process Modeling. Springer-Verlag.

Workflow Languages and Workflow Nets

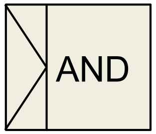
- Workflows are executable sequences of actions, sharing data from a repository, or communicating with streams.
  - Actions in workflows can be refined (as transitions in Petri Nets)
- Special languages exist, such as
  - YAWL Yet another workflow language
  - BPMN Business Process Modeling Notation
  - BPEL Business Process Execution Language
  - For checking of wellformedness constraints, they are reduced to Petri Nets
- **Workflow nets** are reducible workflows with single sources and single sinks

Workflow Nets

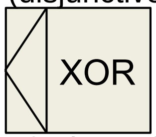
- **Workflow nets** extend DFD with control flow and synchronization (by transitions)
  - They avoid global repositories and global state
  - They provide richer operators (AND, XOR, OR), inhibitor arcs, and synchronization protocols
- Workflow nets can be compiled to Petri Nets (polynomially reducible)
- All workflow nets become single-entry/single-exit, so that only reducible nets can be specified

Complex Operators in Workflow Nets: YAWL Join and Split Operators

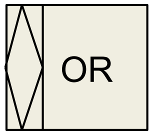
- All icoming places are ready (conjunctive input)



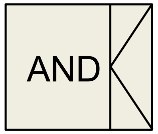
- One out of n incoming places are ready (disjunctive input)



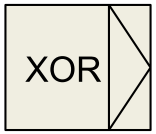
- Some out of n incoming places are ready (selective input)



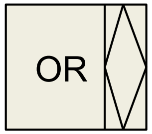
- All outgoing places are filled (conjunctive output)



- One out of n outgoing places are filled (disjunctive output)

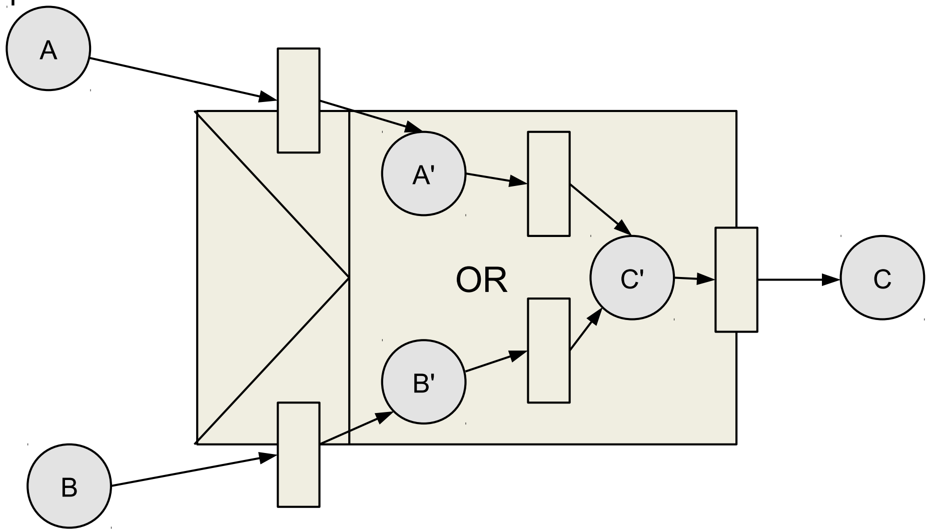


- Some out of n outgoing places are filled (selective output)



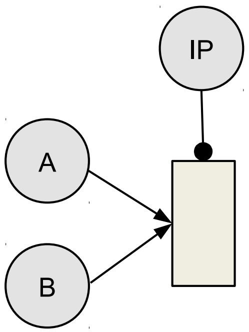
Reduction Semantics of Operators

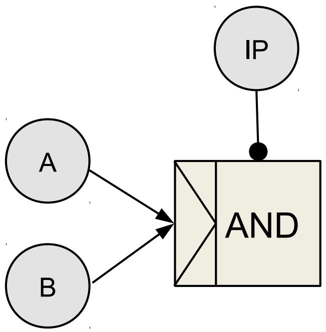
- Complex operators refine to special pages with multiple transition ports



Inhibitor Arcs

- An **inhibitor arc** prevents the firing of an operator or transition





- Transition only fires if inhibiting place IP is *not ready*.
- AND-Operator only fires if IP is *not ready*.

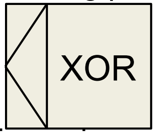
TECHNISCHE UNIVERSITÄT DRESDEN

23.4.2 Open Operators in Workflow Nets

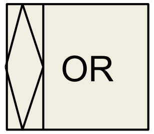
Open Operators

- OR and XOR operators are **open**, i.e., can be extended by incoming resp. outgoing edges *without violating the contract of the operator*

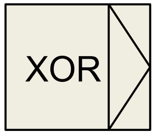
- **Open joins:**
- One out of n incoming places are ready



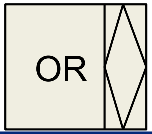
- Some out of n incoming places are ready



- **Open Splits:**
- One out of n outgoing places are filled



- Some out of n outgoing places are filled



Workflows with Harmless Extensions

- If edges are added to an open operator, they *enrich the semantics* of the net, but do not destroy or change it (monotonic extension)
- Therefire, adding an edge retains the contracts, i.e., basic assumptions, of a workflow net.

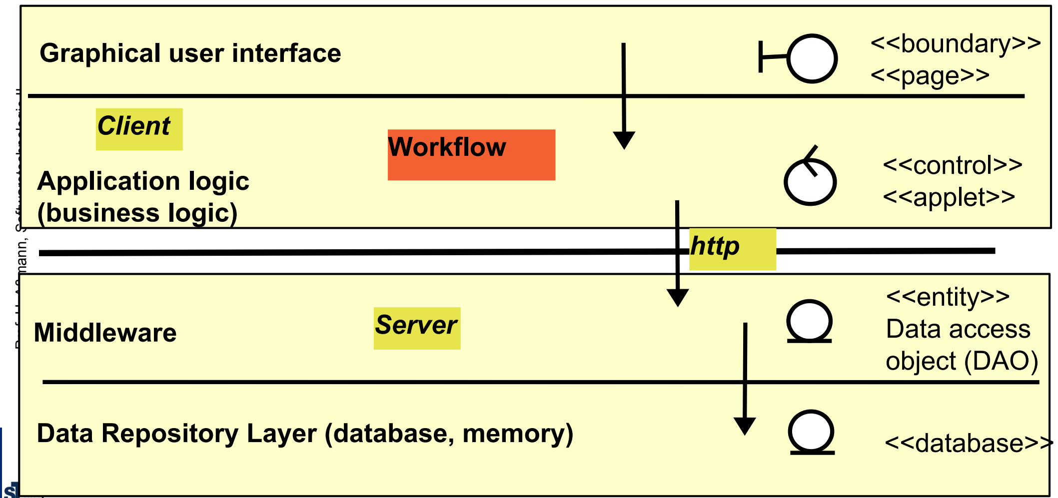
Extending the open operators of a core workflow does not change the contracts of it.

TECHNISCHE UNIVERSITÄT DRESDEN

23.4.3 Applications of Workflow Ne

Rpt. from ST-1: 4-Tier Web System (Thick Client)

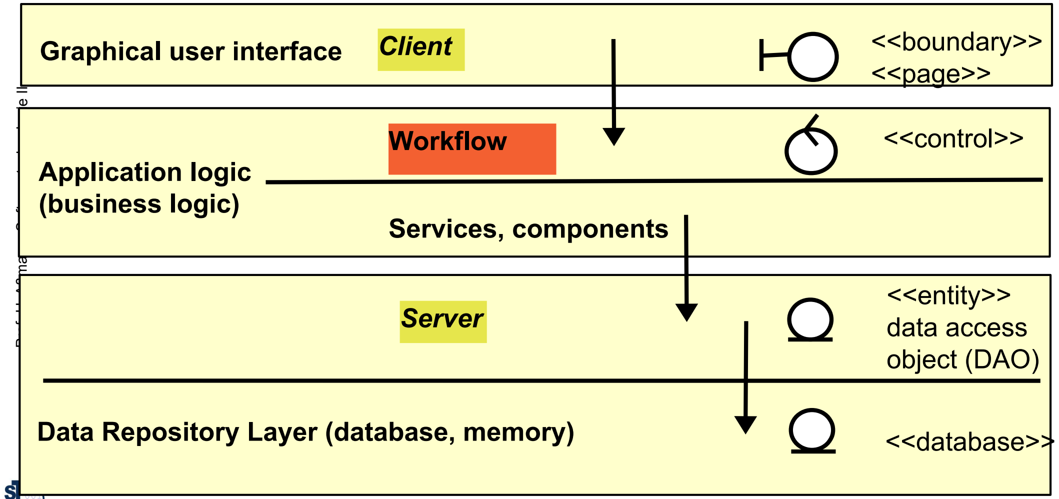
- Workflow specifications are for the application logic layer



Rpt. from ST-1: 5-Tier with Workflow Language

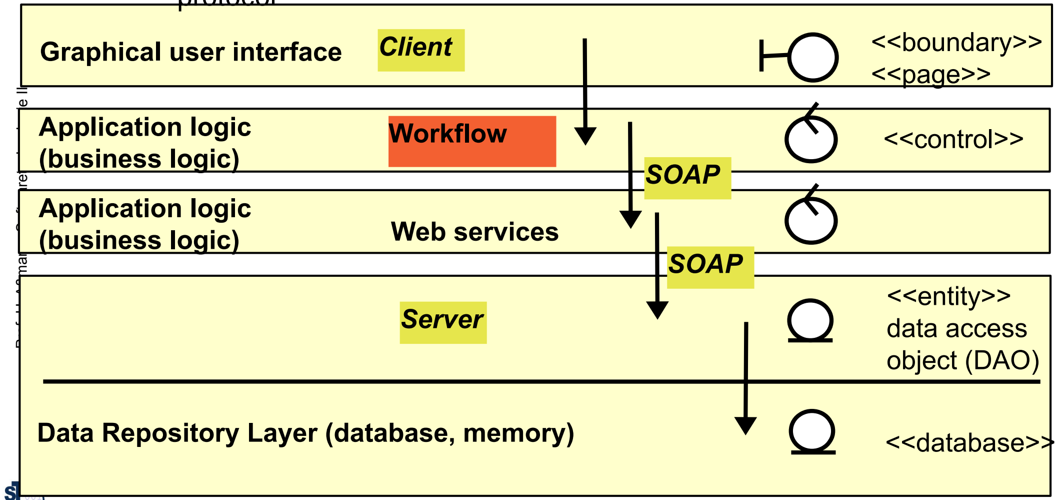
Workflow languages (BPMN, BPEL, WF Nets) describe the top-level of the application architecture

- Services and components are called by the workflow



Rpt. from ST-1: 5-Tier with Workflow Language and Web Services

- Workflow languages (BPMN, BPEL, WF Nets) describe the top-level of the application architecture
  - Services and components are called by the workflow via SOAP protocol



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
- Workflow languages extend DFD with control flow and can be compiled to Petri nets

The End