23. Action-Oriented Design Methods

1

Prof. Dr. U. Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

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

WS 14/15 - 10.01.15

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

Lecturer: Dr. Sebastian Götz

Obligatory Reading

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

TECHNISCHE UNIVERSITÄT DRESDEN

23.1 Action-Oriented Design

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

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)

- Why should you still learn SA/SD ?
- TIOBE index (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html)

Jan 2015	Jan 2014	Change	Programming Language	Ratings	Change
1	1		C	16.703%	-1.24%
2	2		Java	15.528%	-1.00%
3	3		Objective-C	6.953%	-4.14%
4	4		C++	6.705%	-0.86%
5	5		C#	5.045%	-0.80%
6	6		PHP	3.784%	-0.82%
7	9	↑	JavaScript	3.274%	+1.70%
8	8		Python	2.613%	+0.24%
9	13	↑↑	Perl	2.256%	+1.33%
10	17	↑↑	PL/SQL	2.014%	+1.38%
11	15	↑↑	MATLAB	1.390%	+0.62%
12	26	↑↑	ABAP	1.273%	+0.80%
13	27	↑↑	COBOL	1.267%	+0.81%
14	24	↑↑	Assembly	1.171%	+0.68%

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]
- 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 flows through a DFD
  - Pseudocode (minispecs) describes central algorithms (state-based)
  - Decision Table and Trees describe 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
- SA and SADT are important for *embedded systems* because resulting systems are parallel and hierarchic

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

- On the highest abstraction level, on the **context diagram**:
  - **Elaboration**: Define interfaces of system by a top-level action tree
  - **Elaboration**: Identify the in-out streams most up in the 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

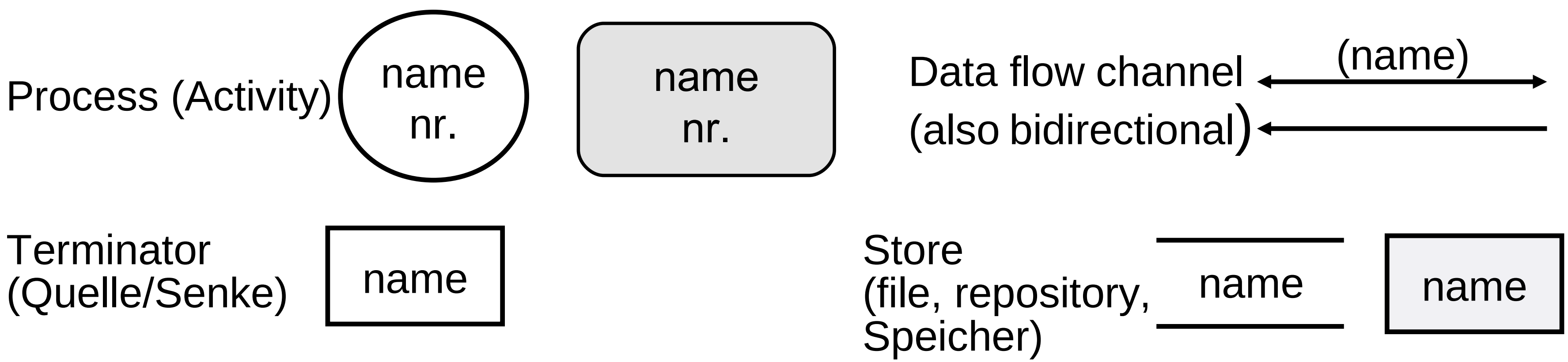
Data-Flow Diagrams (Datenflussdiagramme, DFD)

- DFD are a special form of Petri nets (see Chapter on PN)
- 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

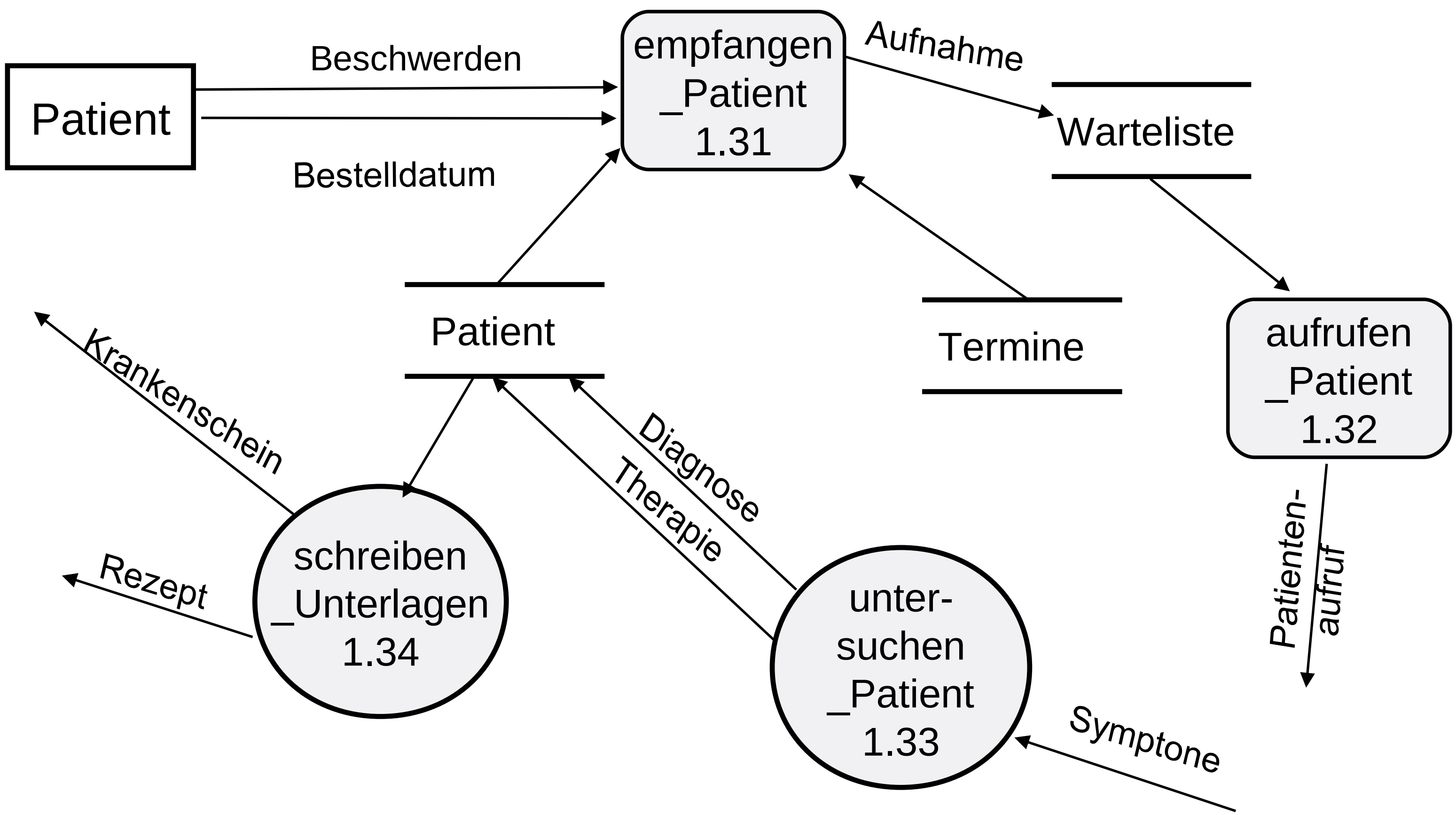
- Reducible (hierarchic) nets of processes linked by channels (streams, pipes)
  - **Context diagram:** top-level, with terminators
  - Parent diagrams, in which processes are point-wise refined
  - Child diagrams are refined processes
  - Refinement can be syntactic or semantic
- Data dictionary contains 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):



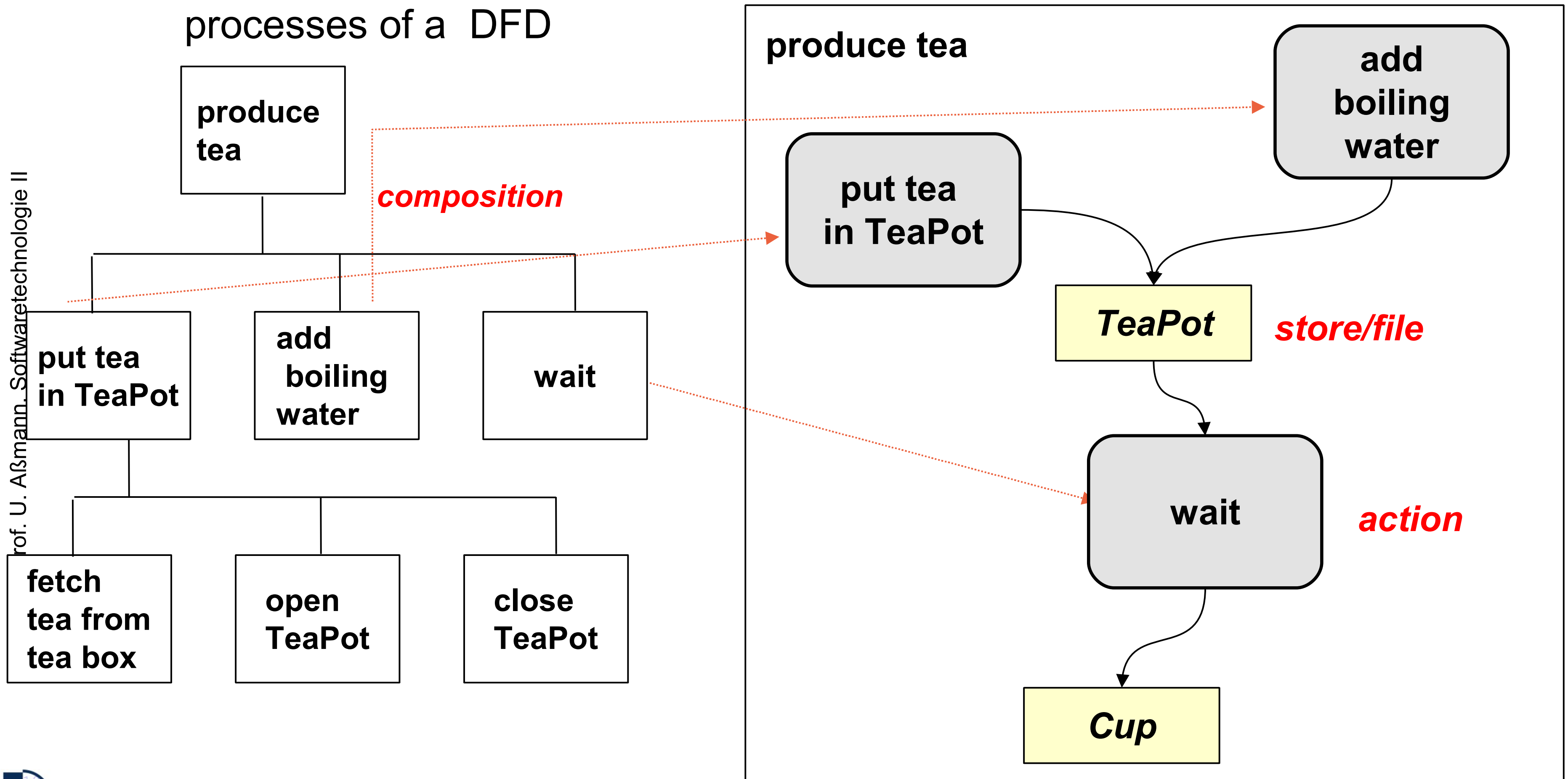
Ex.: DFD "treat_Patient"

- UML uses ovals for activities; SA uses circles



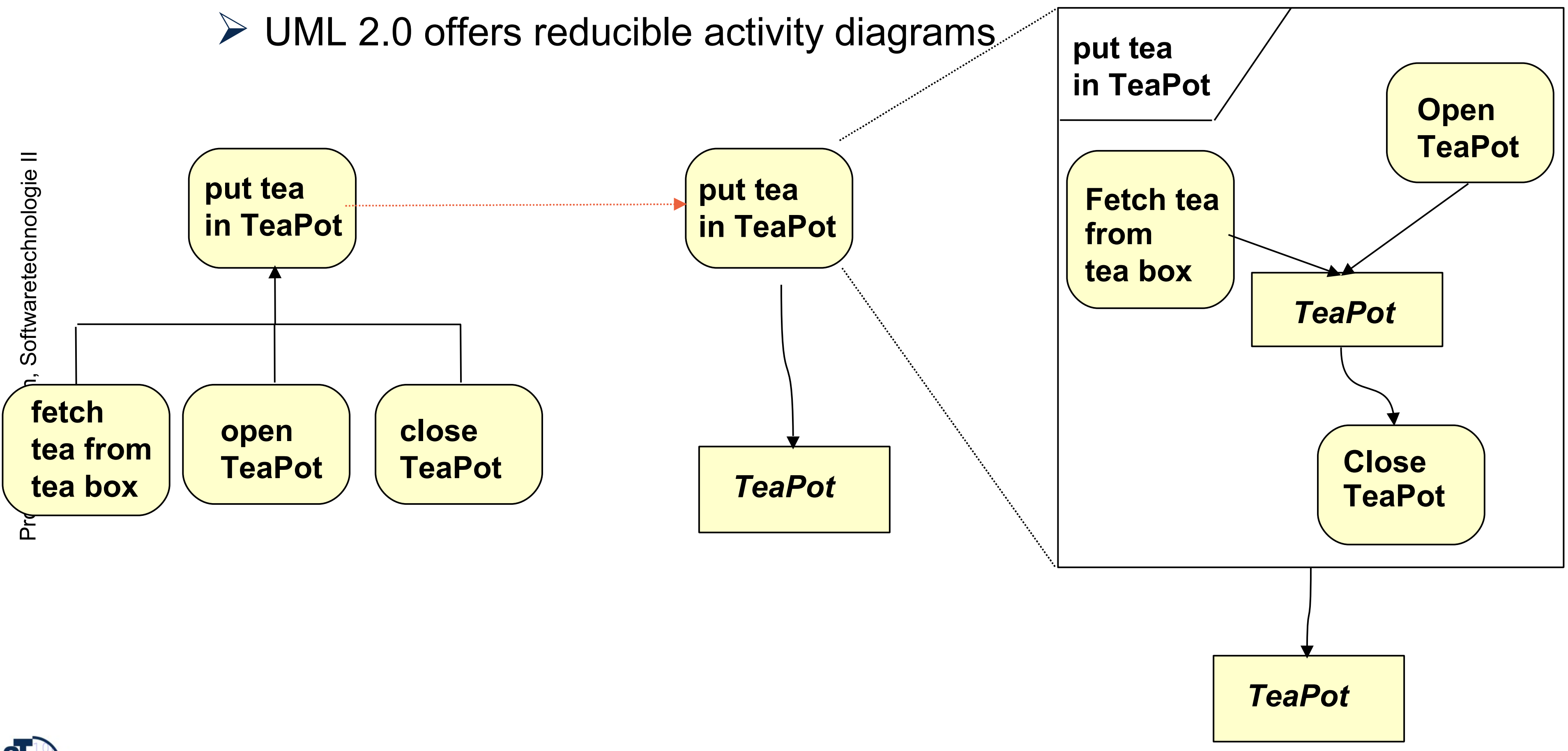
Action Trees and DFDs

- Action trees can be derived from function trees
- 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
  - 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	I or [|]	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).

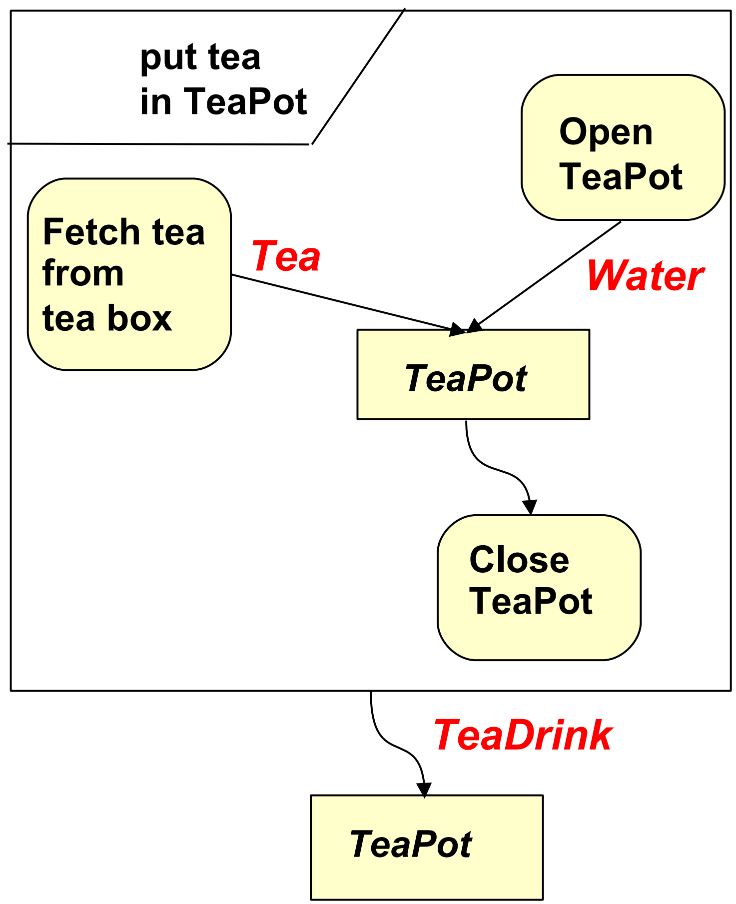
Example Grammar in Data Dictionary

- 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

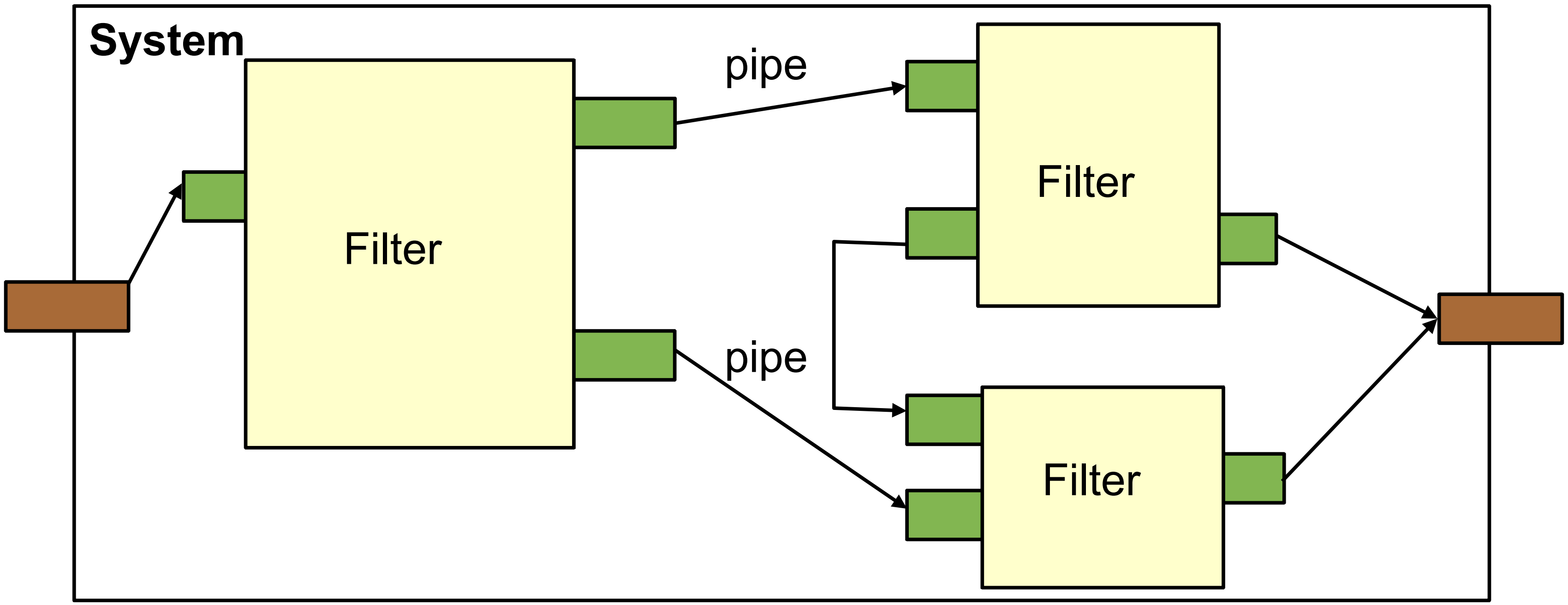
- SA focusses on actions (parallel activities, processes), not functions
  - Describe the data-flow through a system
  - Describe stream-based systems with pipe-and-filter architectures
- Actions are parallel processes
  - SA and SADT can easily describe parallel systems
- Function trees are interpreted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

Implementation Hints

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

Result: Data-Flow-Based Architectural Style

- 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)**



Application Areas are Manifold

- 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 Workflow Nets

Readings

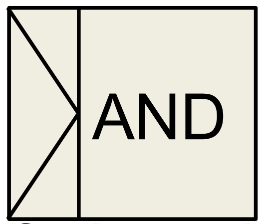
- W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of workflow task structures: A petri-net-based approach. Information Systems, 25(1): 43-69, 2000.
- P.D. Bruza, Th. P. van der Weide. The Semantics of Data-Flow Diagrams. Int. Conf. on the Management of Data. 1989
  - http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.9398
- Matthias Weske. Business Process Modeling. Springer-Verlag.

Workflow Nets

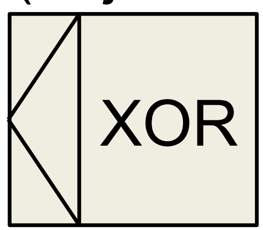
- In general, **workflows** are executable sequences of actions, *sharing data* from several repositories or communicating with streams.
- **Workflow nets** are reducible with single sources and single sinks (**single-entry/single-exit)**, so that only reducible nets can be specified
  - They extend DFD with control flow and synchronization
  - 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
- Further, specialized workflow 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 PN

Complex Transition Operators in Workflow Nets: Join and Split Operators of YAWL

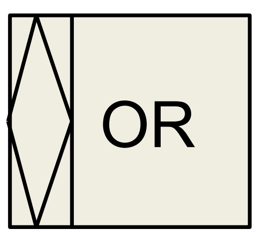
- All incoming places are ready (conjunctive input, AND-join)



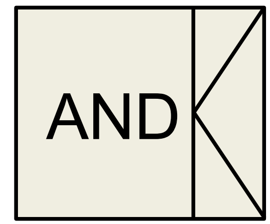
- One out of n incoming places are ready (disjunctive input)



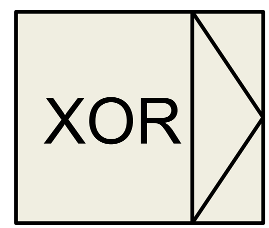
- Some out of n incoming places are ready (selective input)



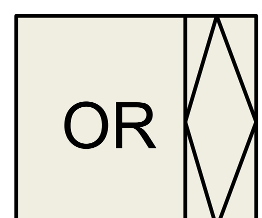
- All outgoing places are filled (conjunctive output, AND-split)



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

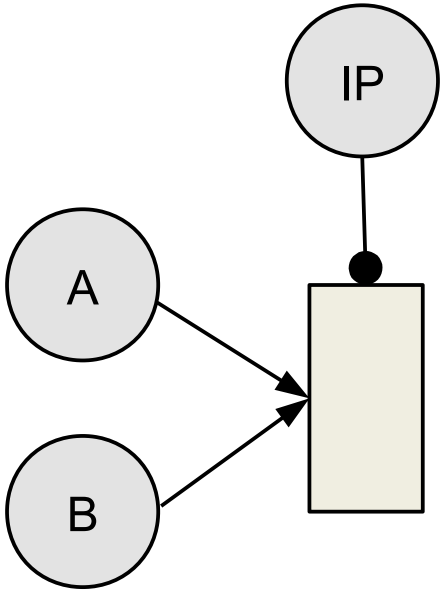


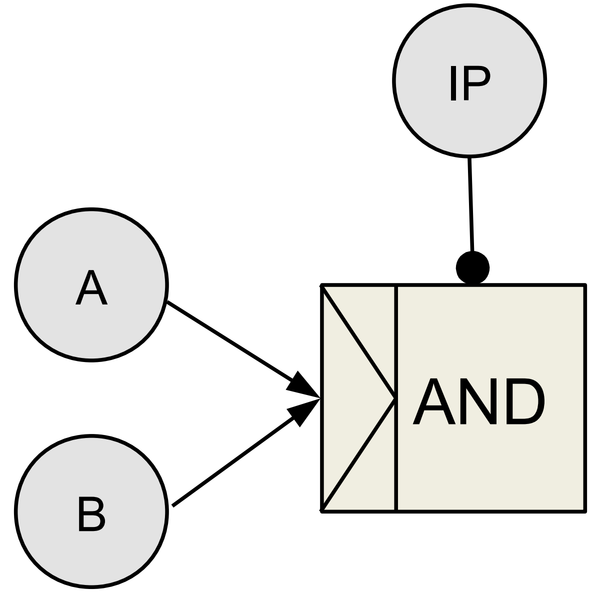
- Some out of n outgoing places are filled (selective output)



Inhibitor Arcs

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

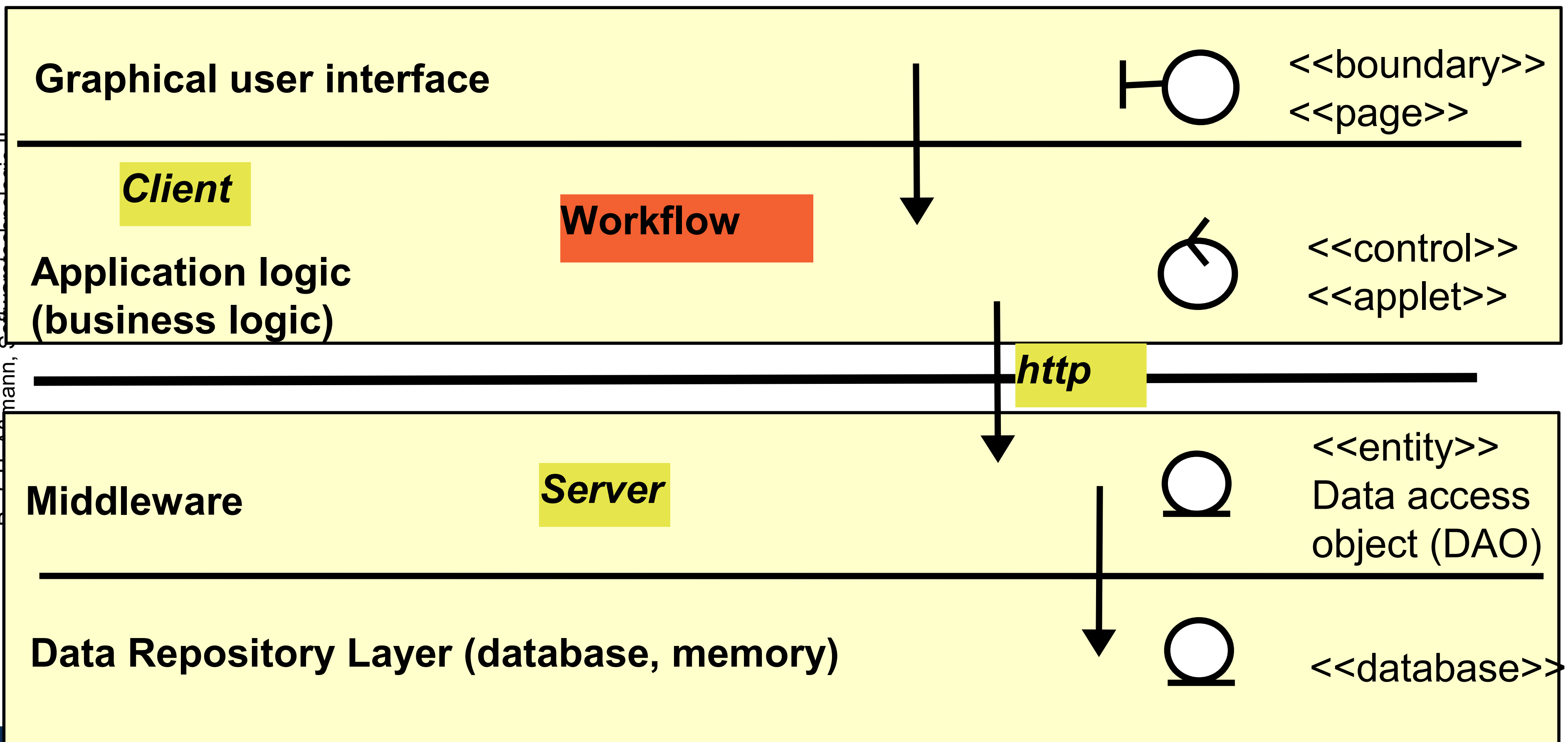




- Transition only fires if inhibiting place IP is *not ready*.
- AND-Operator only fires if IP is *not ready*.

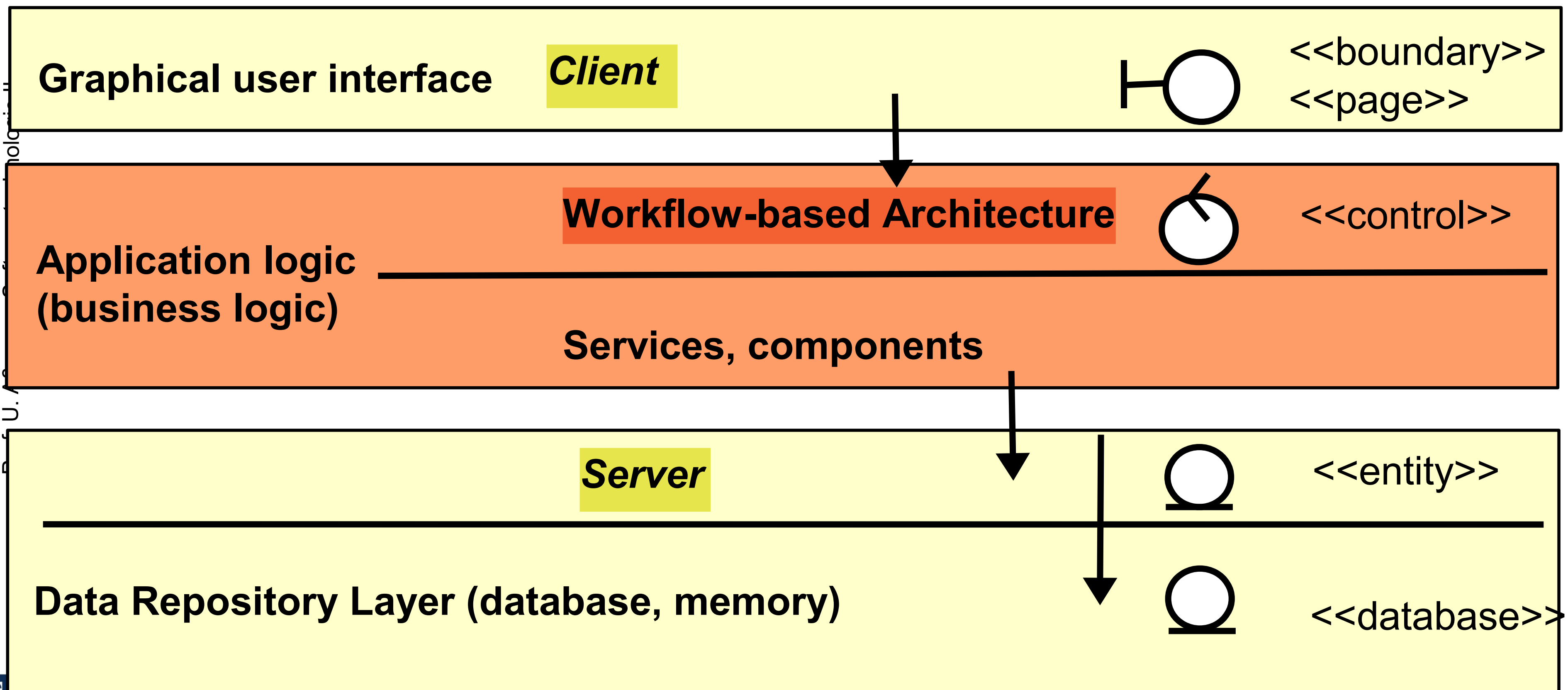
4-Tier Web System (Thick Client)

- Workflow specifications are for the application logic layer



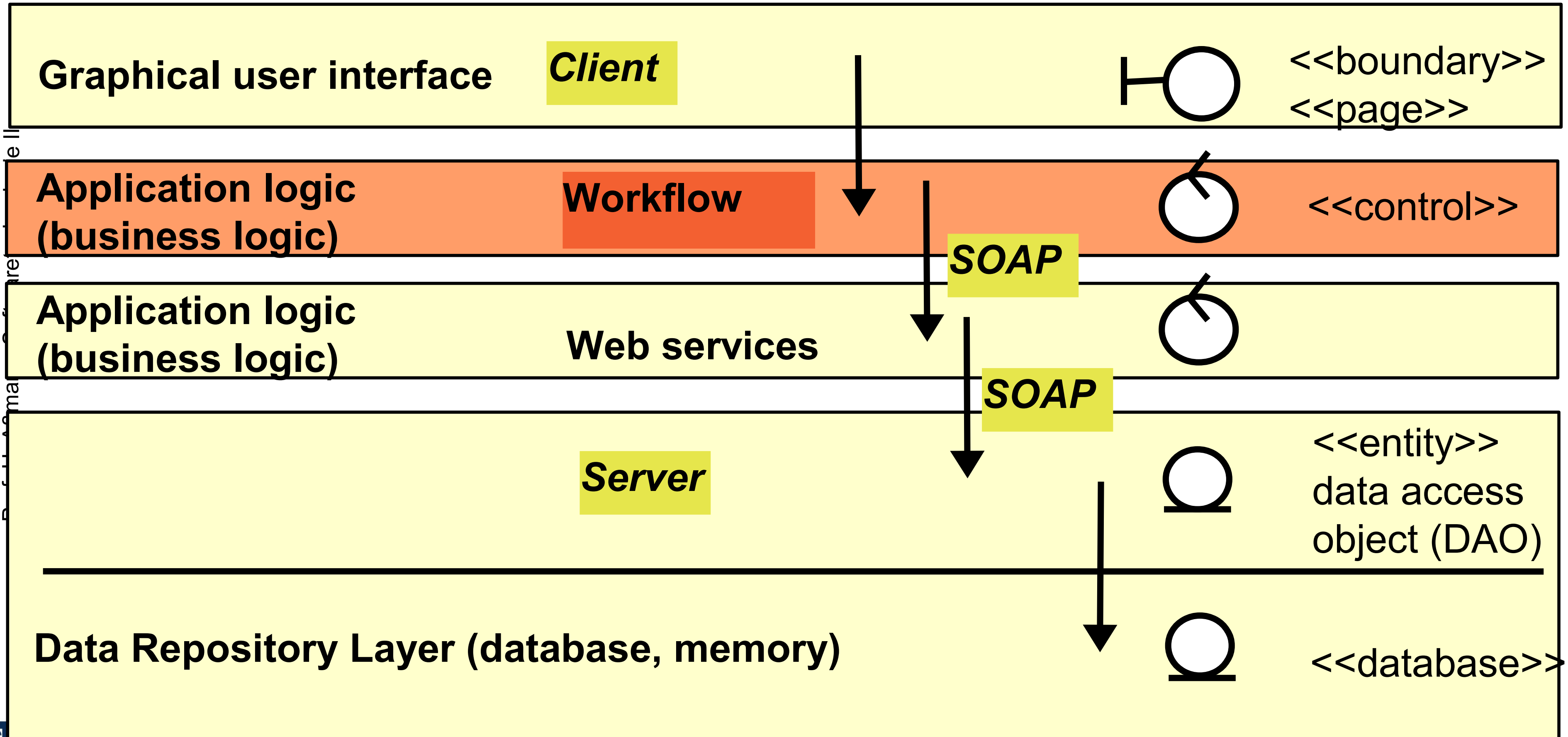
5-Tier with Workflow Language

- In a **Workflow Architectural Style**, a workflow in a language specifies the application architecture
  - All services and underlying components are called by the workflow
  - The workflow is executed by a special workflow engine



5-Tier with Workflow Language and Web Services

- Workflows describe the top-level application architecture
  - Services and components are called by the workflow



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, action-oriented design methods are appropriate
  - Action-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
- In a Workflow-Based Architecture, all services are described by architectural workflows

Appendix
Possible Exam Questions

- Which advantages has the reducibility of the SA DFD specification?
- Show a refinement of a DFD, starting from a given function tree
- Which relation has a DFD and a CPN?
- How would you implement a DFD specification?
- What is the unique characterization of a workflow-based architecture?
- How to extend a workflow net?