ATPN 2004 Tutorial

What's new in UML 2?

Reiko Heckel
Jan Hendrik Hausmann

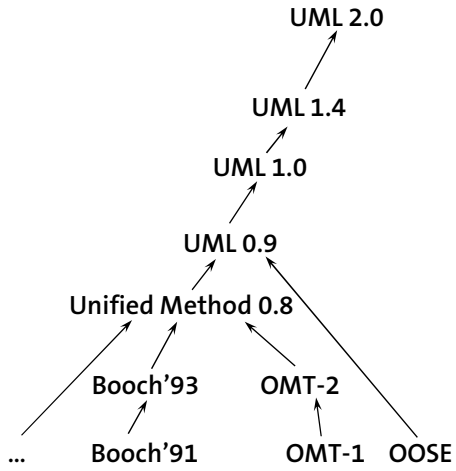
Universität Paderborn, Germany

Outline

- new ideas: MDD and MDA
  - motivations and terminology
  - one notation, different flavours
  - what's new
- new notations: UML 2.0 diagrams
- open problems: what's left for version 3
- new activity diagrams: zooming in (2nd part)

Evolving Motivations

- industry standard for *model-driven* application integration
- industry standard (OMG) for OO modelling notations
- unification of OO modelling notations
- from structured to OO modelling methods



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

3

MDD and MDA
Some Terminology, due to Bran Selic

Model-driven Development: approach to software development in which the focus and primary artifacts are models instead of programs

- abstraction from implementation details
  - reduce complexity
  - improve portability
- automation of
  - code generation
  - verification
  - execution

Model-driven Architecture: OMG initiative to define a set of standards for MDD (strategically, UML is the new CORBA)

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

4

Views on Model-Driven Development

1. Models as sketches
   - human-oriented
   - informal, incomplete

→ ***Martin Fowler's UML mode, see*** *www.martinfowler.com/bliki/UmlMode.html*

2. Models as blueprints
   - solution-oriented
   - formal, consistent

Model-Driven Architecture

- platform-independent
- platform-specific
- implementation

3. Models as programs
   - technology-oriented
   - complete, executable

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

5

Take frozen fish fingers out of package and fry for 5 – 7 minutes on all sides.

Do we need informal models?

- Humans, based on domain knowledge, are able to deal with
  - incompleteness
  - ambiguity
  - contradictions
- Too much formalism is like too much bureaucracy

ısmann
21.6.2004

6

What's New ?
User level (superstructure) and Internal (infrastructure)

- architectural modelling
  - decomposition of classifiers using connectors and ports
  - better integration of structure and behaviour
- models as programs
  - (action semantics for) new activity diagrams
  - MSC-like structuring and control flow for sequence diagrams
- extensibility beyond profiles
- compliance levels
- certification

Not in this tutorial:

- alignment of meta model with MOF (common core)
- general cleanup

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

7

Example: Workflow Modeling

Online Shopping with Max

Looking for the most recent *Harry Potter*, we employ a *Shopping Agent* to

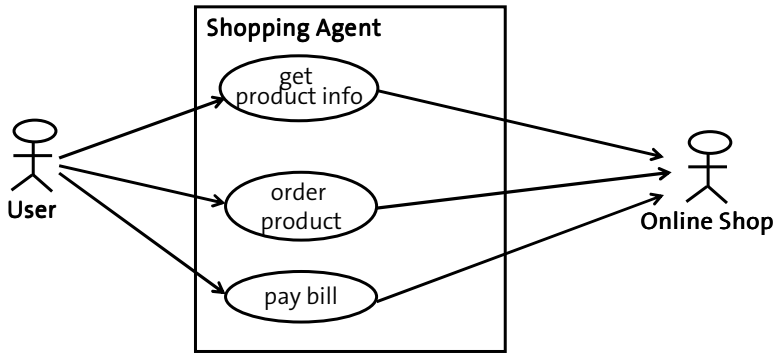
- obtain product info
- choose an offer
- order and pay

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

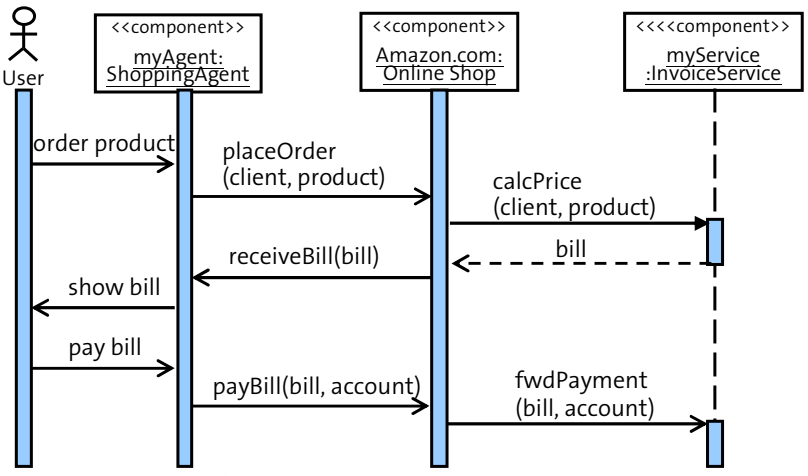
Heckel / Hausmann
ATPN Tutorial – 21.6.2004

8

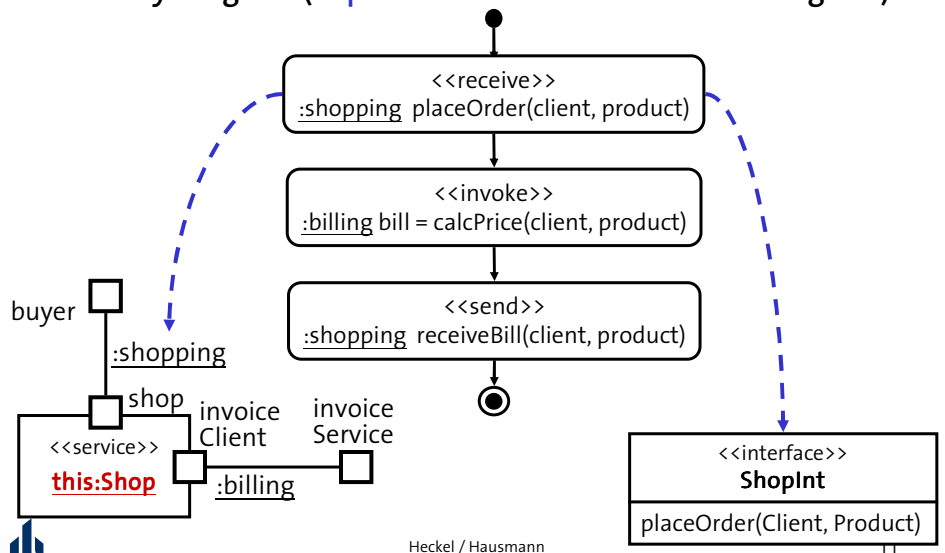
Model as Sketch: Use Case Diagram



Models as Blueprint: Sequence Diagram



Model as Program: Executable Business Process as Activity Diagram (depends on Structure and Class Diagram)



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

11

New Notations: UML 2.0 Diagrams

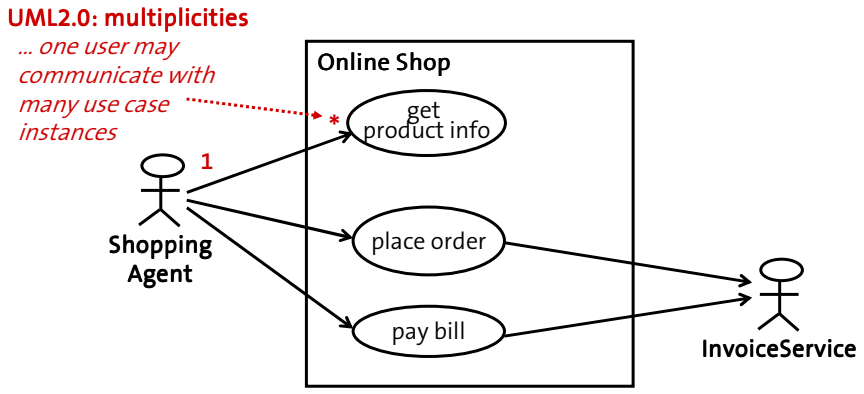
- structural diagrams
  - class
  - object
  - package
  - component
  - deployment
  - *composite structure*
- behavioural diagrams
  - use case
  - *activity (in 2nd part)*
  - interaction
    - *interaction overview*
    - *timing*
    - communication (← collaboration)
    - *sequence*
  - state machine

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

12

Use Case Diagram



- ***actors*** (user roles) and ***use cases*** (complete functional units) of the ***system,*** as well as their communication relations
- structure requirements, often in combination with informal descriptions

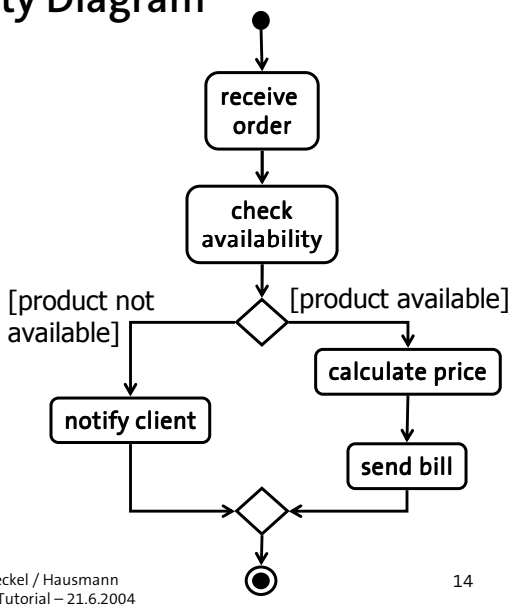
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

13

Activity Diagram

- model behaviour in terms of (sequential or concurrent) flows of actions
- used for to describe business processes and control flow



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

14

Sequence Diagrams

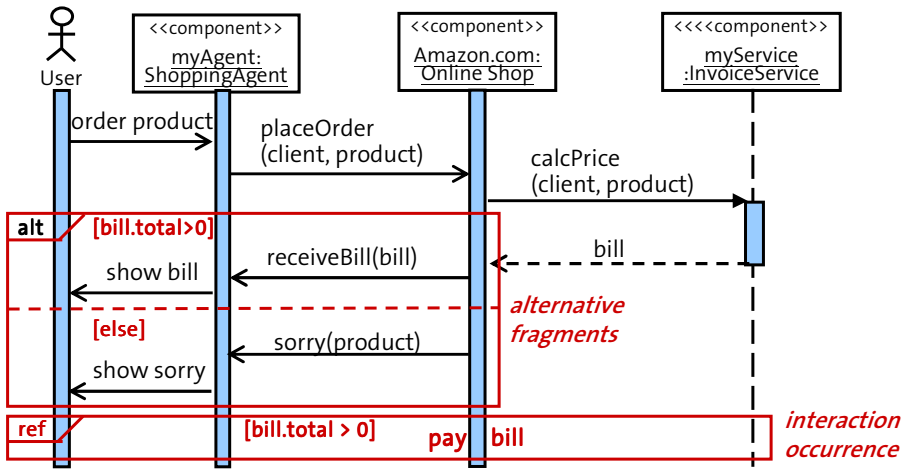
UML 1.x:

- represent interaction between users and components (objects) within the system
- mainly used to visualize scenarios at the instance level

UML 2.0:

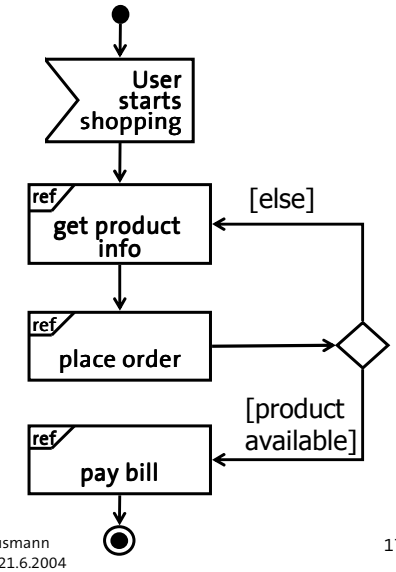
- shift of focus to description of mandatory behaviour
- control flow
  - guarded choice
  - iteration
  - calls to other diagrams

Sequence Diagram: Interaction Scenario for *order product* and *pay bill*



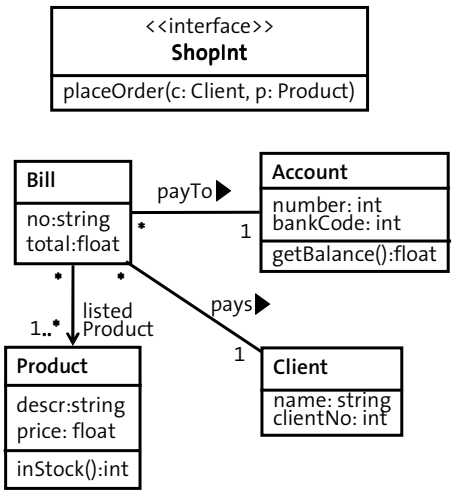
UML 2.0: Interaction Overview Diagram

- a variant of the activity diagram
- represents high-level control flow between different interactions (cf. high-level MSCs)



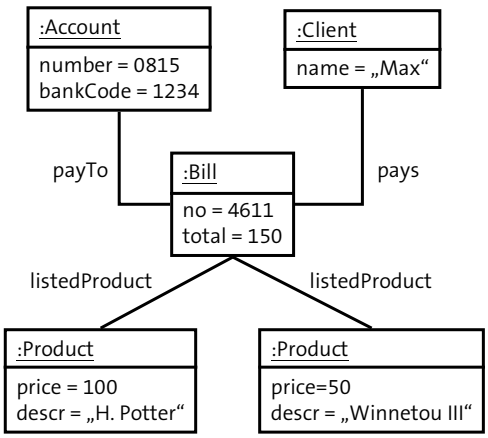
Class Diagram

- defines the types of objects in the system, with their attributes, method signatures, and associations
- used to model real world concepts or implementation classes



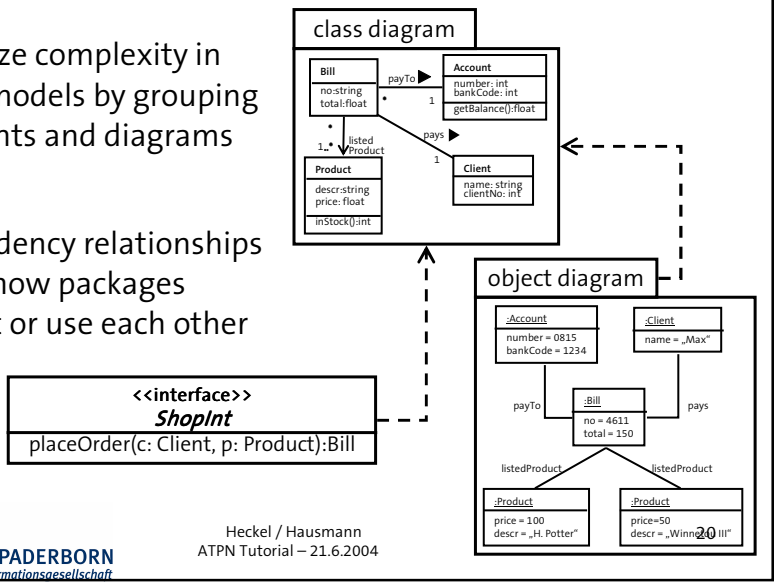
Object Diagram

- a graph of objects with attribute values (instances of classes) and links (instances of associations)
- represents a situation in the real world or the data state of a system



Package Diagram

- organize complexity in large models by grouping elements and diagrams
- dependency relationships show how packages import or use each other



Component and Composite Structure Diagram

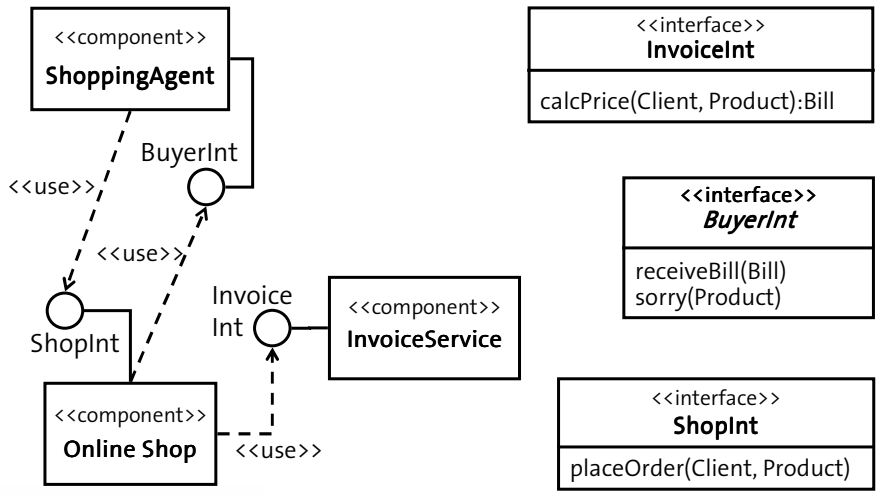
UML 1.x

- components
- interfaces
- implementation and use relations

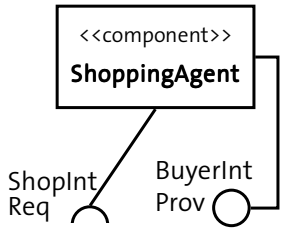
UML 2.0

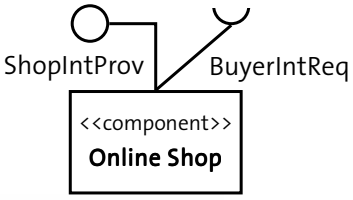
- separation of *required* and *provided* interfaces
- *port:* a class specifying communication endpoints
  - can have instances
  - can implement / use multiple interfaces
- *connector:* an instance of an association between ports representing an ongoing communication
- nesting of instances (roles)

UML 1.x: Components and Interfaces, Implementation and Usage



UML 2.0: Separation Required and Provided Interfaces





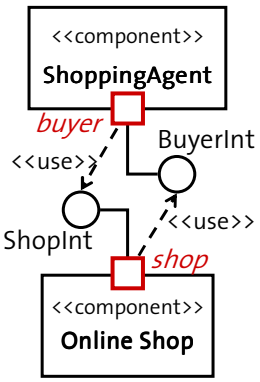
- decouple components by specifying required services explicitly
- motivation
  - dynamic binding
  - service-oriented architectures

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

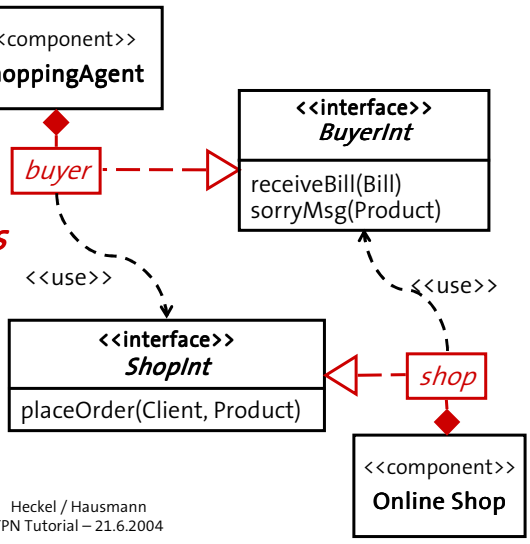
Heckel / Hausmann
ATPN Tutorial – 21.6.2004

23

UML 2.0: Ports as Communication Endpoints

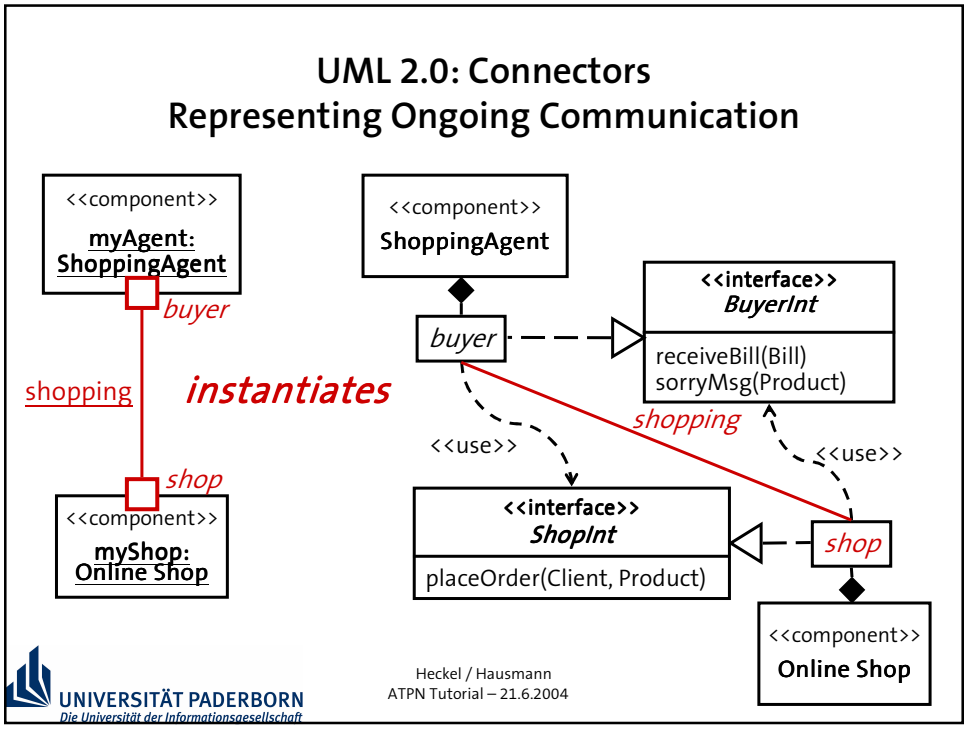


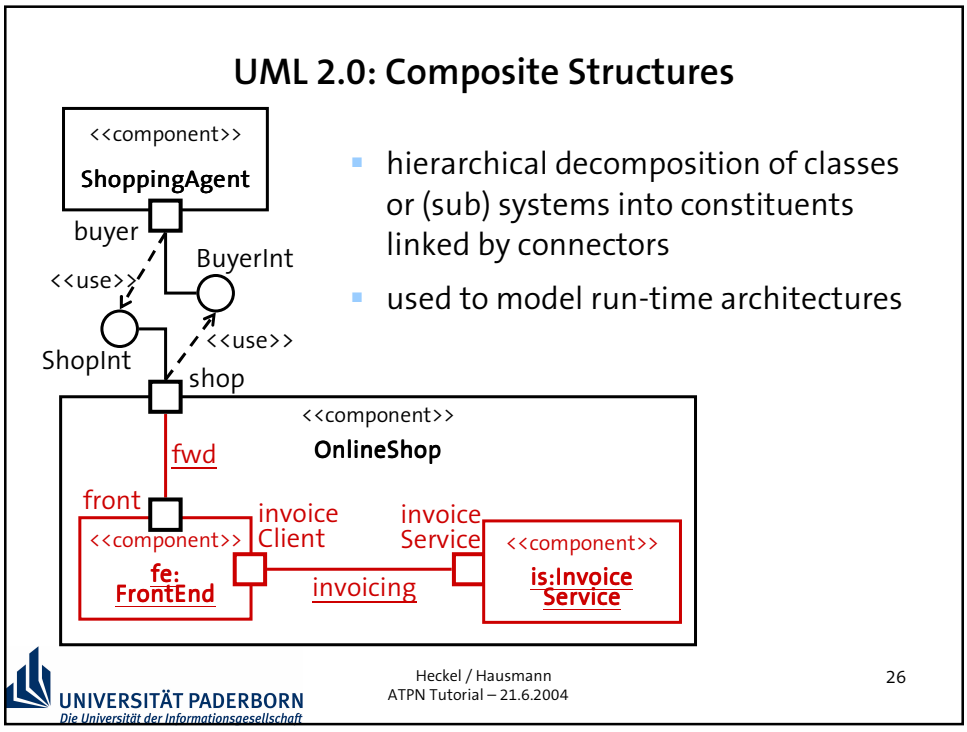
means



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

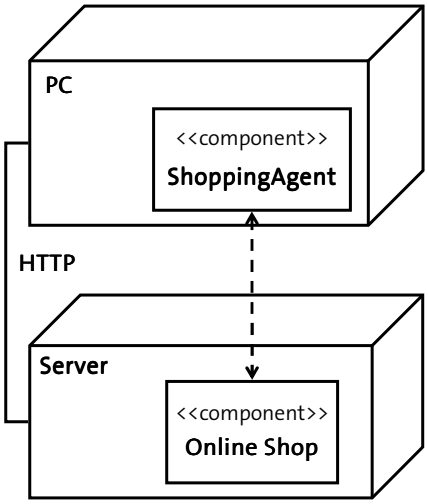
Heckel / Hausmann
ATPN Tutorial – 21.6.2004


UML 2.0: Connectors
Representing Ongoing Communication
<<component>>
myAgent:
ShoppingAgent
buyer
shopping
instantiates
shop
<<component>>
myShop:
Online Shop
<<component>>
ShoppingAgent
buyer
<<interface>>
BuyerInt
receiveBill(Bill)
sorryMsg(Product)
shopping
<<use>>
<<use>>
<<interface>>
ShopInt
placeOrder(Client, Product)
shop
<<component>>
Online Shop
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Heckel / Hausmann
ATPN Tutorial – 21.6.2004


UML 2.0: Composite Structures
<<component>>
ShoppingAgent
buyer
BuyerInt
<<use>>
ShopInt
<<use>>
shop
<<component>>
OnlineShop
fwd
front
<<component>>
fe:
FrontEnd
invoice
Client
invoicing
invoice
Service
<<component>>
is:Invoice
Service
hierarchical decomposition of classes or (sub) systems into constituents linked by connectors
used to model run-time architectures
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Heckel / Hausmann
ATPN Tutorial – 21.6.2004
26

Deployment Diagram

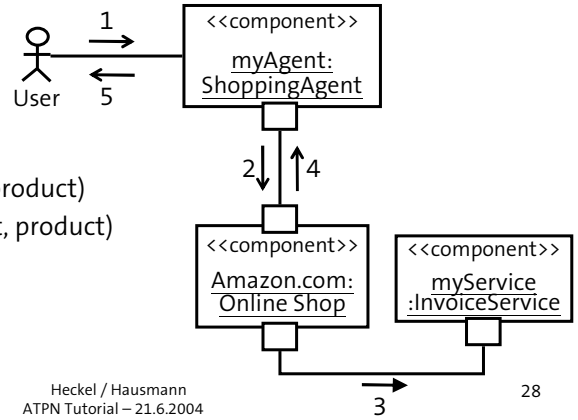
- define physical system architecture in terms of nodes (machines) and connections
- specify location (deployment) of components



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Communication Diagrams (UML 1.x Collaboration Diagrams)

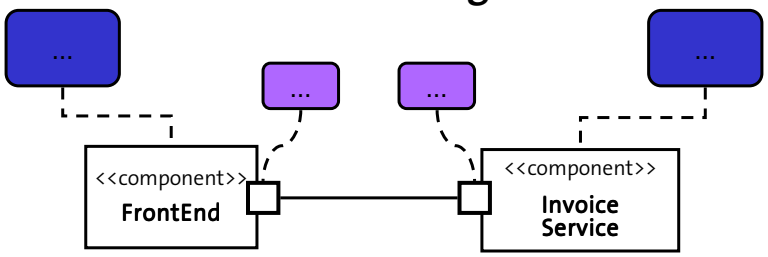
- notational variant of sequence diagrams, integrating interaction and structure



1: order product
2: placeOrder(client, product)
3: bill=calcPrice (client, product)
4: receiveBill(bill)
5: show bill
...

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

State Chart Diagrams



UML 1.x:

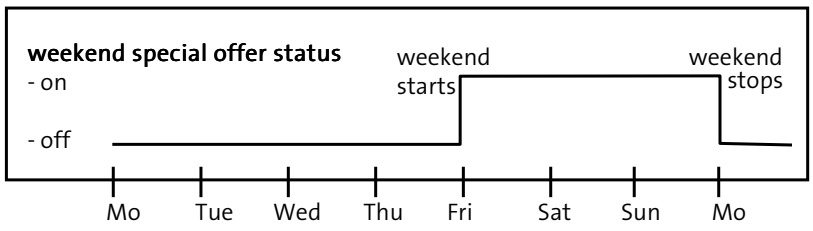
- statecharts for classes, modelling object life cycle

UML 2.0:

- statecharts for components modelling reactive behaviour
- statecharts for ports modelling protocols
- statechart inheritance and incremental modelling

Timing Diagrams

- show state changes and events in their relationship to clock times
- used for defining clock-driven behaviour (mostly) of embedded real-time components



Extension Mechanisms

UML 1.x:

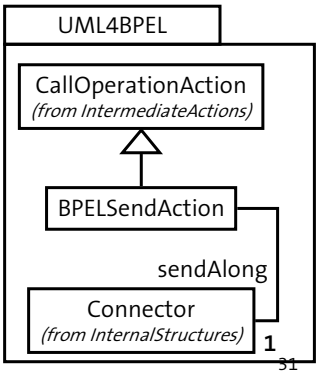
- individual <<stereotypes>>
- profiles as model-level encoding of meta model extensions by means of
  - <<stereotypes>>
  - tagged values
  - constraints

"tagged value refers to the name of a connector"

<<send>>
:shopping receiveBill(client, product)

UML 2.0:

- true meta model extension, encapsulated in packages



Compliance and Certification

UML 2.0 compliance for tools

- 38 compliance points for tools: mandatory UML kernel plus 37 optional
- four options for each point
  - no compliance
  - partial
  - compliant
  - interchange
- summing up to three compliance levels
  - basic: 1 – 10 (→ class, activity, interaction, use case)
  - inter: 1 – 25 (→ statecharts, profiles, components, deployment)
  - complete: all (→ actions and advanced features)

UML 2.0 certification for humans

- three tests, two hours each: beginning, intermediate, advanced

Open Problems

- **model transformation:** refinement of, and code generation from, models
- **consistency:** syntactic and semantic relations between different diagrams
- **semantic customization:** how to specify the behaviour of extensions

Classical solutions (grammars, algebraic methods, ...) are not readily applicable to the UML due to its

- graphical syntax
- semi-formal nature

SegraVis Research Training Network

"Syntactic and Semantic Integration of Visual Modeling Techniques"

Flexible grants* (3 months – 1 year) for *young researchers
(< 36 of age, pre or post doc) at

Paderborn	Leiden
Antwerp	London
Barcelona	Milano
Berlin	Darmstadt
Bremen	Pisa
Canterbury	Roma

Contact: reiko@upb.de or see www.segravis.org.

UML 2.0 Activity Diagram Party

Jan Hendrik Hausmann
Universität Paderborn
hausmann@upb.de

(!) Warnings (!)

- The UML specification is
  - huge (Activity Diagrams/Actions > 150 pages)
  - not yet fixed (>500 open issues, ~100 relating to ADs)
  - a unification of different influences (ADs=WfM+OO+PNs+Programming Languages)
  - agreed upon in a committee (expect some compromises)
  - lacking a formal semantics (yet)
- *I'm here to tell, not sell!*

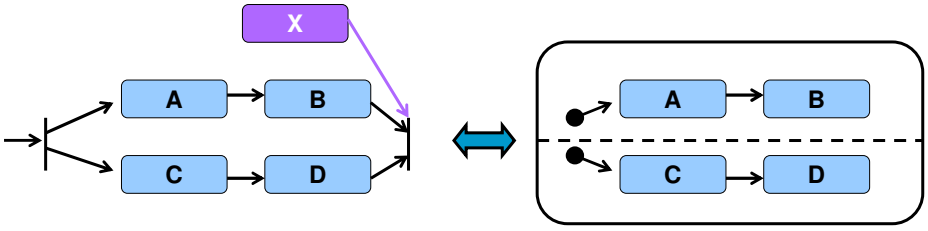
Outline

- I- History and Purpose of Activity Diagrams + Quick Tour
- II- Core Elements
- III- Semantics and Pitfalls
- IV- Advanced constructs (brief overview)

The role of Activity Diagrams in the UML

- In UML 1.x
  - Data flow techniques were not object-oriented
  - added at the last minute as an „extension“ of statecharts (=> semantic problems)
  - lack of support for process and program modeling
    - resource assignment
    - exceptions
    - ...
  - no tool support

Coupling ADs with Statecharts- a good idea?



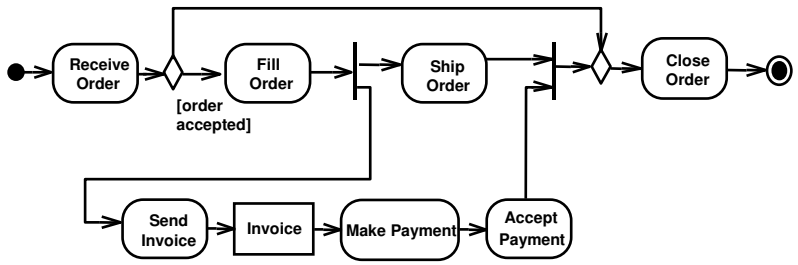
Run to completion semantics of Statecharts implied

- A||C,B||D (no true concurrency)
- Only well-nested forks/joins

Activity Diagrams in UML 2.0

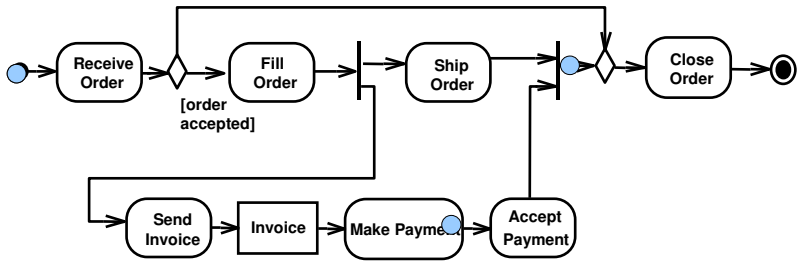
- One of three main behavior diagrams
- *Activity modeling emphasizes the sequence and conditions for coordinating other behaviors, rather than which classifiers own the behavior*
- Activity diagrams can be used for a number of things:
  - use case scenarios
  - workflow specifications (real world -> executable)
  - method implementation

Quick Tour



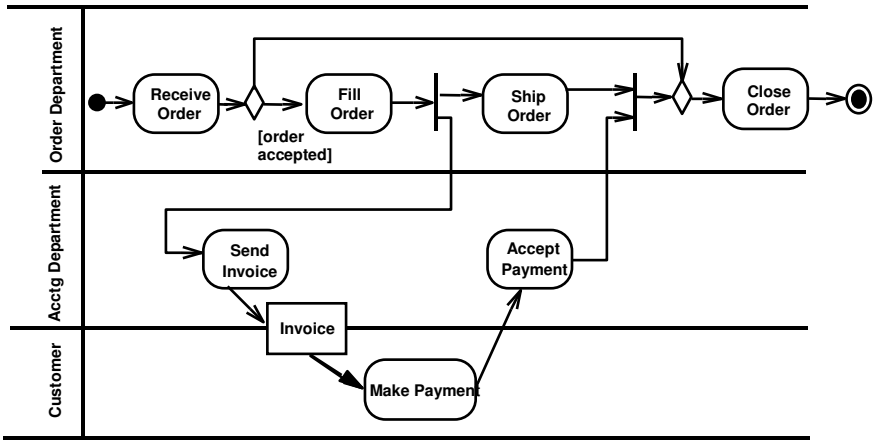
- Activity Diagram = Directed Graph
- Nodes
  - Executable Nodes
  - Control Nodes
  - Object Nodes
- Edges indicate flow (of data or control)

Quick Tour (2)



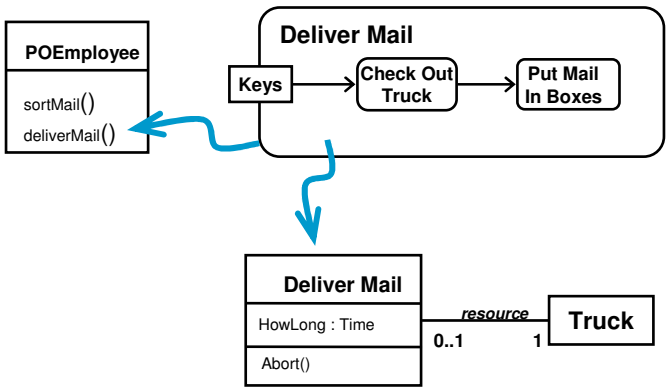
- Semantics (roughly) Token flow

Quick Tour (3)



- Assigning ressources to tasks

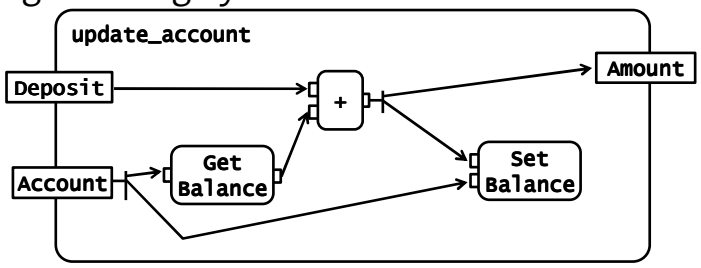
Quick Tour (3)



- Activities can be assigned to objects
- Activities can be stand-alone behavior

Quick Tour (4)

- Programming by Activities



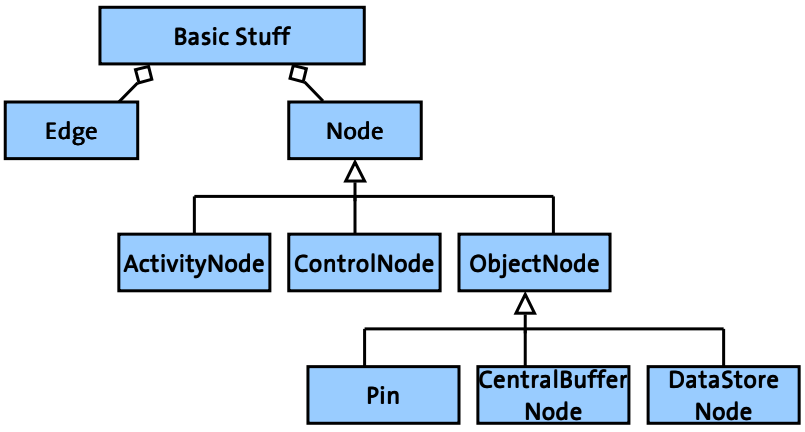

```
amount function update_account (dep:Deposit, acc:Account)
        { Amount new_bal = acc.getBalance()+dep;
          acc.setBalance(new_bal);
          return new_bal;}
```

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

45

Part I – Basic Stuff

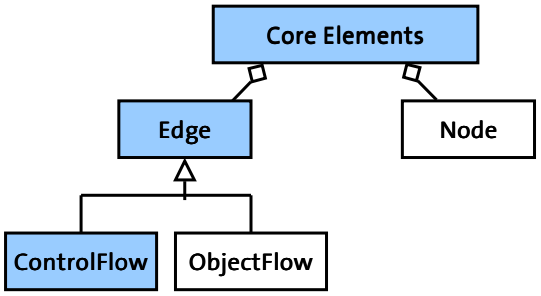


UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

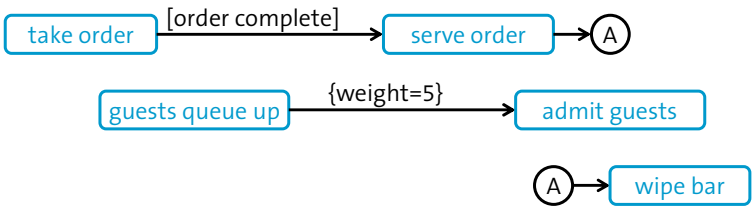
Heckel / Hausmann
ATPN Tutorial – 21.6.2004

46

Part II – Core Elements

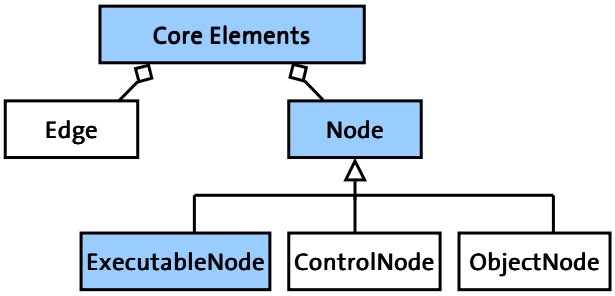


Edges

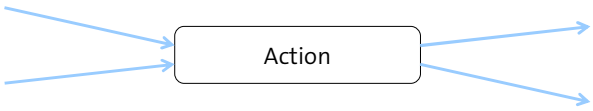


- Edges guide the token flow
- Guards impose conditions on the traversing tokens
- Weights determine number of tokens that traverse simultaneously (default:1)
- Connectors avoid long and tangled lines

Part I – Core Elements

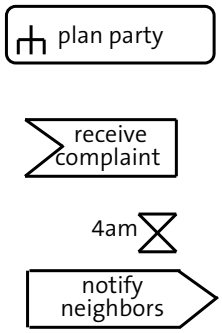


Executable Nodes



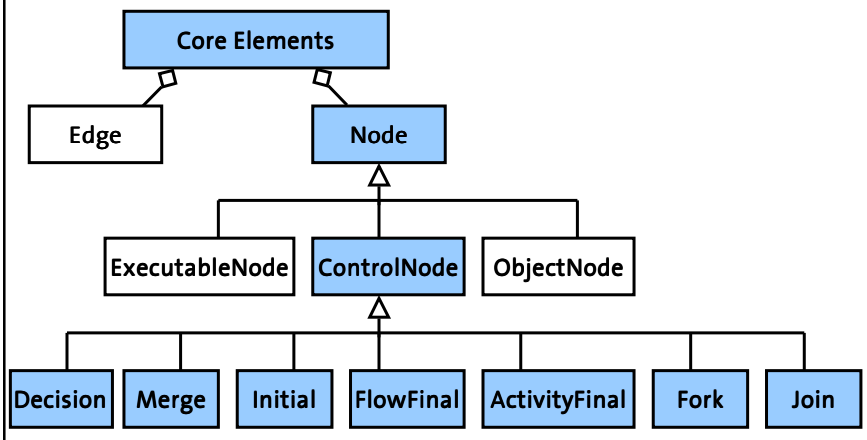
- Will „do“ something
- Entails an Action that describes the execution
- Dynamic semantics:
  - *All incoming edges must offer tokens for the execution to start*
  - *All offered tokens are consumed at once*
  - *The detailed semantic of execution of an action and definition of completion depends on the particular subclass of action.*
  - *When completed, an action execution offers object tokens on all its output pins and control tokens on all its outgoing control edges (implicit fork), and it terminates.*
- Prevention of (local) deadlocking by partial token consumption
  => Inspired by PN Transitions

Some Actions



- CallAction: will invoke some other activity (rake notation optional)
- AcceptEventAction: waits for the occurrence of an event
- Special notation for time events
- SendSignalAction

Part II – Core Elements



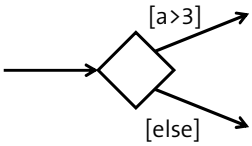
Control Nodes

- Control nodes guide the token flow between other nodes
- No behavior of their own
- Tokens cannot be buffered at Control Nodes (exceptions for initial and fork node)

⇒ Flow-to-completion semantics: Tokens will traverse paths of edges and control nodes from end to end in one go.

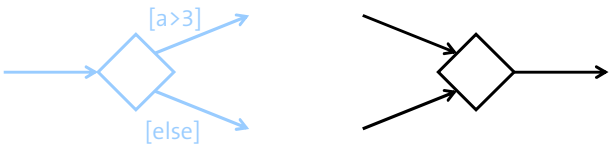
⇒ Intention: Token Competition with "fairness": first completely open path will get the token, no buffering of tokens in "dead ends"

Decision Nodes



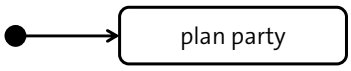
- multiple outgoing edges with logically disjoint guards
- predefined guard [else]
- *each arriving token can traverse only one outgoing edge*
- inspired by XOR-branching in WfMS

Merge Nodes



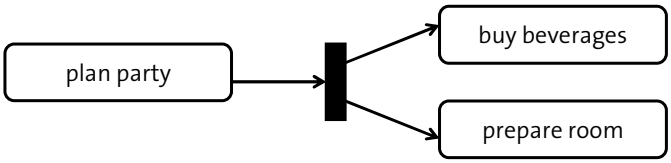
- multiple ingoing edges
- used to merge branches without synchronisation
- *tokens offered on the incoming edges are offered on the outgoing edge*
- (!) OR-merge, 2 inputs will produce two outputs!
- (!) Decisions/Merges can be used independently (no well-nested structure required!)

Initial node

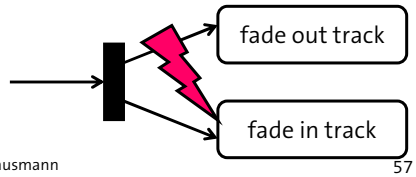


- determines the start of the control flow
- can be replaced by parameters
- *a control token is placed on each initial node when the activity is starts (i.e. by invocation)*
- if outgoing edges are blocked by guards, tokens are allowed to rest at an initial node

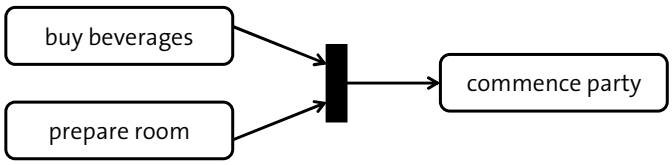
Fork nodes



- initiates independent (concurrent) branches
- *Tokens arriving at a fork are duplicated across the outgoing edges*
- (!) No true parallelism!

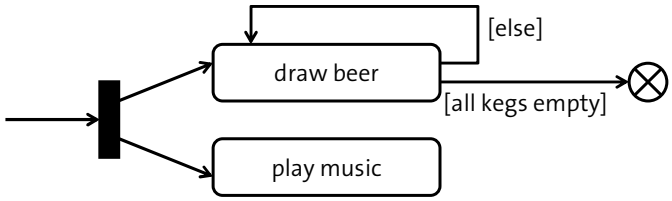


Join nodes



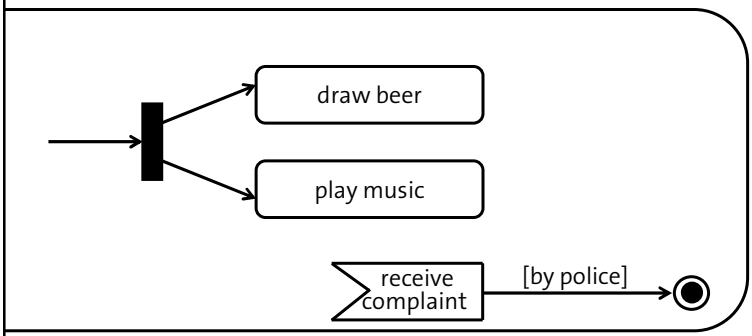
- Joining of concurrent branches
- *If there is a token offered on all incoming edges, then tokens are offered to the outgoing edge (if all incoming tokens are control tokens, then one control token will be offered on the outgoing edge)*
- AND-Join
- (!) Forks/Joins can be used independently (no well-nested structure required!)

FlowFinalNodes



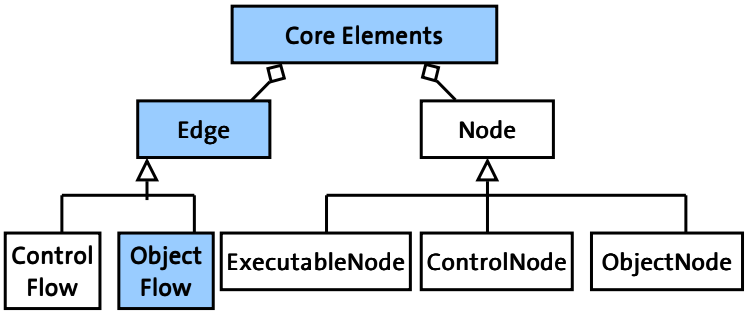
- FlowFinal nodes will end one flow of the net
- All arriving tokens will be deleted
- Other flows might continue

Activity Final Nodes



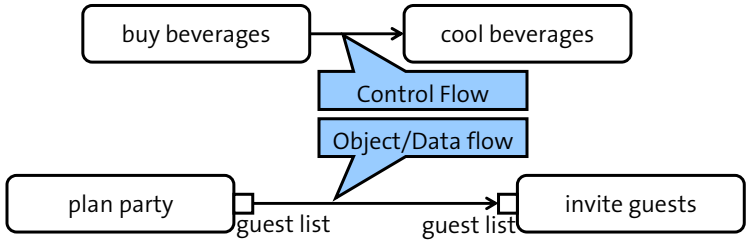
- *A token arriving at an activity final node aborts all flows in the containing activity*
- (!) Handle with care if using re-entrant behavior

Part II – Core Elements

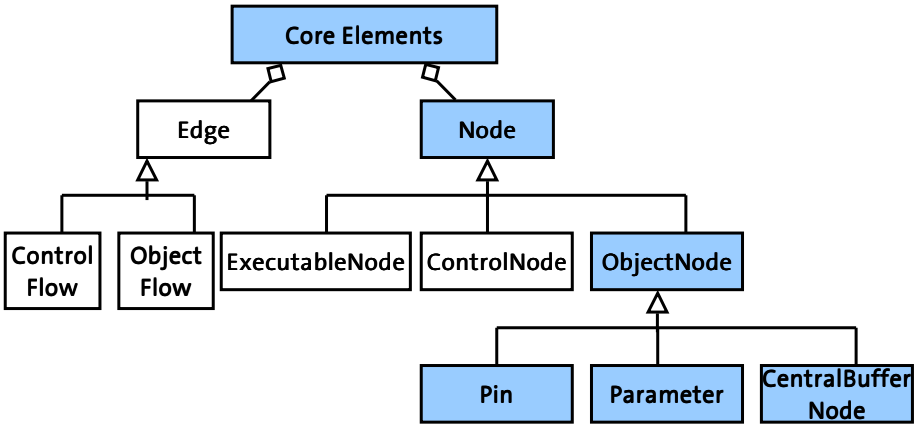


Object Flows

- Up to now: tokens=control
- With object nodes: tokens= (pointers to) objects/data
- Every edge attached to at least one object node is an object flow



Part II – Core Elements



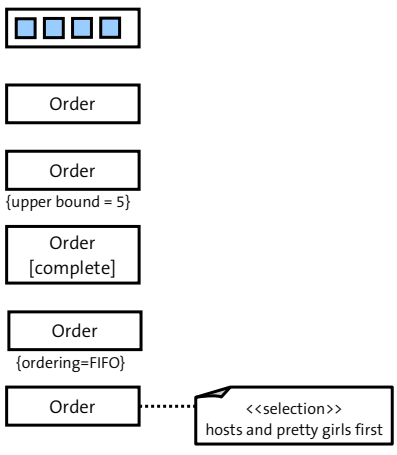
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

63

Object nodes

- Can buffer tokens
- Can have type
- Can have capacity
- Can require state
- Can have ordering
- Can specify selection (user defined ordering)



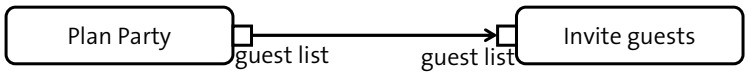
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Heckel / Hausmann
ATPN Tutorial – 21.6.2004

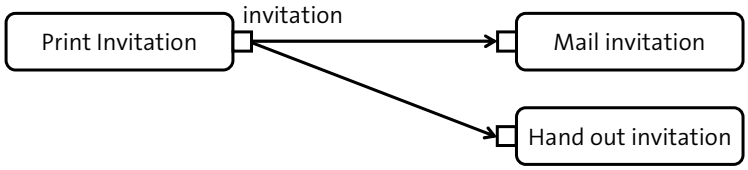
64

Pins

- *A pin is an object node for inputs and outputs to actions*

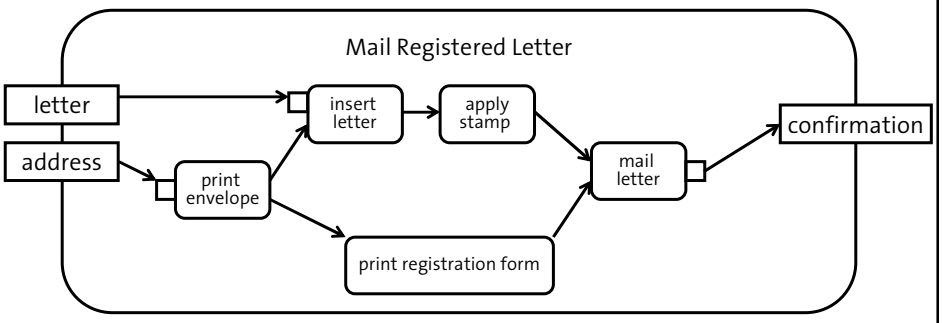


Output Pins – Token Competition



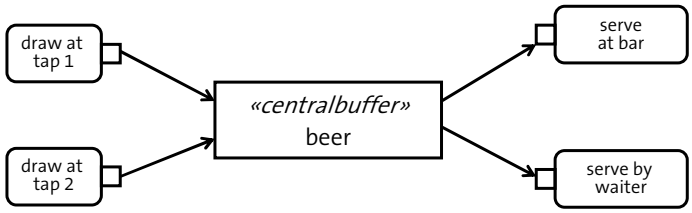
- multiple outgoing edges from an output pin = token competition
- first edge to accept the token gets it
  (!) race conditions, non-determinism

Parameters



- Parameters pass objects/data in and out of activities

Central Buffer Nodes



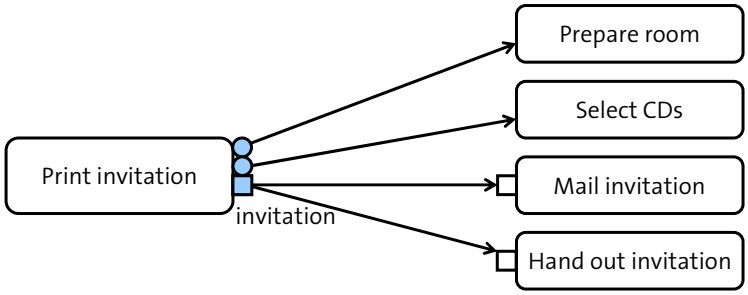
- Collecting from multiple sources
- Buffering for multiple consumers

III – In Depth – Semantics and Pitfalls

- Many syntactic constructs
- Local semantics
- HOW DOES IT WORK TOGETHER?

(!) Not yet documented in a precise and consistent way

Flows revisited



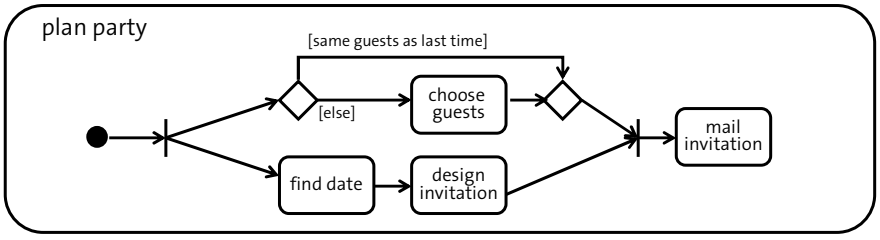
- (!) Multiple outgoing control flows: all get a token!
- (!) Multiple outgoing object flows: one gets the token!

Token Competition vs Traverse to Completion

- Token Competition: An object/data token can move down one (of multiple) outgoing object flows
- Traverse-to-completion: A token will only move down a completely open path (no buffering at control nodes)
- An activity will only accept tokens if it can get all required tokens

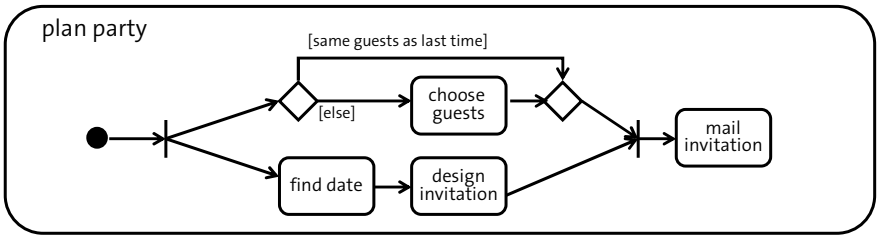
HOW DOES THAT WORK TOGETHER?

Forks revisited



- This AD fragment will never work for the [same guests...] case (according to the current semantics), because
  - The uppermost branch (assuming we will invite the same guests) cannot be open unless *design invitation* produces a token
  - The fork will be unable to supply a token to the lower branch because the topmost one isn't open => *design invitation* will never get a token
  - The fragment deadlocks!

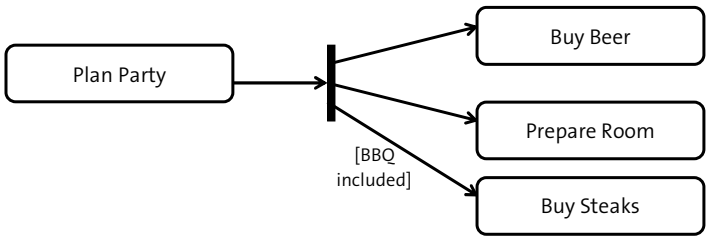
Fixing the semantics of the Fork Node*



- Forks will be allowed to buffer tokens
  - If one of the outgoing flows is open, the fork will accept a token
  - This token is duplicated for all edges, moves down all open ones, FIFO-queued on the blocked ones

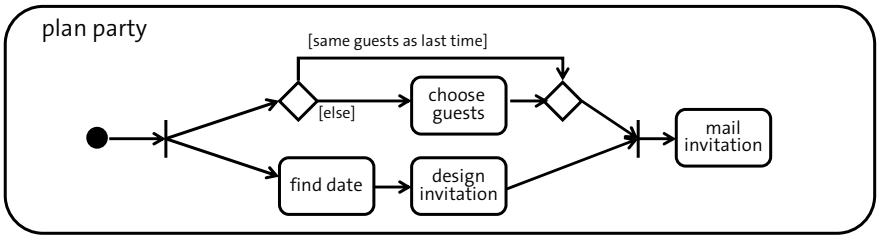
** currently a proposal in the Finalization Task Force*

Fixing the semantics of the Fork Node (2)



- What about Forks with guards on the outgoing edges?
  - some paths are permanently blocked (guards fail)
  - FIFO-queued tokens will block this path forever
  - ⇒ Tokens will only be queued for outgoing flows which are temporary blocked, no tokens will be produced for flows with failed guards.

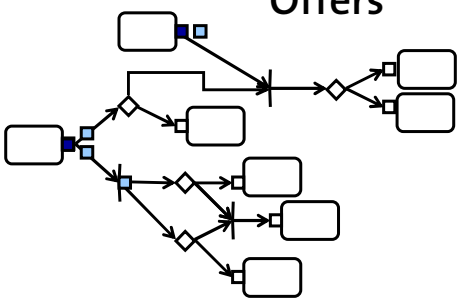
Joins revisited



- How can a join decide whether all of its inputs offer tokens?
  - Simple for lower branch
  - Complex evaluation for upper branch
- Tokens should not move down partial paths for deadlock prevention
- But caching partial evaluations is useful

=> Introduce Offers as “tentative tokens”

Offers

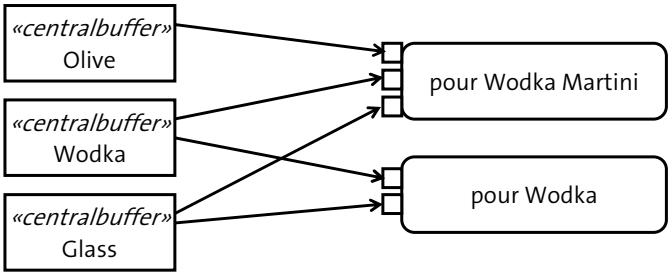


- Offers move down complex paths made up of control nodes and flows
- Offers represent a token
- Once an offer finds an open path (i.e. is accepted by an Object Node or Executable Node), the represented token moves along the path
- Token Competition => multiple offers; if one is accepted, all other vanish

Token/Offer semantics

- Token competition by having multiple concurrent offers for one token
- Traverse-to-Completion semantics by not moving tokens unless a completely open path is found
- Local evaluations possible due to the caching of partial evaluations (i.e. offers can be buffered)

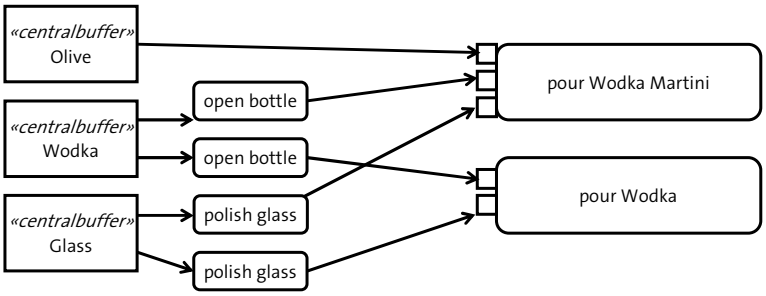
Deadlock prevention revisited



What happens, if only one glass and one bottle of Wodka is left?

- Actions accept all their inputs at once
- Prevention against “direct” deadlocks

Deadlock prevention revisited (2)

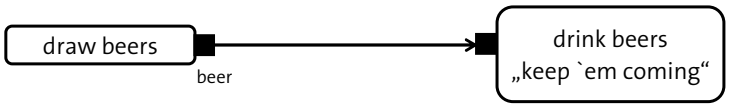


What happens, if only one glass and one bottle of Wodka is left?

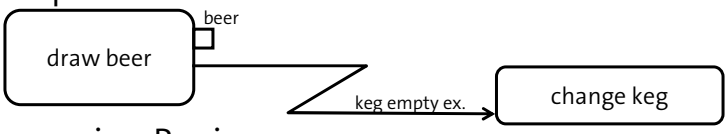
- No prevention against "remote" deadlocks

IV – Advanced Concepts -Things I did not tell you-

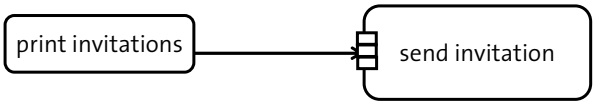
- Streaming Pins



- Exceptions



- Expansion Regions



-Things I did not tell you- (2)

- Transformations
- Merge specifications
- Structured Activites
- ParameterSets
- Partitions
- ...

=> There is much to explore

Sources and Literature

- UML Specifications
  - **Infrastructure**: The definition of the core concepts of the UML. Mostly for people who want to understand the inner structures, less important for "pure users". http://www.omg.org/cgi-bin/doc?ptc/2003-09-15
  - Superstructure: The definition of the diagram elements, their notation and semantics. Reference for all things UML. Current state: Final Adopted Specification. Finalized version (Issued Specification) is expected "in 2004". http://www.omg.org/cgi-bin/doc?ptc/2003-08-02
  - State of the finalization: The OMG Issues Database contains all open/closed issues and related change proposals: http://www.omg.org/issues/uml2-superstructure-ftf.open.html

Sources and Literature

- UML Books
  - **Tom Pender : UML Bible**. Very thorough description of the UML 2 elements and concepts.
    http://www.amazon.com/exec/obidos/tg/detail/-/0764526049
  - **Martin Fowler: UML destilled**. Much more condensed look at UML 2 with enlightening and amusing comments.
    http://www.amazon.com/exec/obidos/tg/detail/-/0321193687

Sources and Literature

- Activity Diagrams
  - **Conrad Bock: Tutorial in Journal of Object Technology**. Very informative articles by one of the chief authors of the new ADS:
    http://www.jot.fm/issues/issue_2003_07/column3
    http://www.jot.fm/issues/issue_2003_09/column4
    http://www.jot.fm/issues/issue_2003_11/column1
    http://www.jot.fm/issues/issue_2004_01/column3
  - **Engels/Förster/Heckel/Thöne : Process Modeling in UML.** Chapter in the forthcoming book "Process Aware Information Systems" (Dumas/van der Aalst/ter Hofstede; Wiley 2005). Highligting the use of UML for the modeling of Workflow Systems