Adaptive workflow modeling and object flows

Jens Brüning

Putbus, 28.06.07

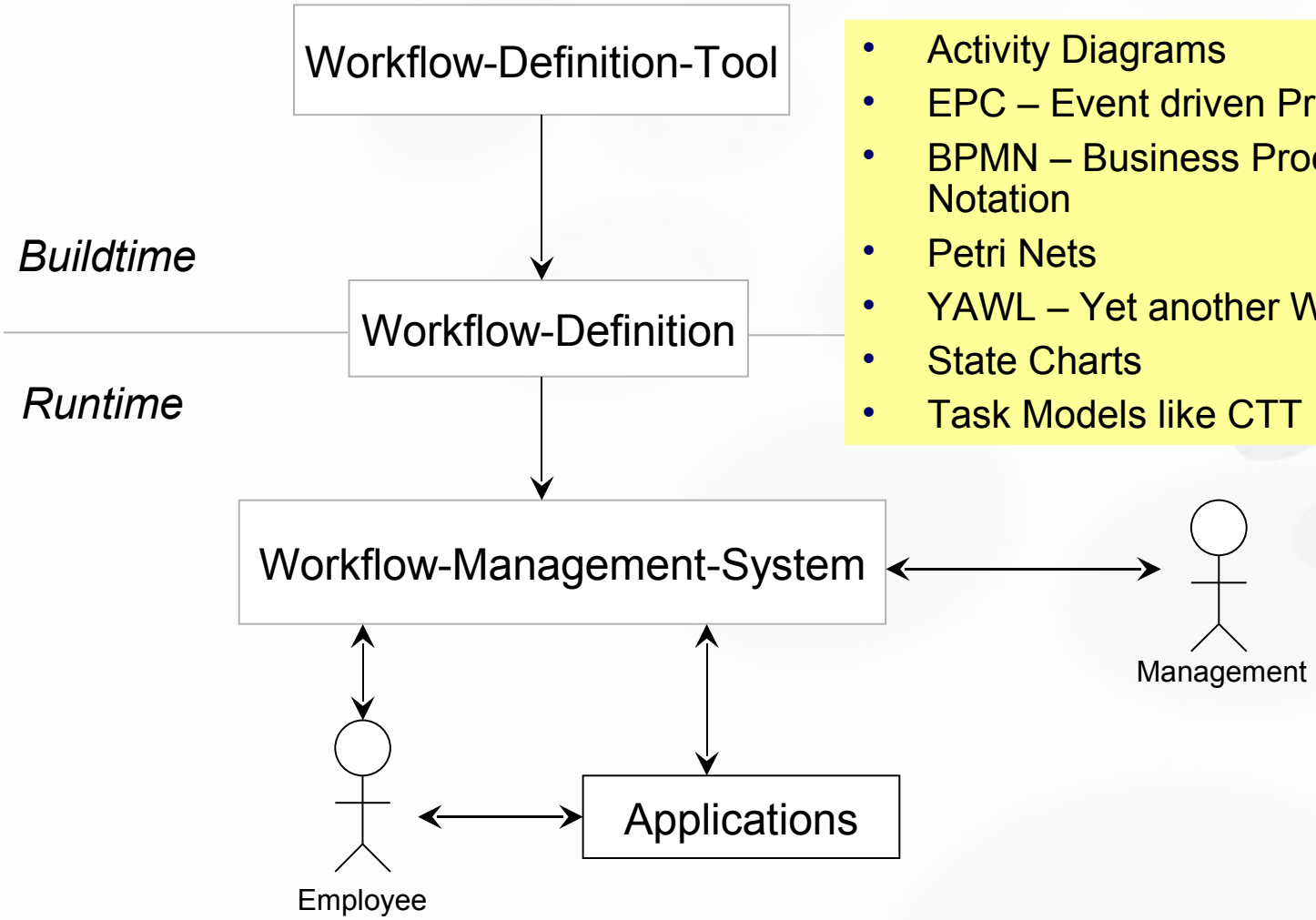
Overview

- Motivation and critics to workflow-modeling and WfMS
- Concepts for making workflows adaptive
- Object-flows in UML *Activity Diagrams*

Goals of workflow-modeling

- Getting a better understanding of the workflow in companies or organizations
- Getting the requirements for supporting IT-Systems
- More control (for the managers) about workflows in the company
- Analyze and optimize business processes

Workflow-System characteristics

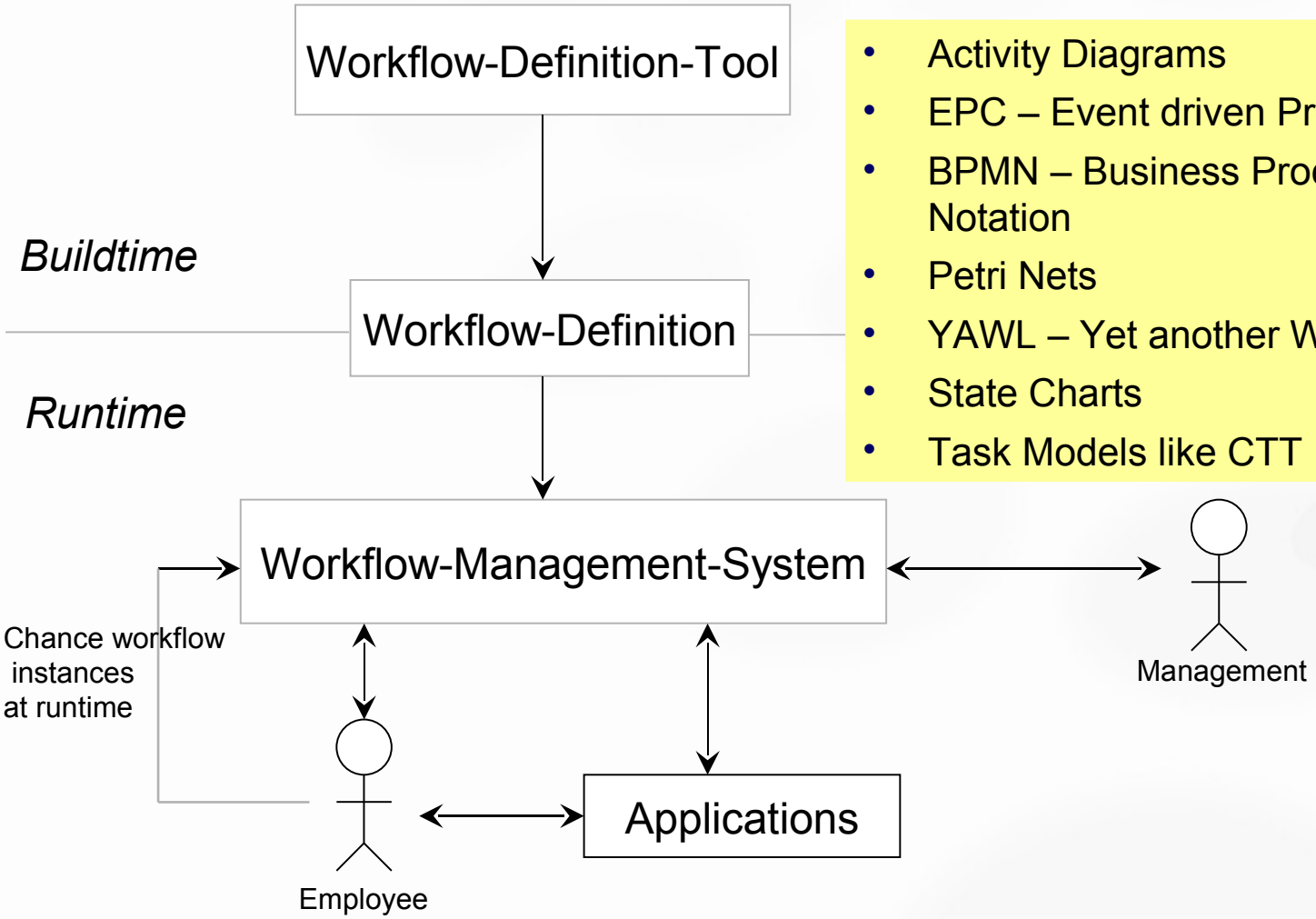


- Activity Diagrams
- EPC – Event driven Process Chains
- BPMN – Business Process Modeling Notation
- Petri Nets
- YAWL – Yet another Workflow Language
- State Charts
- Task Models like CTT

Problems to the classical Workflow-concept

- More control to the management
  - Employees can be better observed/controlled
  - Privacy problems
- Workflows can be too static for the employees
  - Less freedom for individual decisions
  - How to react on unpredicted situations?
  - Workflow is determined at beginning of workflow
  - Workflow cannot be evolved in runtime

Adaptive workflow concept



- Activity Diagrams
- EPC – Event driven Process Chains
- BPMN – Business Process Modeling Notation
- Petri Nets
- YAWL – Yet another Workflow Language
- State Charts
- Task Models like CTT

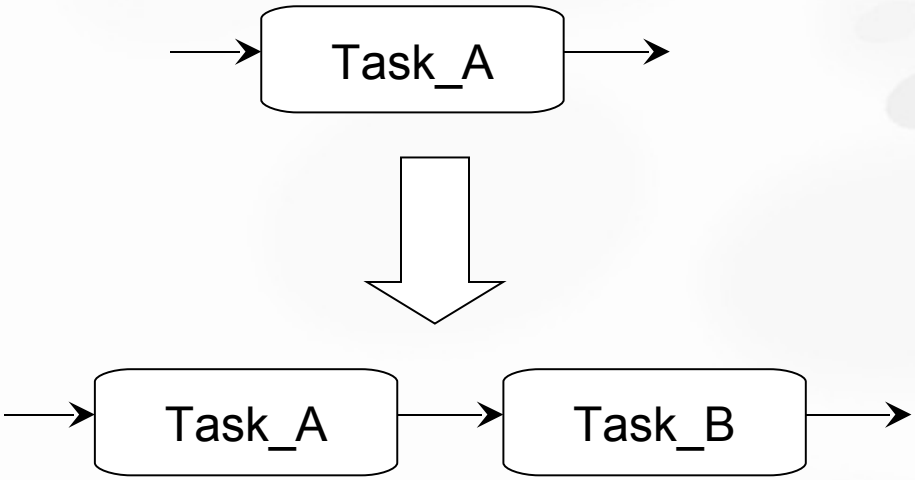
Adaptive Workflow modeling concepts

- Functions for manipulating the workflows / for making the workflow definitions adaptive:
  - *Append* function*
  - *Delete* function
  - *Reorder* function*
  - *Substitute* function*
  - *Change temporal relationship* function
  - *Insert in-between* function
  - *Exception* pattern

*: Already proposed by van der Aalst

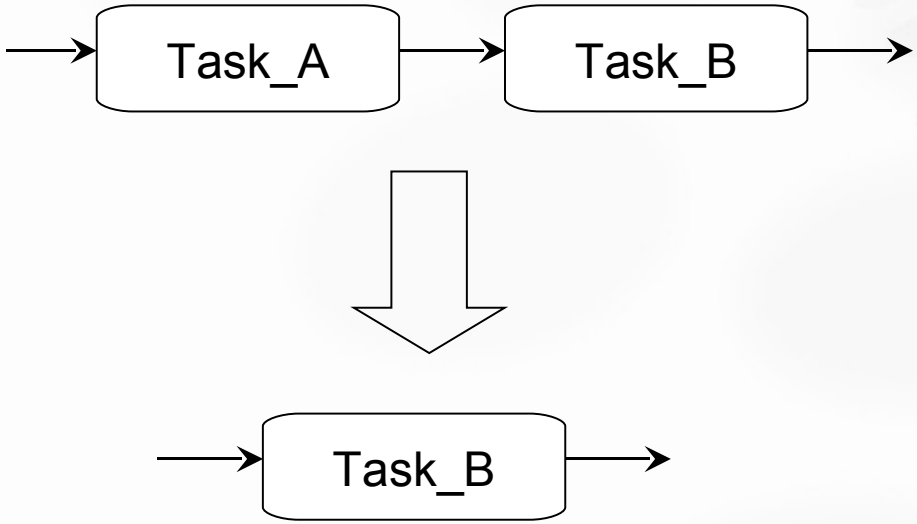
Append function

- Append function for extending the workflow
- Activity must not be in the state: *executed*
- Activity can be in the state: *running*



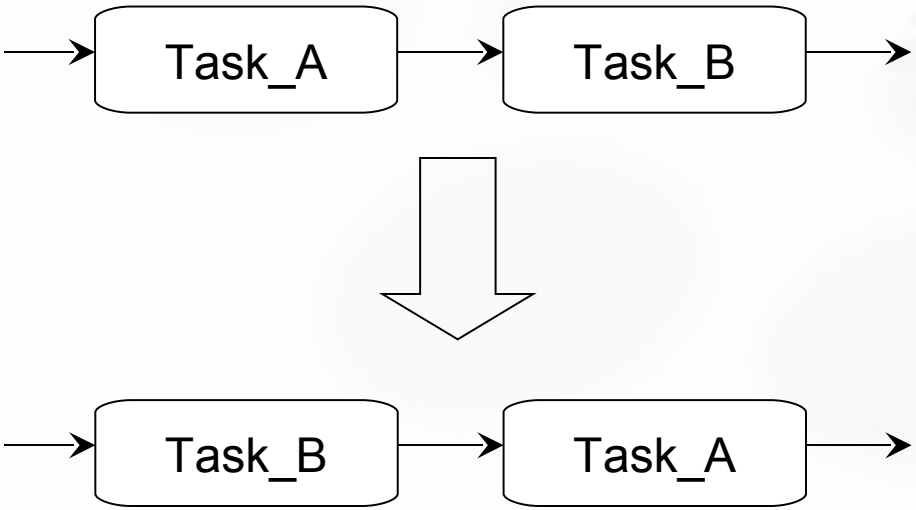
Delete function

- Delete function for reduce / downsize the workflow
- Activity must not in the state *executed* or *running*



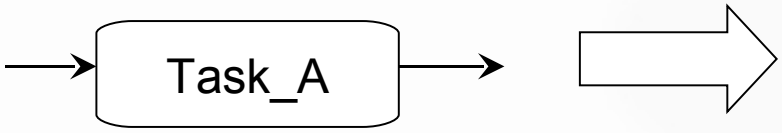
Reorder function

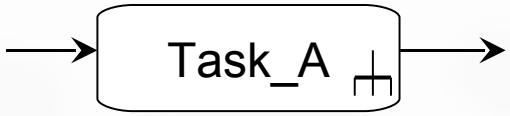
- Changes the order of activities in a sequence
- Task must not be in the state *executed* or *running*

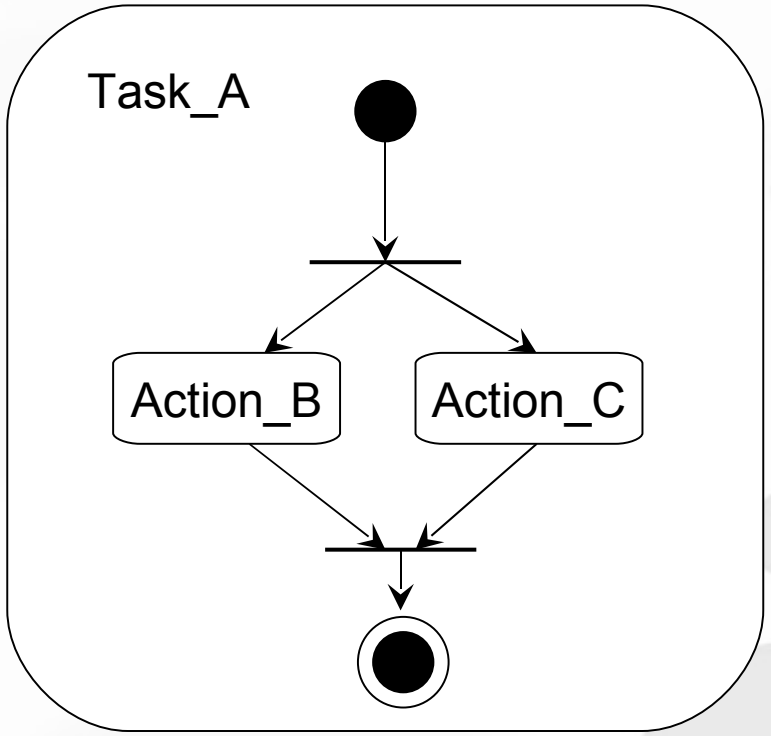


Substitution function for refinement

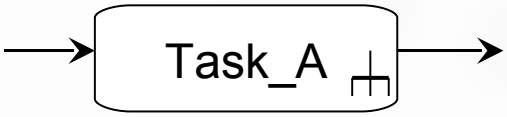
- Substitution function used for refining a task
- Can be used if the task has not started yet
- Task must not be in the state: *executed* or *running*

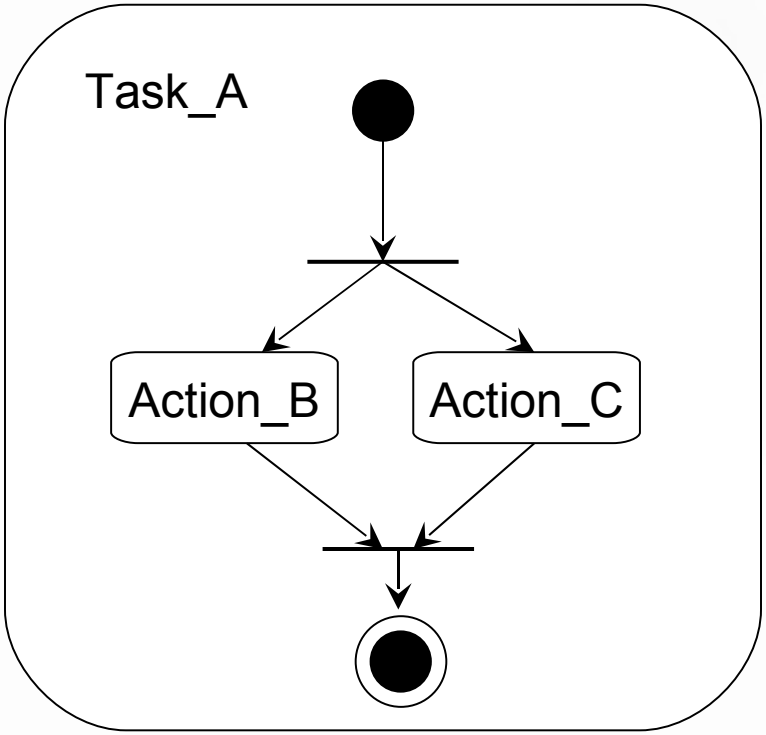




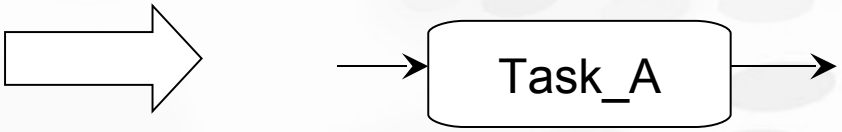


Substitution function for flattening



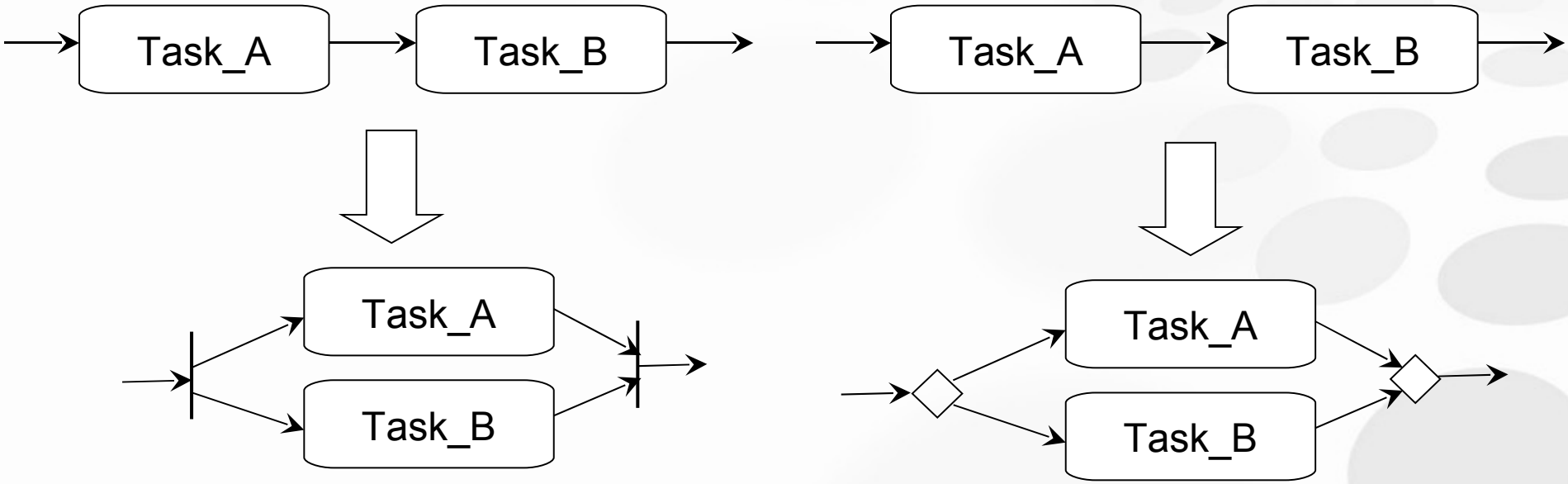


- Substitution function used for flattening a task
- Task must not be in the state *executed* or *running*



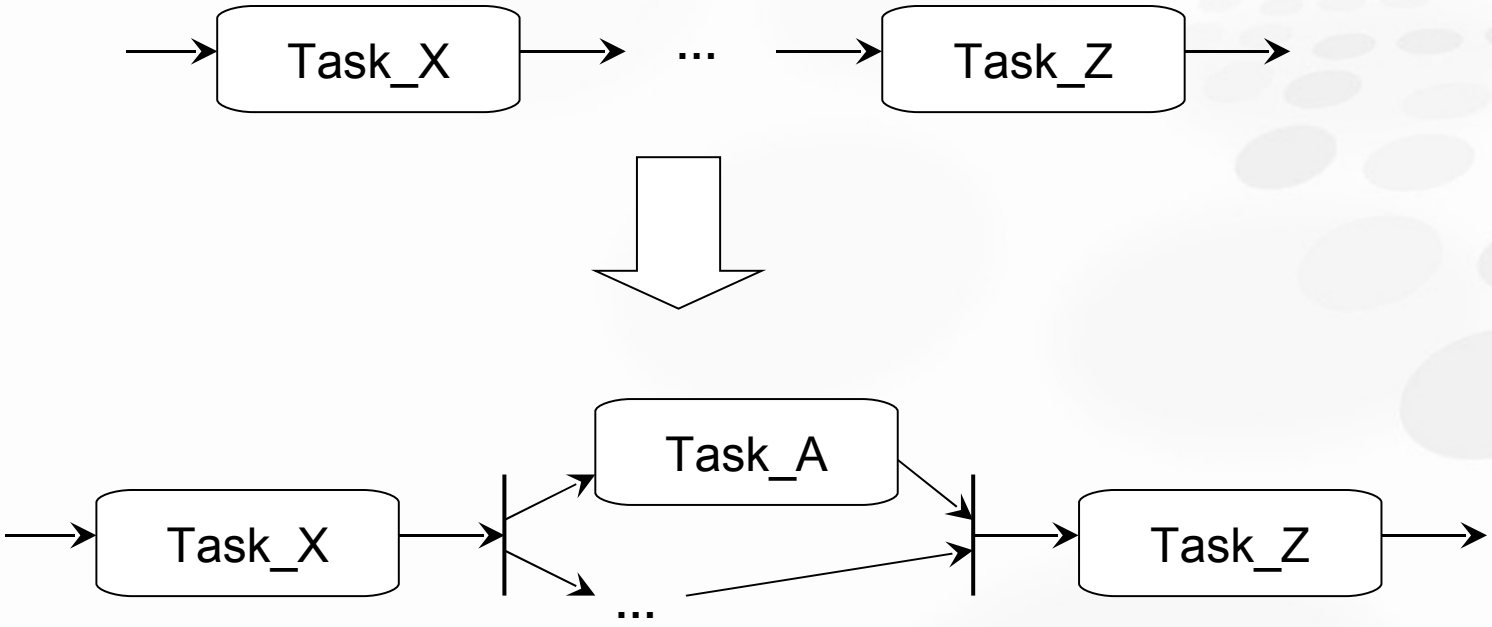
Change temporal relationship function

- Change temporal relationship function
  - sequence → parallel
  - sequence → choice, choice → sequence
  - choice → parallel



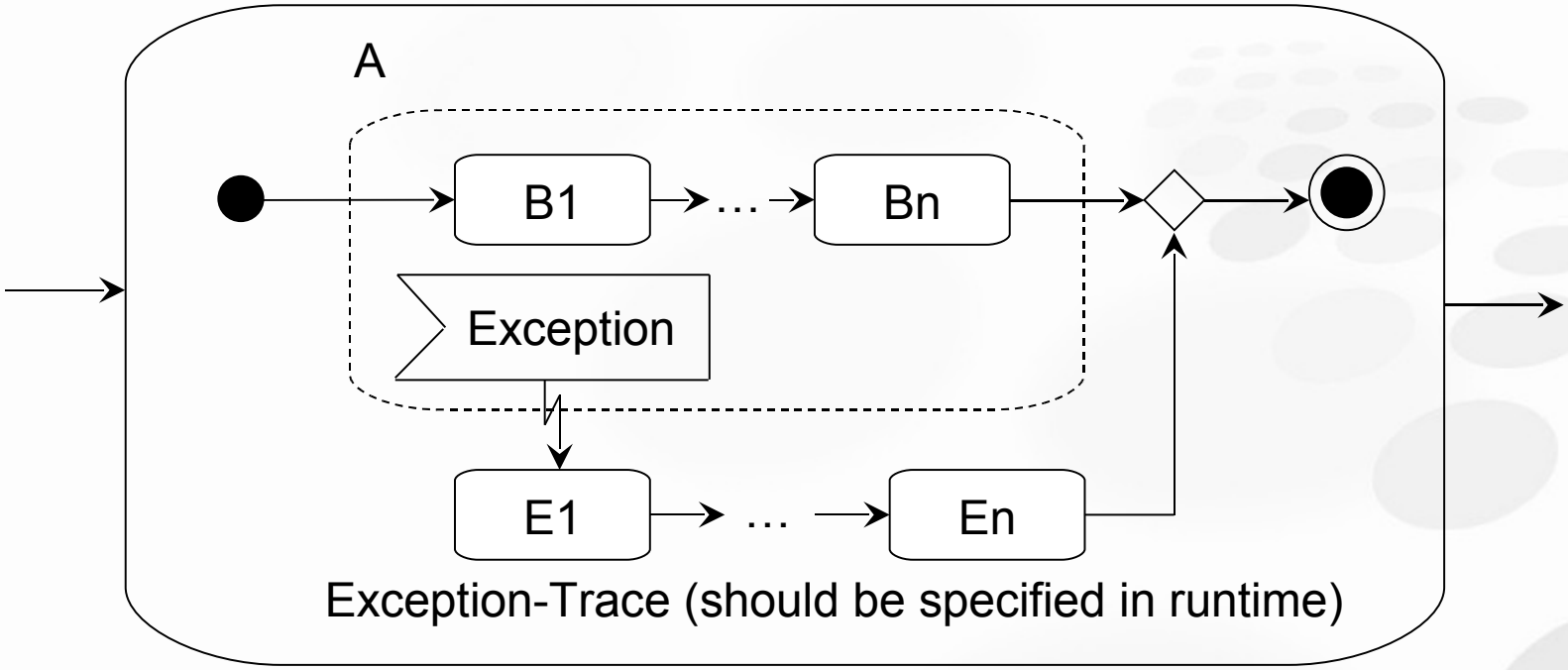
Insert in-between function

- Inserts a task in-between two activities
- Example: insert Activity A behind Activity X and before Activity Z



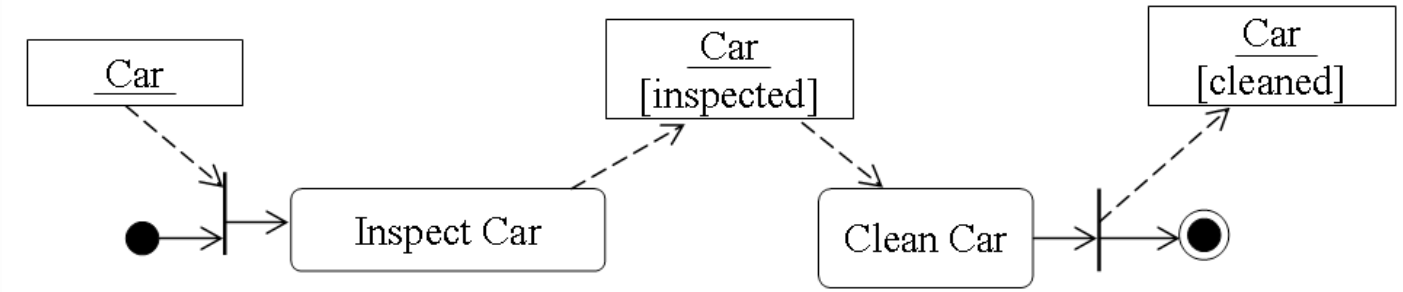
Exception function

- Used in structured activity nodes if an activity is crashed
- Activity can be fixed in exception trace and workflow can continue



Motivation for object flows

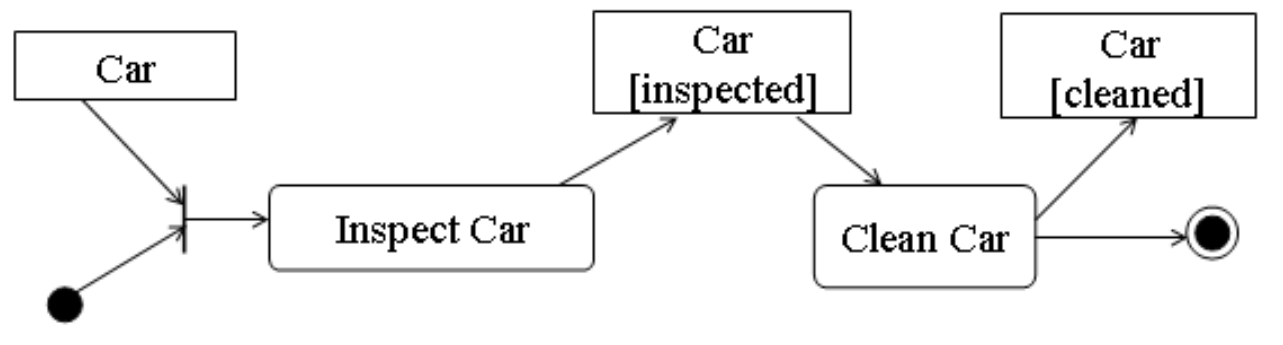
- Objects can be used for input parameters
- Objects can be used for outputs of the activity
- Object Transformation process can be modeled in that way in the workflow



(UML 1.5)

Object flows in UML 2.1

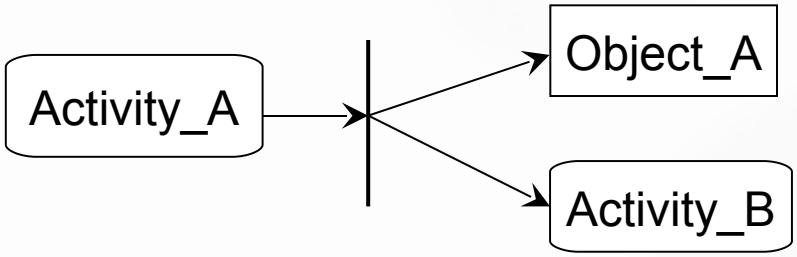
- Changes to UML 2.1 object flow concept
  - Flow operators are used in a different way in UML 2.1
  - Object nodes are buffers yet
  - In UML2 Object-Flows contain Control Flow



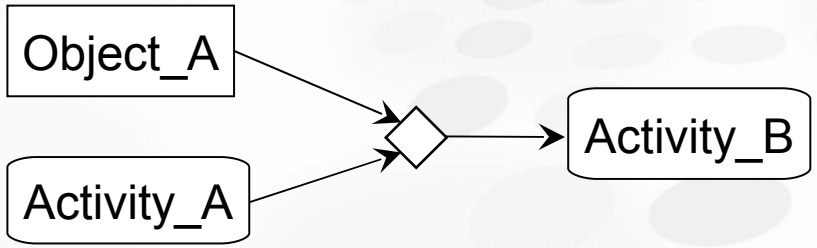
(UML 2.1)

Object flows in UML 2.1

- Object- and Control Flows cannot be mixed at *Choice*, *Merge* and *Fork*-nodes
- Almost all CASE tools allow these invalid diagrams (e.g. VisualParadigm, MagicDraw)



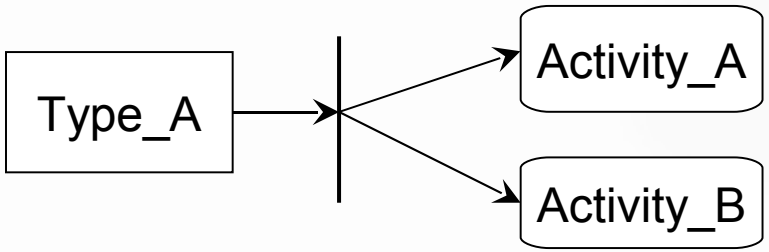
not allowed

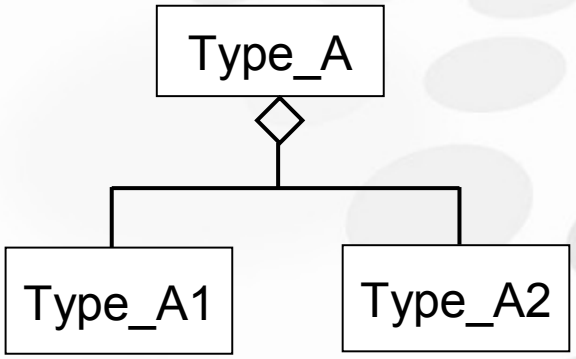


not allowed

Forks in Object Flows

- possible behaviours for the fork
  - Copy of a reference
  - Copy of an object
  - Split of a set token
  - Split of a composed object

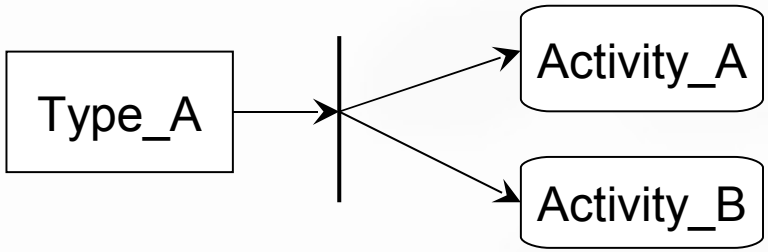




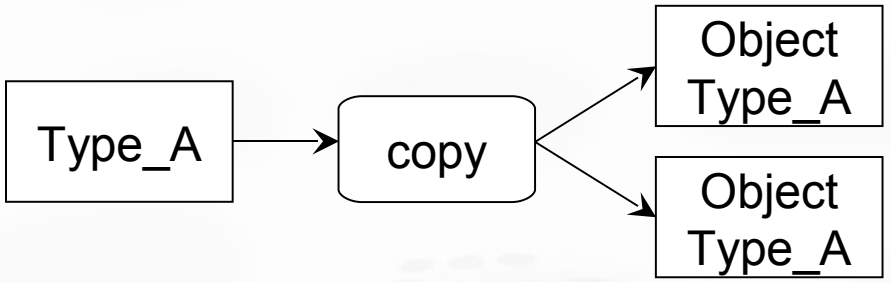
allowed

Behaviour of forks in object flows

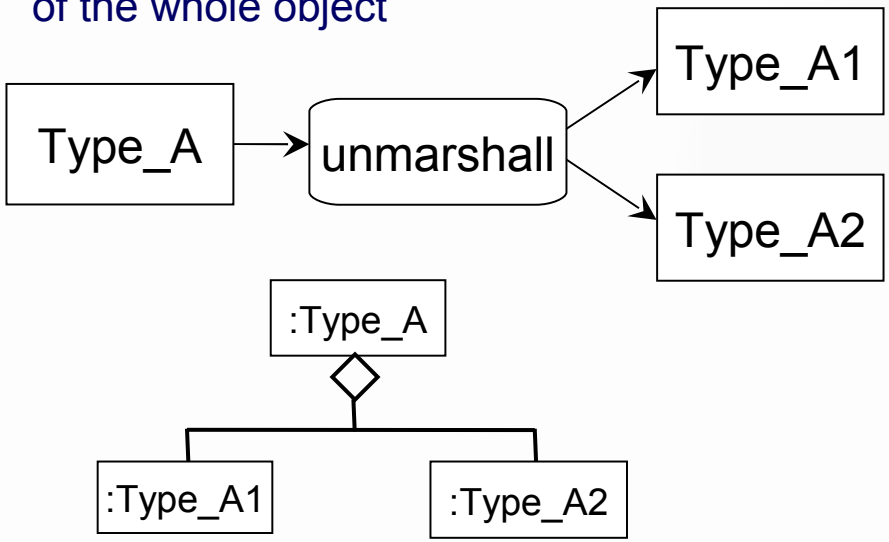
- Copy a reference of an object



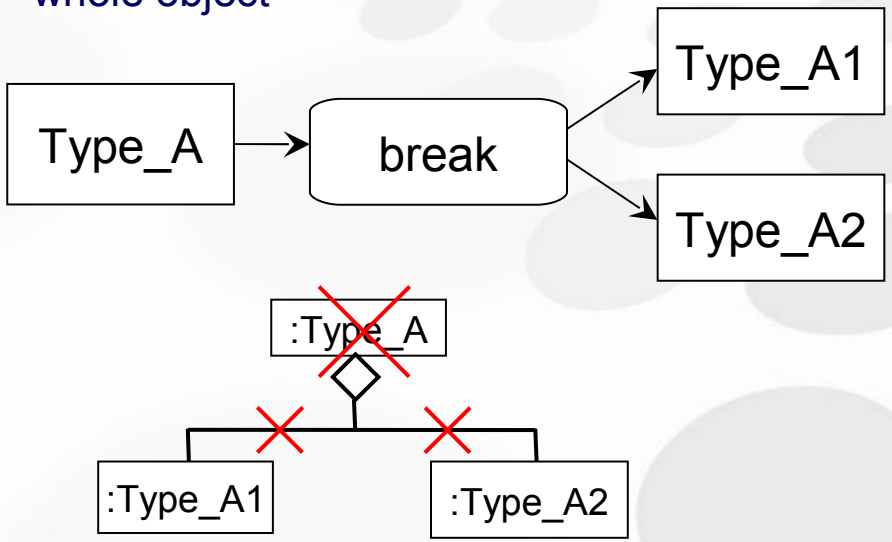
- Copy of an object itself



- Getting the components out of the whole object

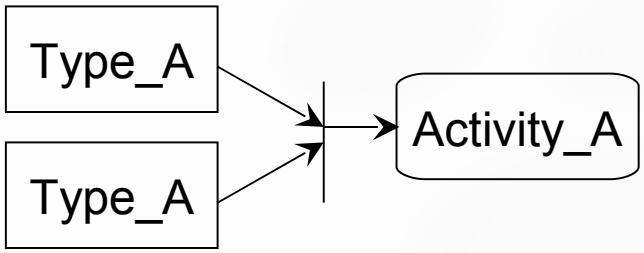


- Breaks the components out of the whole object

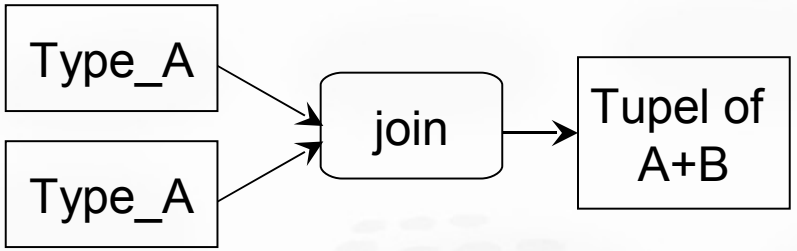


Behaviour of joins in object flows

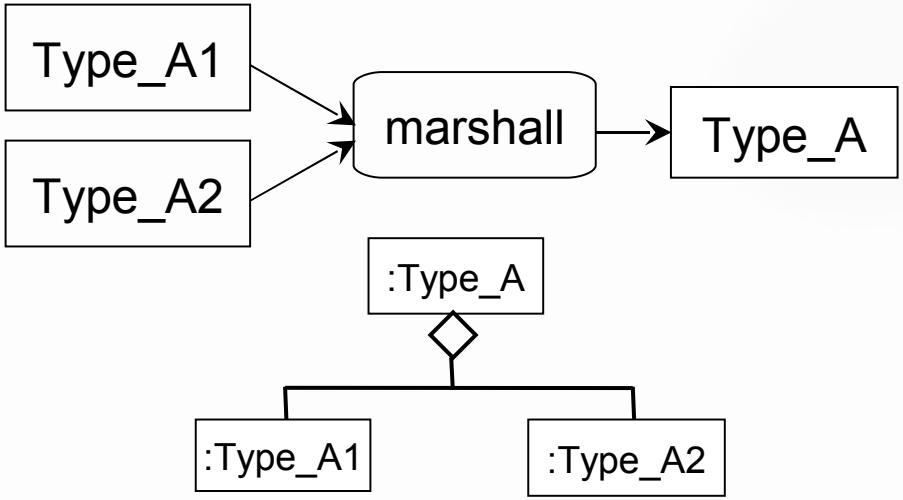
- Two object tokens are waiting at the join
- Activity_A is executed twice afterwards



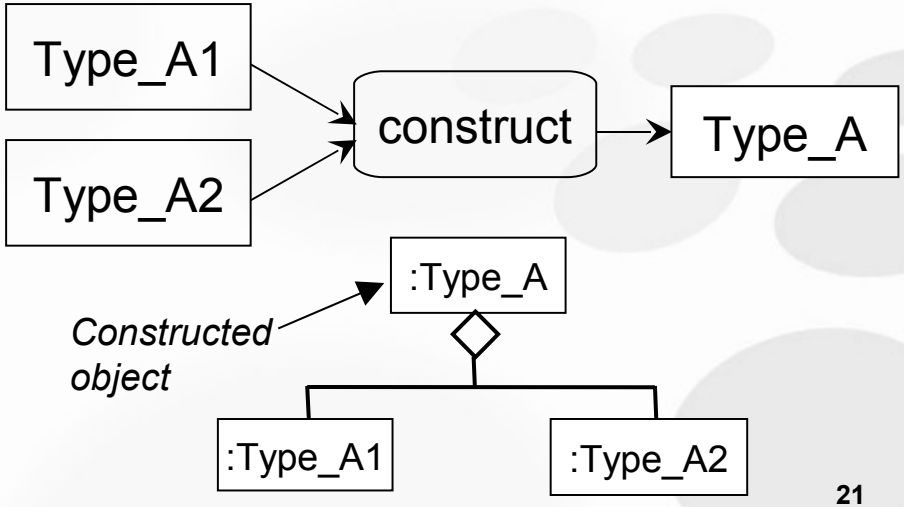
- Two object tokens are waiting at the join
- following activities are executed once



- Together belonging tokens are joint
- the whole object of Type_A is already existent

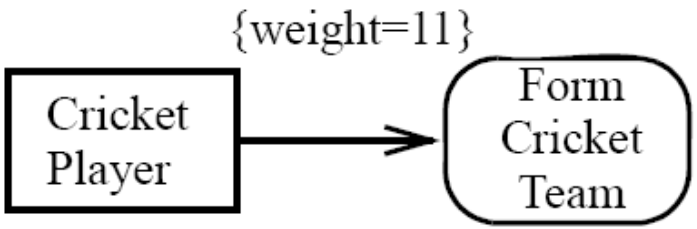


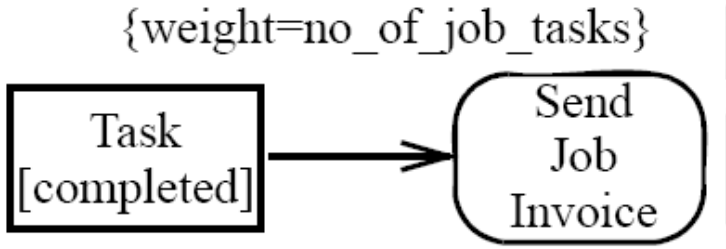
- Whole object is constructed
- outcoming token is referencing the whole



Weighted Edges in Object Flows

- Weighted edges are only allowed form object-nodes to activities
- Number of tokens traverse the edge at the same time and the activity is invoked once

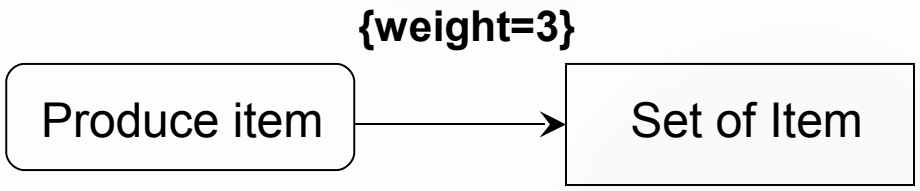




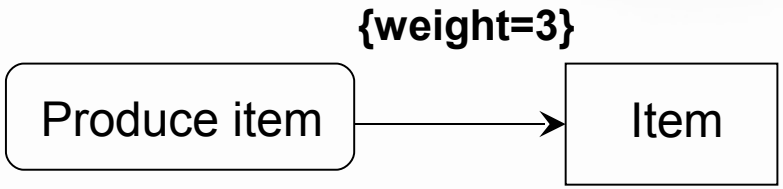
[Examples from the UML 2.1 specification]

New semantics for weighted edges

- Weighted edges wrap the object tokens into set tokens
- Weighted edges from activities to object nodes would be allowed then

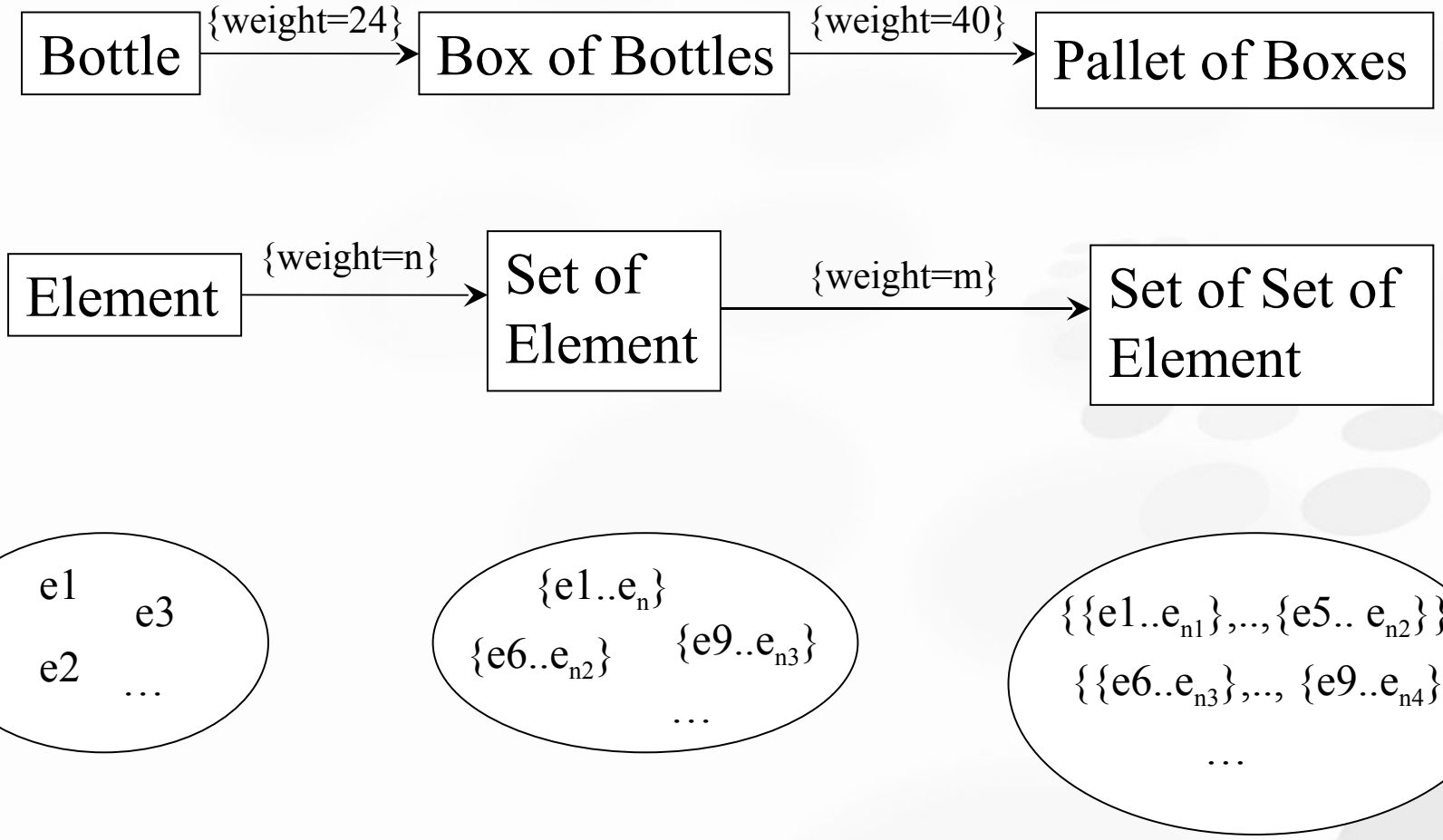


3 items are produced and are collected in one set token



Would not be allowed

Proposed semantics for Weighted Edges



Finally, thank you for your attention