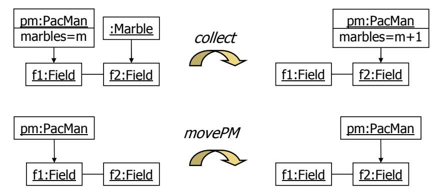
University of Leicester

Graph Transformation in a Nutshell

Reiko Heckel
University of Leicester, UK

SegraVis Advanced School on Visual Modelling Techniques
Leicester 8-11 September 2006

Rules of the PacMan Game: Graph-Based Presentation, PacMan



Why it is fun: Programming By Example

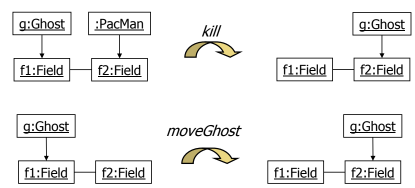
StageCast (www.stagecast.com): a visual programming environment for kids (from 8 years on), based on

- behavioral rules associated to graphical objects
- visual pattern matching
- simple control structures (priorities, sequence, choice, ...)
- external keyboard control

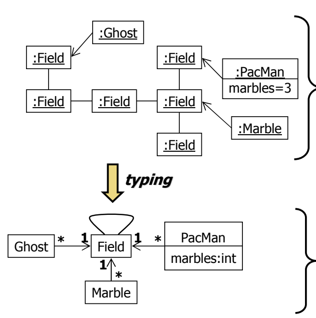
→ intuitive rule-based behavior modelling

Next: abstract from concrete visual presentation

Rules of the PacMan Game: Graph-Based Presentation, Ghost



States of the PacMan Game: Graph-Based Presentation



instance graph *(represents a single state; abstracts from spatial layout)*

type graph *(specifies legal instance graphs → state space)*

Outline

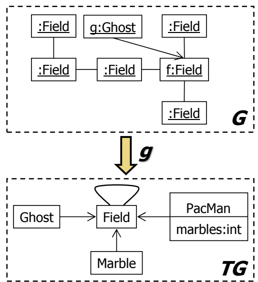
- Graph Transformation
  - ✓ why it is fun
  - how it works
- Applications and Theory

A Basic Formalism: Typed Graphs

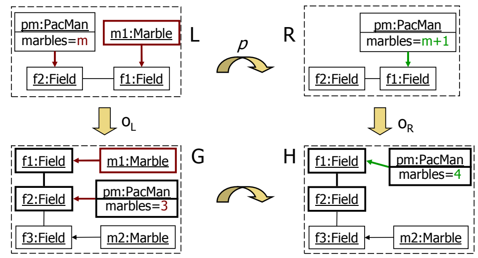
Directed graphs
- multiple parallel edges
- undirected edges as pairs of directed ones

Graph homomorphism as mappings preserving source and target

Typed graphs given by
- fixed type graph *TG*
- instance graphs *G* typed over *TG by* homomorphism



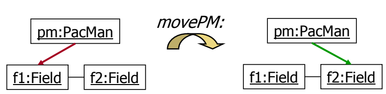
Transformation Step



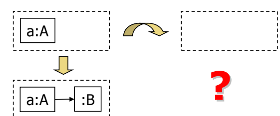
1. select rule $p : L \rightarrow R$; occurrence $o_L : L \rightarrow G$
2. remove from G the occurrence of $L \setminus R$
3. add to result a copy of $R \setminus L$

Rules

p: L → R with $L \cap R$ well-defined, in different presentations
- like above (cf. PacMan example)
- with $L \cap R$ explicit [DPO]: $L \leftarrow K \rightarrow R$



Semantic Questions: Dangling Edges

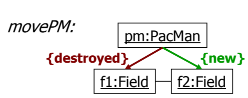


✖ conservative solution: application is forbidden
- invertible transformations, no side-effects

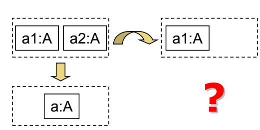
✖ radical solution: delete dangling edges
- more complex behavior, requires explicit control

Rules

p: L → R with $L \cap R$ well-defined, in different presentations
- like above (cf. PacMan example)
- with $L \cap R$ explicit [DPO]: $L \leftarrow K \rightarrow R$
- with *L, R* integrated [UML, Fujaba]: $L \cup R$ and marking
  - *L - R* as {destroyed}
  - *R - L* as {new}



Semantic Questions: Conflicts



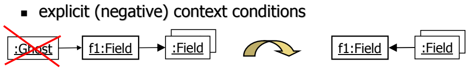
✖ conservative solution: application is forbidden
- invertible transformations, no side-effects

✖ radical solution: give priority to deletion
- more complex behavior, requires explicit control

Advanced Features

Dealing with unknown context

- set-nodes (multi-objects): match all nodes with the required connections
- explicit (negative) context conditions

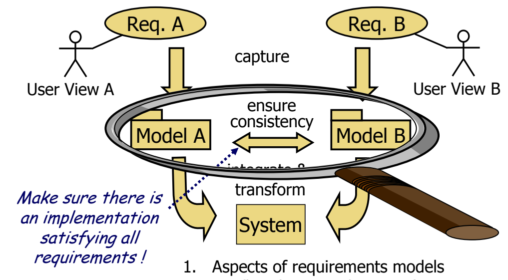


(turns f1 into a trap by reversing all outgoing edges to Field vertices, but only if there is no Ghost)

Control Structures

- priorities
- programmed transformation

Motivation: Software Development as Integration of Views



1. Aspects of requirements models
2. Conflicts between functional requirements
3. Theory and tool support

A bit of History …

Chomsky Grammars	**Term Rewriting**	**Petri Nets**

Graph Transformation and Graph Grammars

Diagram Languages	Models of Computation	Behaviour Modelling and Visual Programming

Aspects of Requirements Models

Model A

Model B

1. Static domain model: Agree on vocabulary first !
   → class and object diagrams
2. Business process model: Which actions are performed in which order ?
   → use case description in natural language, activity or sequence diagrams, etc.

Outline

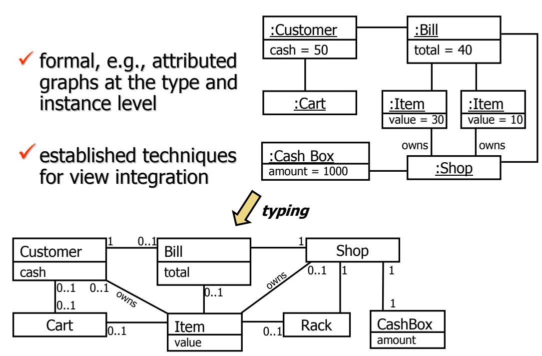
✓ Graph Transformation
- ✓ why it is fun
- ✓ how it works

✖ Applications and Theory
- Modelling and Analysis of Functional Requirements
- Model Transformation and Semantics

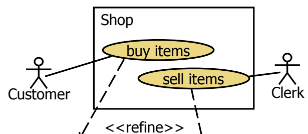
Structure: Class and Object Diagrams

✓ formal, e.g., attributed graphs at the type and instance level

✓ established techniques for view integration



Behaviour: Use Case Description by Structured Text



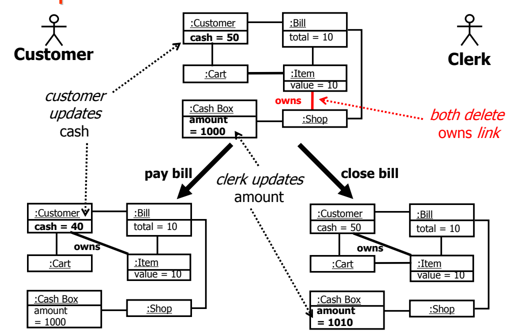
- ✓ based on vocabulary of integrated domain model

- take shopping cart
- select items from rack
- take items out of cart
- pay required amount
- collect items

- create empty bill for new customer
- take items out of customer's cart
- add them to the bill
- collect payment
- pack and give items to customer

- ✗ no way to tell if views are consistent

Conflicts Between Functional Requirements



Aspects of Requirements Models

Model A

Model B

- ✓ Static domain model: Agree on vocabulary first !
  - → class and object diagrams
- ✓ Business process model: Which actions are performed in which order ?
  - → use case description in natural language, activity or sequence diagrams, etc.

3. Functional model: What happens if an action is performed ?
   - → pre-/post conditions as logic constraints
   - → transformation rules on object diagrams (Fusion, Catalysis, Fujaba, formally: graph transformations)

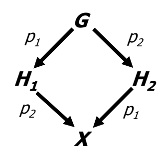
Theory: Independence, Causality and Conflicts in Graph Transformation

- ✗ Alternative steps are *parallel independent* if they do not disable each other.

  Otherwise they are *in conflict*.

- ✗ Consecutive steps are *sequentially independent* if they may be swapped without affecting the result.

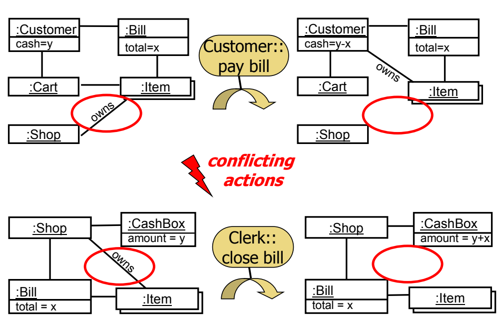
  Otherwise they are *causally dependent*.



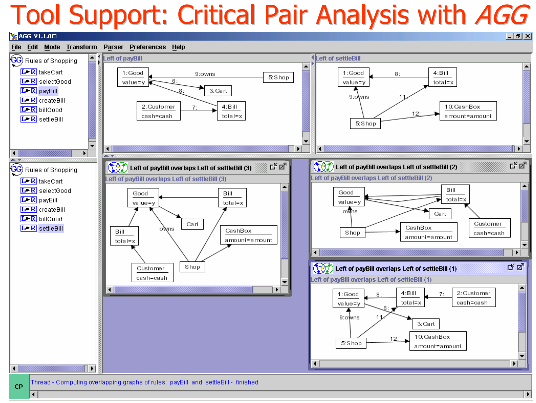
Idea: Find *potential* conflicts and causal dependencies between rules by **critical pair analysis**

Characterization [EPS73]: Two (alternative or consecutive) steps are independent **iff** all commonly accessed items are in read-access only.

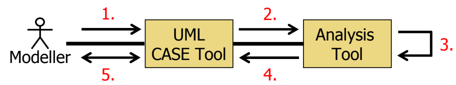
Function: Transformation Rules on Object Diagrams



Tool Support: Critical Pair Analysis with *AGG*



Usage Scenario



1. input model to CASE tool
2. import model by analysis tool
3. analyze model for conflicts
4. back annotate models with conflicts
5. interprete and improve models

Domain expert: "*buy items and sell items should not be in conflict*"
Modeller: "*inconsistency between views*"

Model-driven Development

- Focus and primary artifacts are models instead of programs
- Core activities include
  - maintaining consistency
  - evolution
  - translation
  - execution

  of models
- These are examples of model transformations
- A math. foundation is needed for studying
  - Expressiveness and complexity
  - Execution and optimisation
  - Well-definedness
  - Semantic correctness

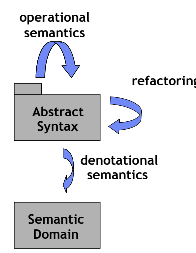
  of transformations
- Graph transformations can be one such foundation

Outline

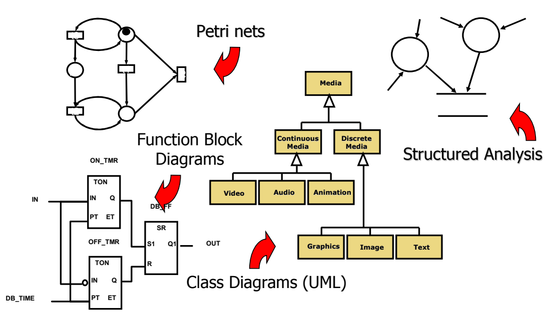
- ✓ Graph Transformation
  - ✓ why it is fun
  - ✓ how it works
- Applications and Theory
  - ✓ Modelling and Analysis of Functional Requirements
  - Model Transformation and Semantics

Outline

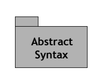
- Model transformation
  - denotational semantics
    - analysis → synthesis
  - operational semantics
  - refactoring



Visual Modeling Techniques

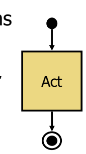


Context-Free Graph Grammar

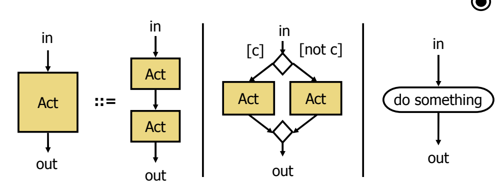


Concrete Syntax of *Well-Formed* Activity Diagrams

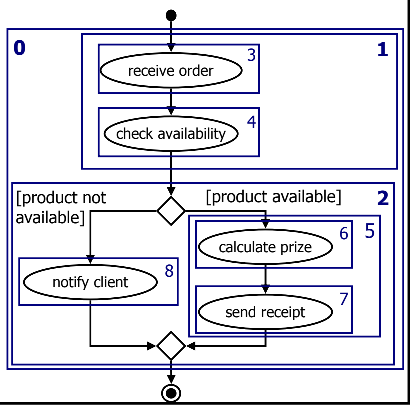
Start Graph:



Productions in EBNF-like notation:



Analysis

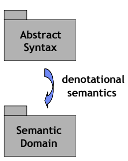


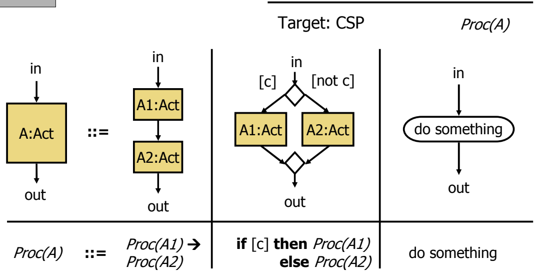
Is this Good Enough?

- ✓ Visual
  - abstract syntax or concrete syntax templates
- ✓ Bi-directional
  - swap source and target grammars
- ✓ Declarative
- × Expressive ?
  - context-free graph languages only
- × Traceable ?
  - through naming conventions
- × Efficient ?
  - NP complete parsing problem
- × ...

➔ Triple Graph Grammars

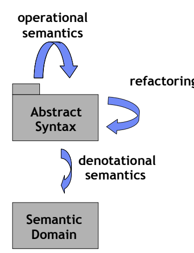
Pair Grammar





Outline

- ✖ Model transformation
  - ✓ denotational semantics
  - operational semantics
  - refactoring



Synthesis

Proc(A0)

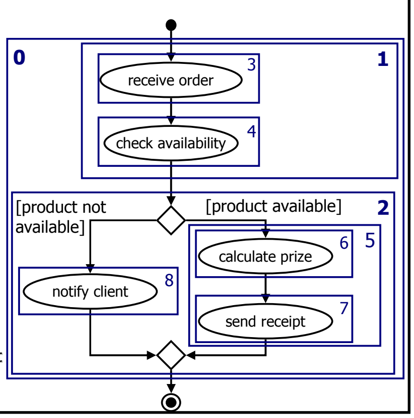
Proc(A1) ➔ *Proc(A2)*

...

Proc(A3) ➔
Proc (A4) ➔
if [product available]
then *Proc(A5)*
else *Proc(A8)*

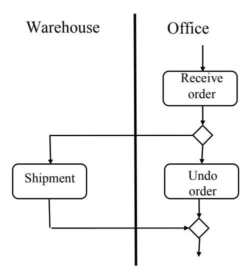
...

receive order ➔
check availability ➔
if [product available]
then calculate prize
➔ send receipt
else notify client

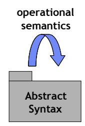


Example: Executable Business Process

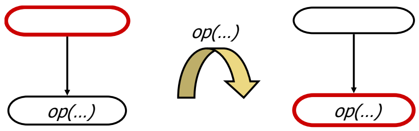
- ✖ refactoring of business processes, replacing centralised by distributed execution
- ✖ How to demonstrate preservation of behaviour?
  1. specify operational semantics of processes
  2. define transformations
  3. show that transformations preserve semantics



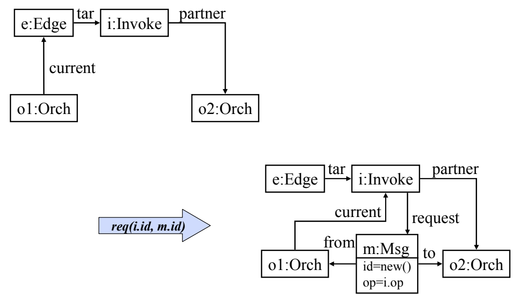
Operational Semantics: Idea



- diagram syntax plus *runtime state*
- GT rules to model state transitions



Rules: Invoke another Service



Operational Semantics: Formally

GTS = (TG, P) with start graph G_0

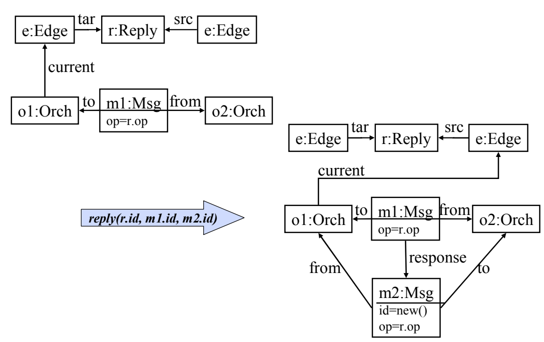
defines transition system

$LTS(GTS, G_0) = (S, L, \rightarrow)$

taking

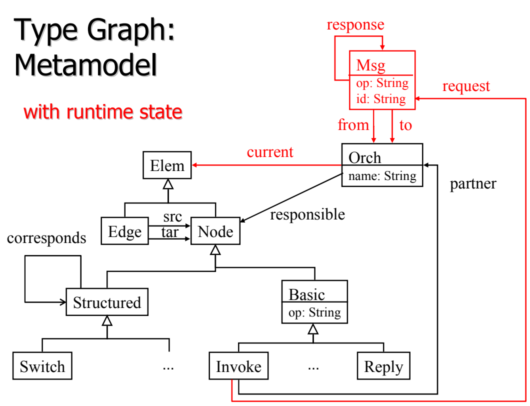
- as states *S* all graphs reachable from G_0
- observations on rules as labels
- transformations as transitions

Rules: Answer the Invocation

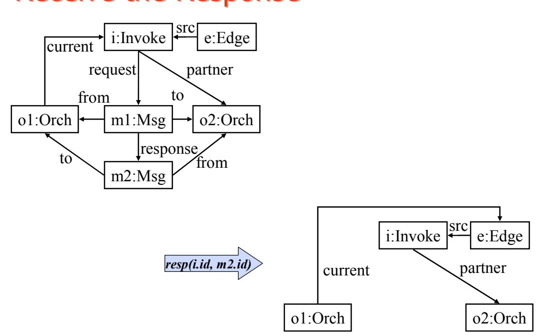


Type Graph: Metamodel

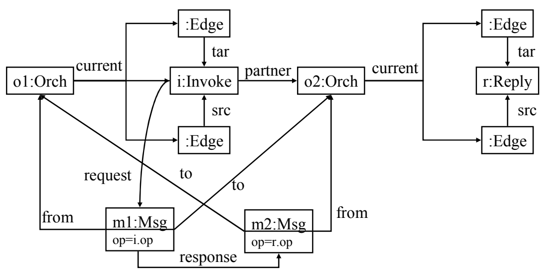
with runtime state



Rules: Receive the Response

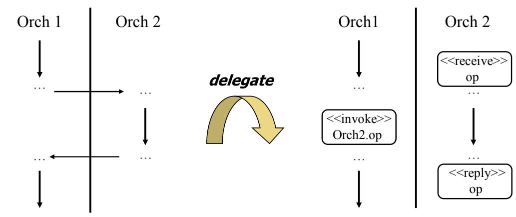


Simulation



Observations: *req(i.id, m1.id); reply(r.id, m1.id, m2.id); resp(i.id, m2.id)*

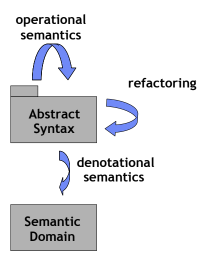
Refactoring Executable Business Processes



- replace local control flow by message passing

Outline

- Model transformation
  - ✓ denotational semantics
  - ✓ operational semantics
  - refactoring



Preservation of Semantics

Show for each refactoring P ➔ P′ that
P′ simulates P, i.e.
- P $\rightarrow_{obs}$ Q implies P′ $\rightarrow_{obs}$ Q′
- Q′ simulates Q

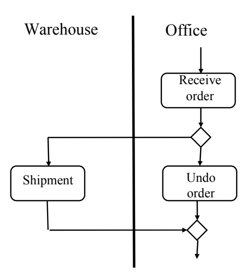
and vice versa.

Approach:
- mixed (local) confluence
- critical pair analysis

$$\begin{tikzcd} P \arrow[r, "obs"] \arrow[d] & Q \arrow[d] \\ P' \arrow[r, "obs"] & Q' \end{tikzcd}$$

Example:
Executable Business Process

- refactoring of business processes, replacing centralised by distributed execution
- How to demonstrate preservation of behaviour?
  - ✓ specify operational semantics of processes
  2. define transformations
  3. show that transformations preserve semantics



Theory: Sources of Inspirations

Chomsky Grammars	Term Rewriting	Petri Nets
⬇	⬇	⬇

Graph Transformation and Graph Grammars

- Formal language theory of graphs;
- Diagram compiler generators

- Well-definedness
  - Termination
  - Confluence
- Semantics of process calculi

- Concurrency theory
  - Causality and conflict
  - Processes, unfoldings
  - Event-structures