

Software Engineering and Petri Nets

Mauro Pezzè
Università degli Studi di Milano - Bicocca



credits

This material is taken from the Tutorial on Software Engineering and Petri presented at ICATPN 2003 by Gregor Engels and Mauro Pezzè



www.lta.disco.unimib.it
© 2003 Gregor Engels, Mauro Pezzè



Mission Statement

- ▶ understanding the prospects for Petri nets in software engineering research and practice
 - what are the opportunities for Petri Nets?
 - what are the barriers and constraints?
- ▶ understanding today's software engineering problems
- ▶ not an introduction to software engineering
 - although we introduce some aspects of it
- ▶ not an introduction to Petri nets
 - (some) familiarity with Petri nets is expected

Role of Petri Nets

Observations:

- ▶ Role of Petri Nets is not increasing in Software Engineering
- ▶ Petri Nets are displaced by other formalisms

but

- ▶ Software Engineering is facing severe problems in building and maintaining high-quality software

Question:

“can Petri nets contribute in solving these problems?”

Problem- vs. Solution-oriented Disciplines

- ▶ Software engineering is a ***problem-oriented*** research discipline
 - driven by demands from the application field
- ▶ Petri net research is a **solution-oriented** research discipline
 - driven by theory

[Michal Young: 2000 Workshop on Software Engineering and Petri Nets]

Problem-oriented Disciplines

- ▶ are eclectic
 - different ways of defining sub-problems
 - radically different solution techniques

- ▶ are fickle
 - adopting and abandoning solution techniques as the field evolves

[Michal Young: 2000 Workshop on Software Engineering and Petri Nets]

Solution-oriented Disciplines

- ▶ are more homogeneous
 - at least large sub-groups agree on common sets of problems
 - or they break into sub-disciplines that do, e.g., functional programming

- ▶ have a deep understanding of their disciplines
 - well-defined theory
 - availability of tools

[Michal Young: 2000 Workshop on Software Engineering and Petri Nets]

Petri Nets for Software Engineering

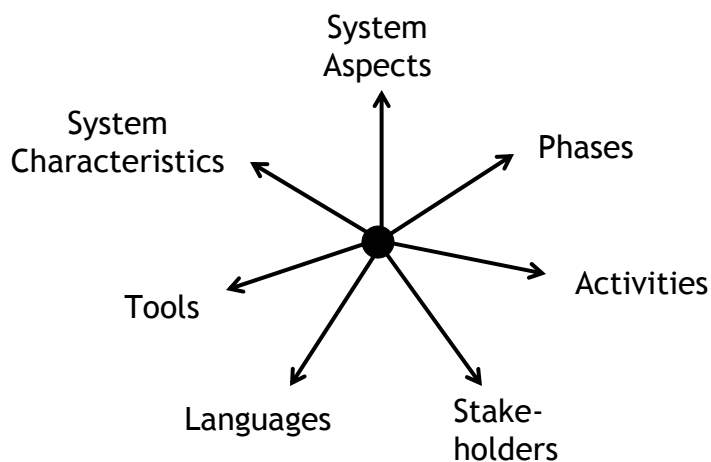
Questions:

- ▶ can results from a solution-oriented discipline like Petri Nets be effectively used to solve problems within the problem-oriented field of software engineering?

- ▶ shortly:

how can Petri nets contribute in solving Software Engineering problems?

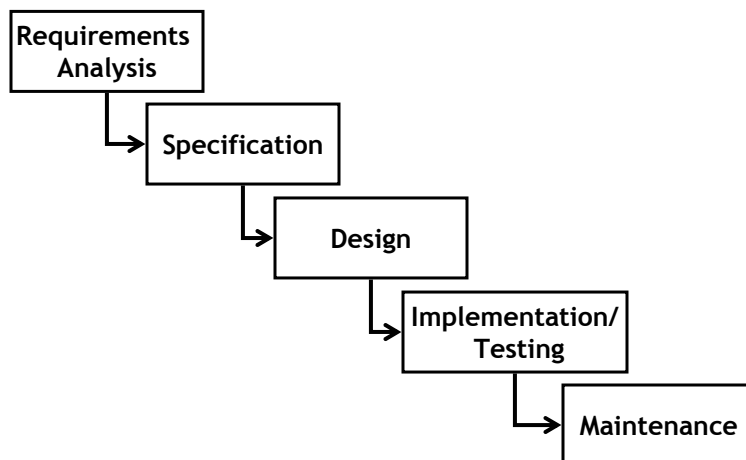
Dimensions of Software Engineering



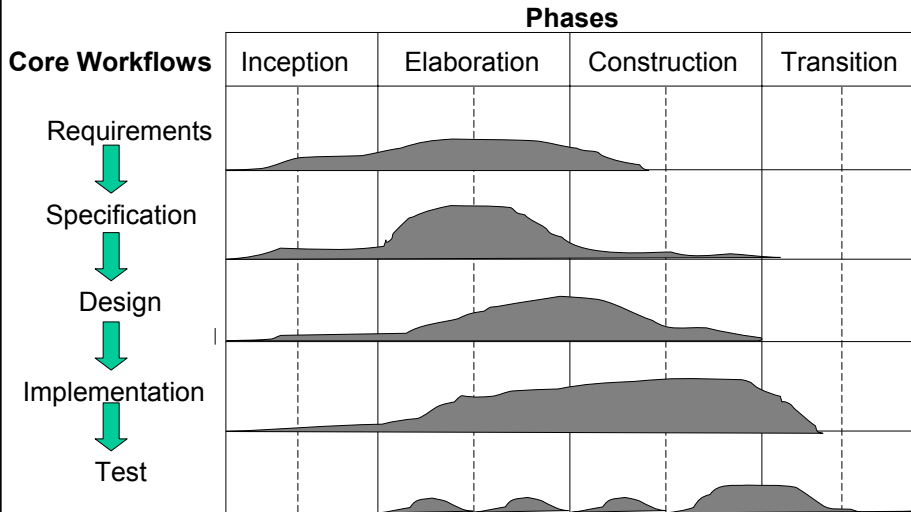
1: System Aspects

- ▶ **structure**
 - objects, data, links
 - „**who** is involved?“
- ▶ **functions**
 - change of object states, data values
 - „**what** is changed?“
- ▶ **behavior**
 - change over time
 - sequential vs. concurrent behavior
 - „**when** is something changed?“
- ▶ „**non-functional**“ properties
 - quality-of-service
 - time / resource constraints
 - „**how** is something performed?“

2: Phases



Unified Development Process



3: Activities

- ▶ analysis
- ▶ abstraction
- ▶ modeling
- ▶ construction
- ▶ refinement
- ▶ documenting
- ▶ testing
- ▶ comprehension
- ▶ refactoring
- ▶ reverse engineering
- ▶ ...

4: Stakeholders

- ▶ user
- ▶ designer
- ▶ developer
- ▶ tester
- ▶ quality assurance engineer
- ▶ consultant
- ▶ purchaser
- ▶ ...

5: Languages

- ▶ **purpose of use**
 - modeling
 - programming
 - documentation
- ▶ **language types**
 - descriptive
 - operational / executable
- ▶ **presentation styles**
 - textual
 - visual, diagrammatic
 - hybrid

6: Tools

- ▶ editor
- ▶ compiler
- ▶ interpreter
- ▶ database
- ▶ analyzer
- ▶ model checker
- ▶ debugger
- ▶ configuration management
- ▶ ...

7: System Characteristics

- ▶ interactive
- ▶ reactive
- ▶ embedded
- ▶ real-time
- ▶ control
- ▶ workflow
- ▶ distributed
- ▶ mobile
- ▶ multimedia
- ▶ web-based
- ▶ ...

Focus of the Tutorial

- **System Aspects**
 - ▶ structure, functions, behavior, non-functional properties
- **Phases**
 - ▶ requirements analysis, specification, design, implementation, testing, maintenance
- **Activities**
 - ▶ analysis, abstraction, modeling, construction, refinement, documenting, comprehension, refactoring, reverse engineering
- **Stakeholders**
 - ▶ user, designer, developer, tester, QA engineer, consultant, purchaser

Focus of the Tutorial (cont'd):

- **Languages**
 - ▶ purpose of use: modeling, programming, documentation
 - ▶ language types: descriptive, operational / executable
 - ▶ presentation styles: textual, visual/diagrammatic, hybrid
- **Tools**
 - ▶ editor, compiler, interpreter, database, analyzer, model checker, debugger, configuration management
- **System Characteristics**
 - ▶ interactive, reactive, embedded, real-time, control, workflow, distributed, mobile, multimedia, web-based

Open Research Problems: Trade-offs

- **Trade-off between**
 - ▶ usage of models for documentation
 - ▶ usage of models for analyzing system properties prior to implementation

- **Trade-off between**
 - ▶ understandability / readability of models
 - ▶ analyzability of models

- **Trade-off between**
informal and formal models

Open Research Problems: Hot Issues

- Scalability, i.e., dealing with huge systems
- Evolution of software and architectures
- Composition of heterogeneous systems
- Separation of a core modeling language from domain-specific extensions
- Semantics of high-level, heterogeneous modeling languages
- Consistency analysis of models
- Modeling of non-functional properties as, e.g., performance, real-time, safety, security, mobility
- Empirical studies, metrics
- Software Engineering Education

Essential Bibliography

- C. Ghezzi, M. Jazayeri, D. Mandrioli: *Fundamentals of Software Engineering*, 2nd Edition, Prentice Hall 2003
- I. Sommerville: *Software Engineering*, 6th Edition, Addison-Wesley 2001
- H. Van Vliet: *Software Engineering - Principles and Practice*, 2nd Edition, Wiley 2000
- A. Finkelstein (ed.): *The Future of Software Engineering*, 22nd International Conference on Software Engineering, ACM Press 2000
- S.K. Chang (ed.): *Handbook of Software Engineering and Knowledge Engineering*, Vol. 1 / 2, World Scientific, 2002

Outline

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

Organization

- Each chapter ends with
 - ▶ a summary of open research problems
 - ▶ an essential bibliography

- Each chapter presentation ends with
 - ▶ a discussion on
 - open issues
 - experiences with alternative approaches
 - future prospects for Petri nets

Part I

Specification and Analysis

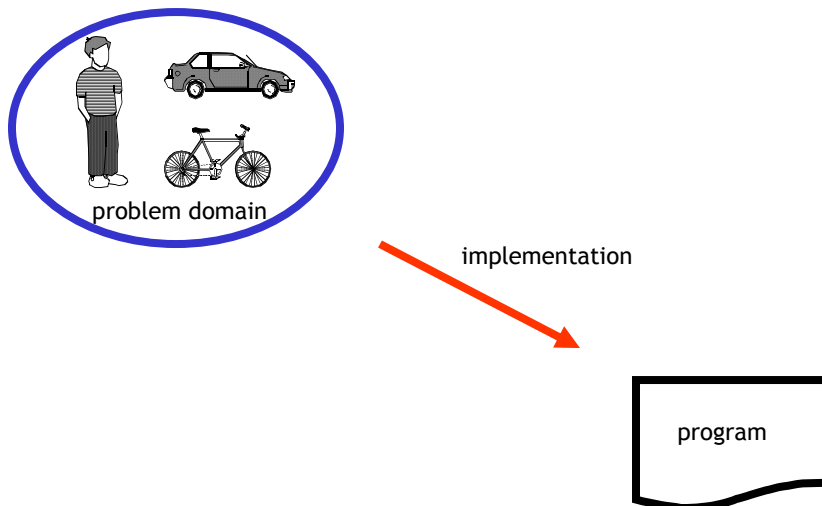


credits
This material is taken from the Tutorial on Software Engineering and Petri
presented at ICATPN 2003 by Gregor Engels and Mauro Pezzè

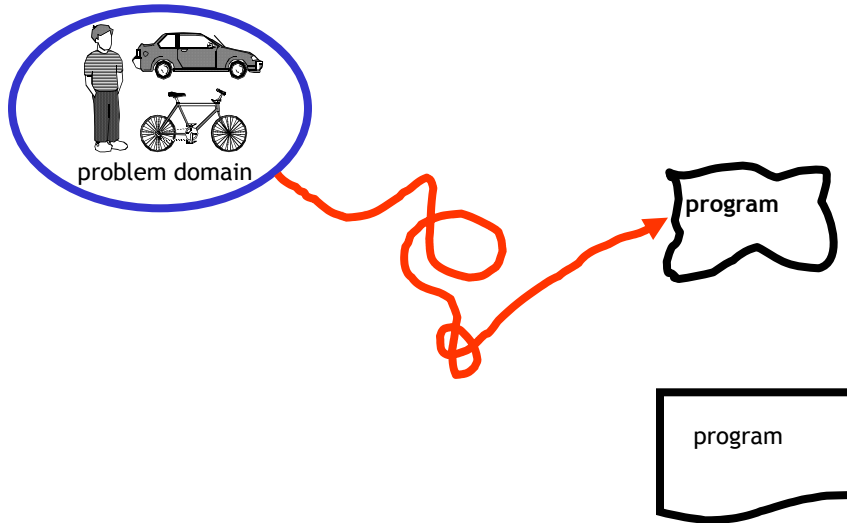
Software Specification

- Part I: Specification and Analysis
 - ▶ **Software Specification**
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

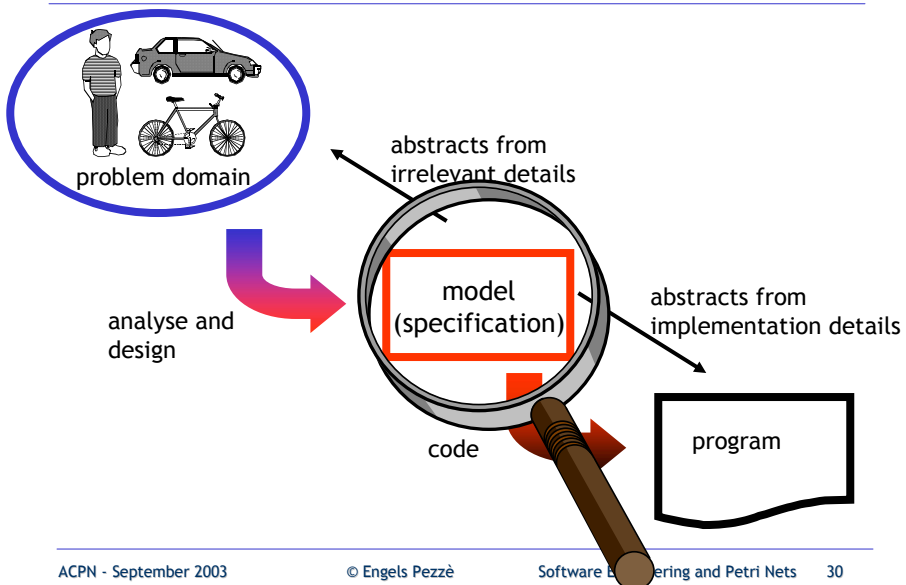
Software Development: Traditional(?) Approach



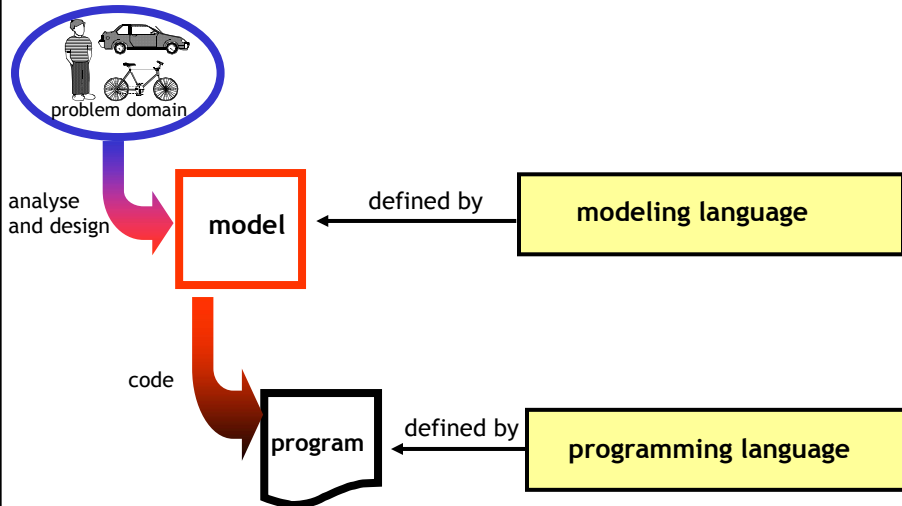
Software Development: Reality



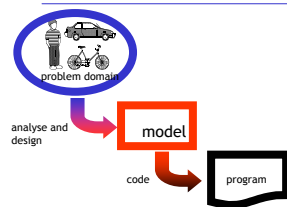
Model-centered Software Development



Languages



Requirements for a Model and a Modeling Language



Requirements for a model

- ▶ all system aspects must be described: structure, functions, behavior, non-functional properties



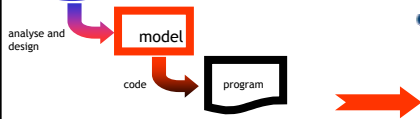
Requirements for a modeling language

- ▶ broad-spectrum language or
- ▶ union of languages

Examples

- ▶ broad-spectrum
 - natural language
- ▶ union of languages
 - Modern SA
 - UML

Requirements for a Model and a Modeling Language



Requirements for a model?

- ▶ can be deployed in all „early“ phases:
 - requirements analysis, specification, design

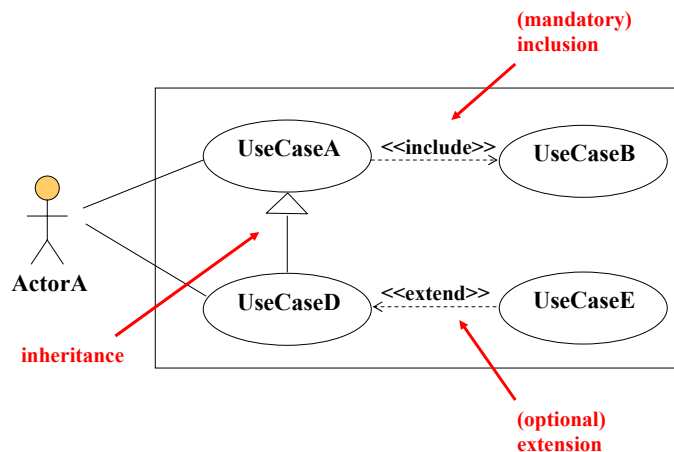
Requirements for a modeling language

- ▶ different abstraction levels of models
- ▶ stepwise refinement of models

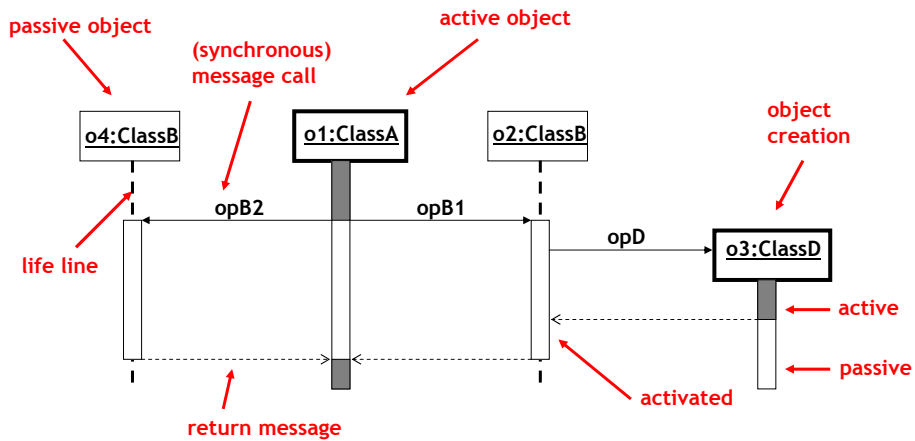
Examples

- ▶ UML Use Case Diagrams vs. UML Sequence Diagrams

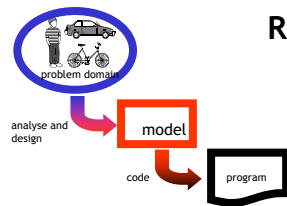
UML Use Case Diagram



UML Sequence Diagram



Requirements for a Model and a Modeling Language



Requirements for a model?

- ▶ can be used and are understood by different stakeholders:
 - user, designer, developer, tester, QA engineer, consultant
- ▶ can be used for different purposes:
 - modeling, documentation, analyzing



Requirements for a modeling language

- ▶ understandable (also) for non-computer scientists
- ▶ precise enough for developer, tester, analyzer
- ▶ analyzable by tool support

Requirements for a Model and a Modeling Language

Requirements for a modeling language

- ▶ understandable (also) for non-computer scientists (e.g. consultant)
- ▶ precise enough for developer, tester, analyzer
- ▶ analyzable by tool support

Examples

- ▶ visual, diagrammatic languages
 - data flow diagrams, ER diagrams, UML
- ▶ languages with a precisely defined syntax and semantics
 - Z, Petri Nets, algebraic specifications, process algebra
- ▶ tool support: FDR

Discussion: Understandability

An easy property!?

The gas station problem:

“if *Customers(1)* prepays before *Customers(2)*, then *Customers(2)* does not gain access to the pump before *Customers(1)*.”

[L. Dillon, Q. Yu “Oracles for Checking Temporal Properties of Concurrent Systems” *Proceedings of FSE 1994*]

A readable(?) formal model

The FIL (Future Interval Logic) specification:

$$\Box([\Diamond\Diamond\neg p, \Diamond\Diamond p | \Diamond)\neg p' \Rightarrow [\Diamond\neg p, \Diamond p | \Diamond m)\Box\neg m')$$

p = customers(1) pay

p' = customers(2) pay

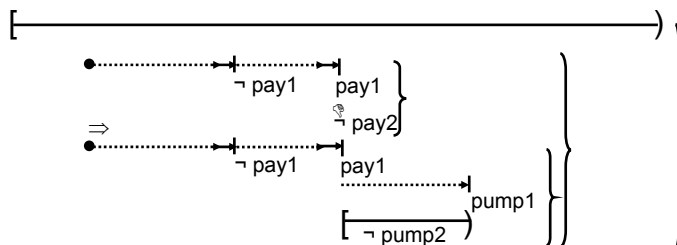
m = customers(1) access the pump

m' = customers(2) access the pump

[L. Dillon, Q. Yu "Oracles for Checking Temporal Properties of Concurrent Systems" *Proceedings of FSE 1994*]

The easy(?) graphical view

The GIL (Graphic Interval Logic) specification:
 (making FIL understandable) :



[L. Dillon, Q. Yu "Oracles for Checking Temporal Properties of Concurrent Systems" *Proceedings of FSE 1994*]

Maybe GIL is not what we want...

The railroad crossing problem
a utility property:

“The gate is up when no train is crossing”

[E. Olderog, A. Ravn, J. Skakkebak “Refining System Requirements to Program Specifications” in *Formal Methods for Real-Time Computing*, John Wiley 1996]

Maybe GIL is not what we want...

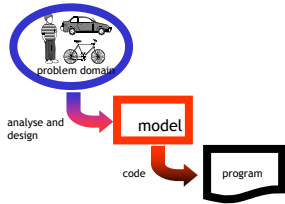
utility

RailCrossing
GRCParm

$$\square \left(\left(\lceil \neg \text{Cross} \rceil \wedge l > \xi_1 + \xi_2 \right) \Rightarrow l = \xi_2 ; \lceil g = 90 \rceil ; l = \xi_1 \right)$$

[E. Olderog, A. Ravn, J. Skakkebak “Refining System Requirements to Program Specifications” in *Formal Methods for Real-Time Computing*, John Wiley 1996]

Requirements for a Model and a Modeling Language



Requirements for a model?

- ▶ can be used for systems of different characteristics:
 - interactive, reactive, embedded, real-time, control, workflow, distributed, mobile, multimedia, web-based



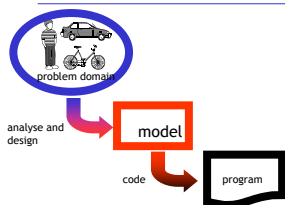
Examples

- ▶ real-time systems
 - UML-RT (?)
- ▶ interactive, reactive systems
 - statecharts

Requirements for a modeling language

- ▶ expressive language features for different system characteristics

Requirements for a Model and a Modeling Language



Requirements for a model?

- ▶ can be used with incomplete and evolving specifications:

Requirements for a modeling language

- ▶ flexibility, adaptability, incrementality

Examples

- ▶ natural languages
- ▶ families of specification languages
 - multiple syntax
 - multiple semantics

Requirements for a modeling language

- ▶ broad-spectrum language or union of languages
- ▶ different abstraction levels of models
- ▶ stepwise refinement of models
- ▶ understandable (also) for non-computer scientists
- ▶ precise enough for developer, tester, analyzer
- ▶ expressive language features for different system characteristics

- trade-off for modeling languages between
 - ▶ understandability
 - ▶ expressiveness
 - ▶ compositionality / modularity
 - ▶ completeness
 - ▶ (in)formality
 - ▶ analyzability

- role of UML in the future

- integration / coupling of modeling languages

Essential Bibliography

- G. Booch, J. Rumbaugh, I. Jacobson. *Unified Modeling Language User Guide*, Addison-Wesley, Reading, 1999.
- P. Chen. Entity-Relationship Model: *Towards a Unified View of Data*, ACM Trans. On Database Systems **1**, 1 (1976), 9-36.
- G. Engels, L. Groenewegen. *Object-Oriented Modeling: A Roadmap*. In A. Finkelstein (ed.): *The Future of Software Engineering*, 22nd ICSE, ACM Press, 2000, 103-116.
- D. Harel; *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming **8** (1987), 231-274.
- K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*. Vol. 1/2, Springer 1992/1995.
- J.M. Spivey. *The Z Notation - A Reference Manual*, Prentice Hall, 1992.
- E. Yourdon. *Modern Structured Analysis*, Yourdon Press, 1989.

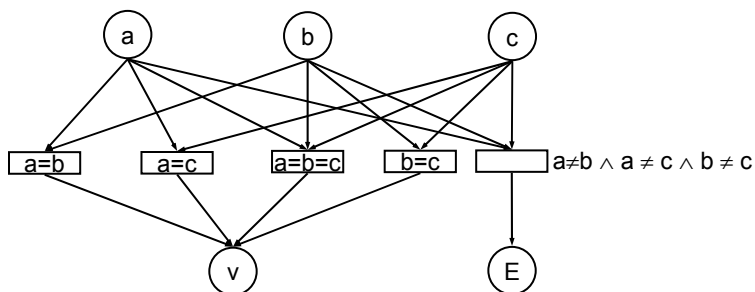
Petri Nets as Specification Means

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ **Petri Nets as Specification Means**
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

Petri Nets as Specification Means

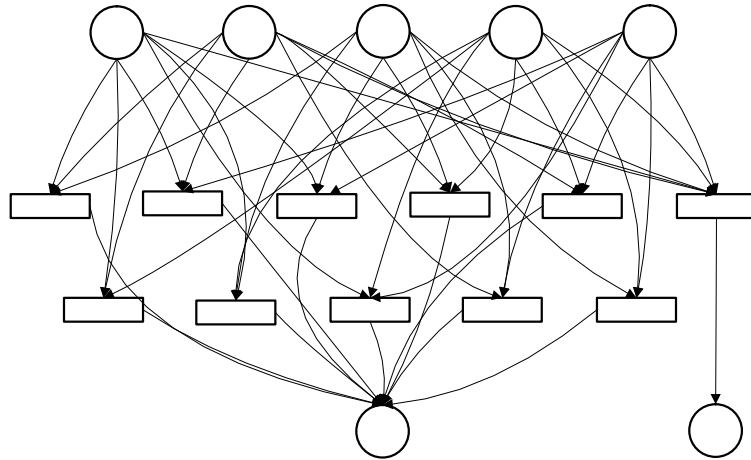
- Can we use Petri nets for specifying software systems?
 - ▶ when?
 - ▶ how?
 - ▶ which Nets?
 - ▶ for which system?
 - ▶ at which abstraction level?

Place Transition Nets as specification notation



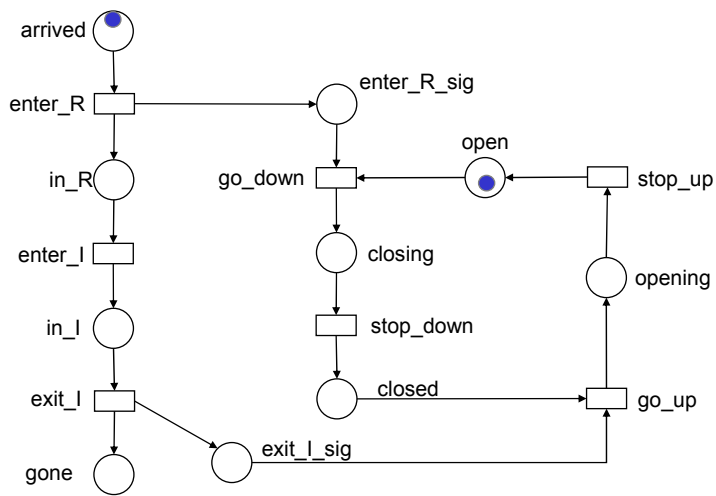
- Intuitive?
 - ▶ Can anyone tell what is this specification about?
 - ▶ need of a comprehensive (high-level) view
- Simple?
 - ▶ Maybe too simple
 - ▶ we often need additional information (annotations)

Scalability: from 2-out-of-3 to 3-out-of-5



almost incomprehensible

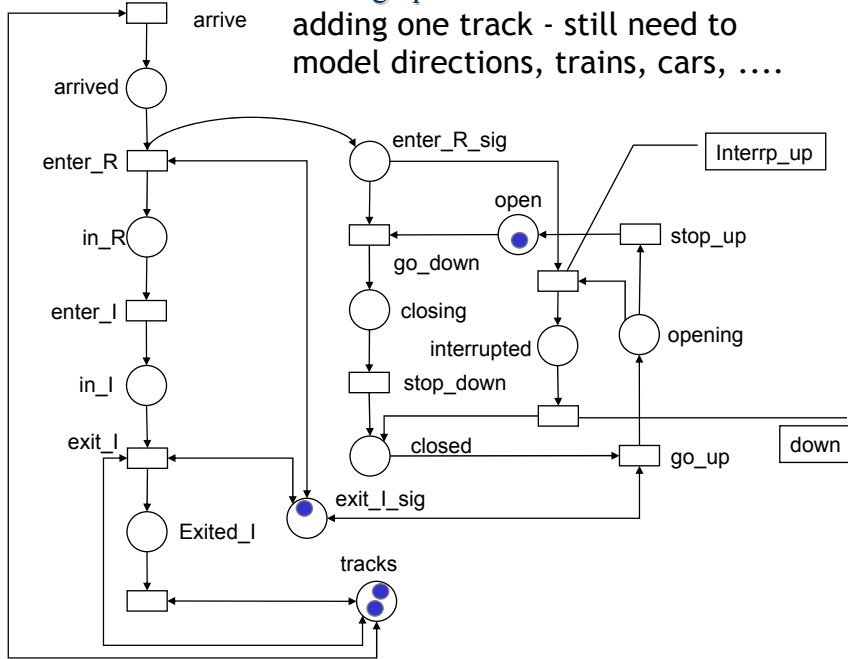
Modularity and scalability: a second example: railroad crossing: one direction, one track



a=b=c

Scaling up....

adding one track - still need to
model directions, trains, cars,

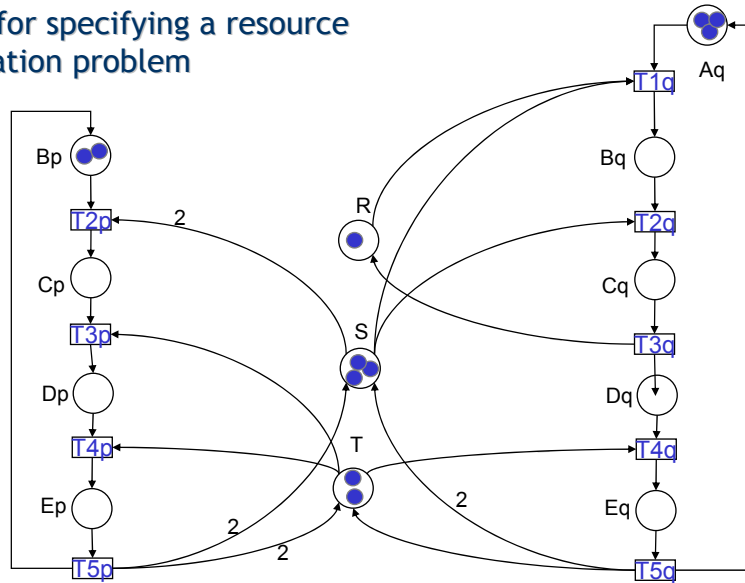


Place Transition Nets

- Seldom useful for communication among software engineers:
 - ▶ need labels to identify PN elements
 - ▶ hardly scalable
 - ▶ no support for modeling conditions
 - ▶ no time relations
 - ▶ no support for modularity
- but ...great for analysis

High-Level Petri Nets (CPNs)

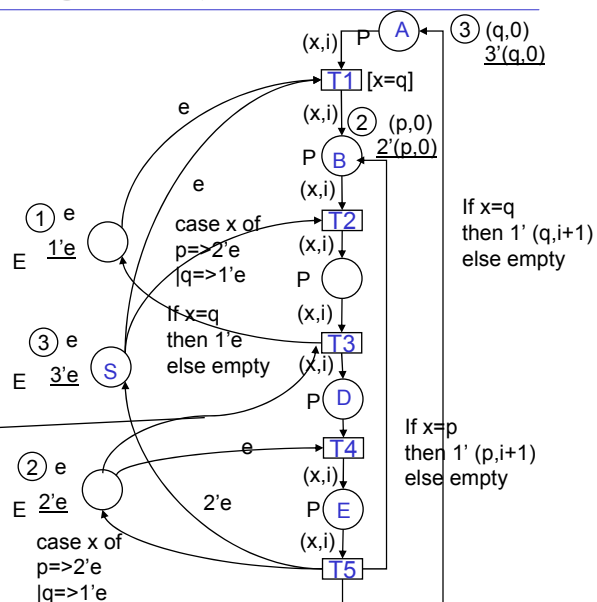
A PT for specifying a resource allocation problem



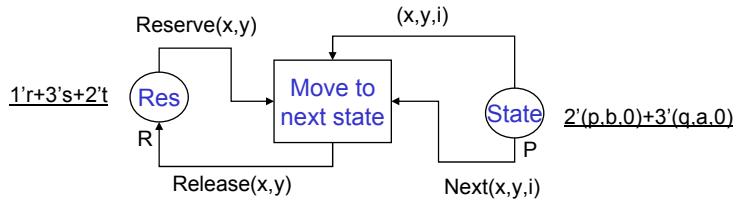
Adding Conditions (and compacting the PN)

Color U = with p|q;
Color I = int;
Color P = product U * I;
color E = with e;
var x : U;
var i : I;

If x=p
then 1'e
else empty



Scaling up: a compact (intuitive?) model



```

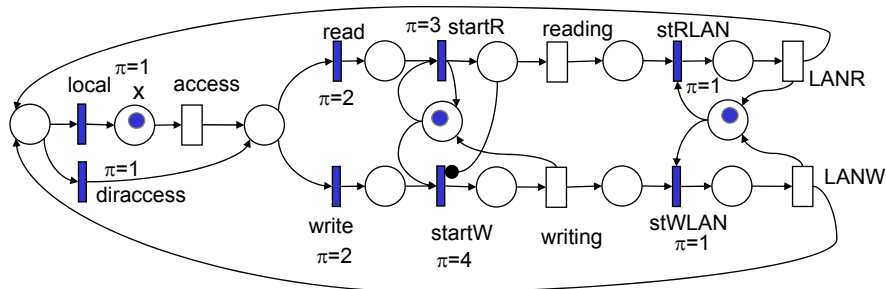
Color U = with p|q; Color S = with a|b|c|d|e;
Color I = int; Color P = product U * S * I; Color R = with r|s|t;
fun Succ(y)=case y of a=>b| b=>c| c=>d| d=>e| e=>a;
fun Next(x,y,i)=(x,if(x,y)=(p,e) then b else Succ(y), if y=e then i+1 else i);
fun Reserve(x,y) = case (x,y) of (p,b)=>2's|(p,c)=>1't |
    (p,d)=>1't|(q,a)=>1'r+1's|(q,b)=>1's|(q,d)=>1't | _=>empty;
fun Release(x,y) = case (x,y) of (p,e)=>2's+2't | (q,c)=>1'r|(q,e)=>2's+1't| _=>empty;
var x : U; var y : S; var i : I;
    
```

High-Level Petri Nets

- Colors (token identity, predicates, actions)
 - ▶ allow for modeling a larger set of systems (labels are in the model now)
 - ▶ increase compactness of the model
- Hierarchy (Hierarchical CPNs) and information hiding (OO PN)
- ▶ increase modularity and scalability
- but...
 - ▶ intuitiveness remain limited
 - ▶ some examples of industrial use
 - ▶ benefits do not balance costs
- still great for analysis

Stochastic Petri Nets (GSPNs)

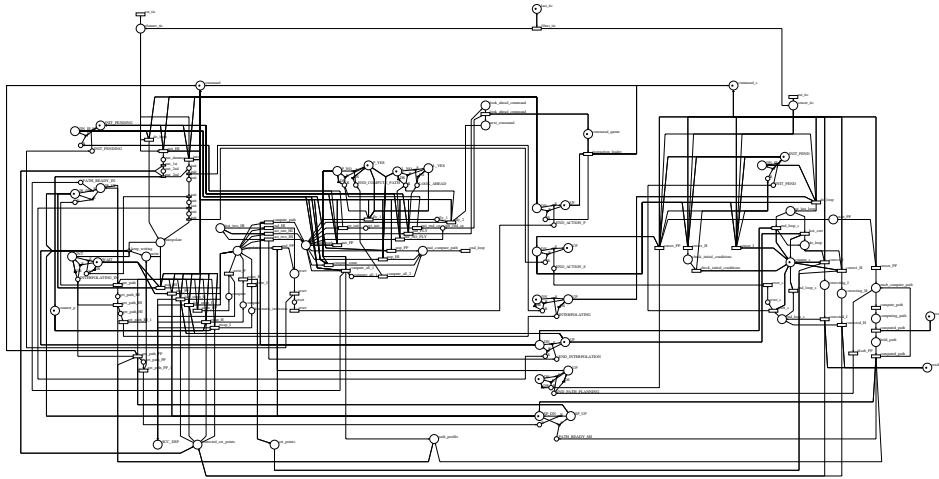
Stochastic distribution for modeling firing time



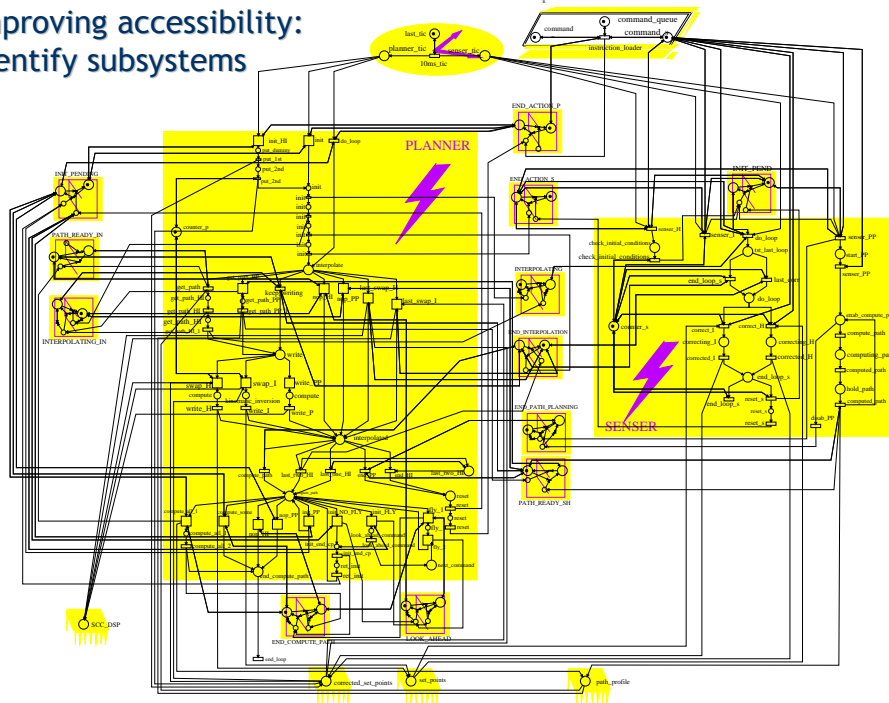
$W(\text{access}) = \lambda$		
$W(\text{read}) = 8$ (80% lettura)	$W(\text{reading}) = \mu_r$	$W(\text{LANR}) = \rho_r$
$W(\text{write}) = 2$ (20% scrittura)	$W(\text{writing}) = \mu_w$	$W(\text{LANW}) = \rho_w$
$W(\text{startR}) = 1$ (indifferente)	$W(\text{stRLAN}) = 1$	$W(\text{local}) = 9$
$W(\text{startW}) = 1$ (indifferente)	$W(\text{stWLAN}) = 1$	$W(\text{diraccess}) = 1$

GSPNs

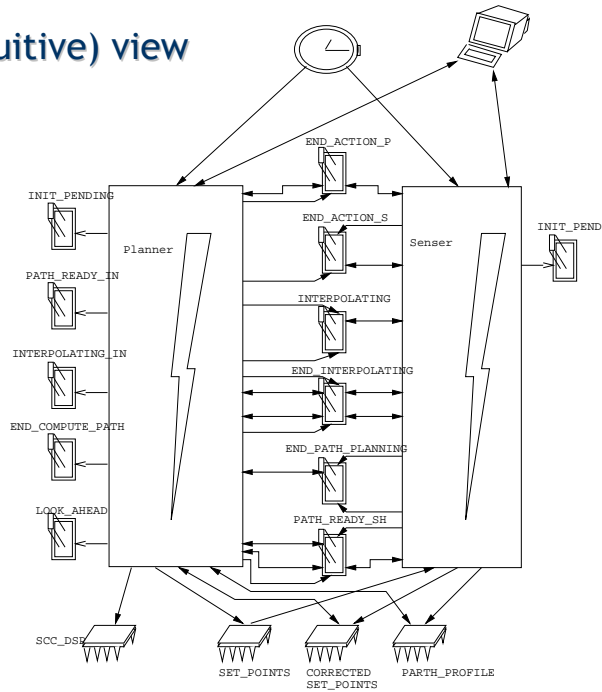
- still not completely intuitive
- but... the target is performance analysis
- used in some application domains:
 - ▶ mostly because of analysis capabilities
 - ▶ mainly not a communication means among developers



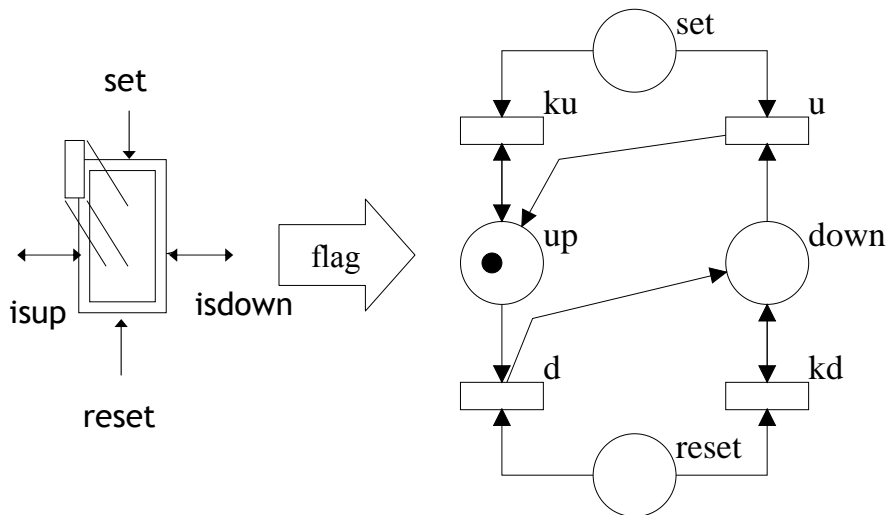
Improving accessibility: Identify subsystems



An abstract (intuitive) view



Abstracting from the net: mapping from abstract view to Petri nets



Summary and Open Research Problems

- Petri nets (as most formal methods) are useful for analysis
- Petri (PT) nets (as most formal methods) are not designed for communicating among software engineers
- Models derived from the PT nets (CPNs, GSPNs,...) allow for better modeling but do not solve all communication problems
 - ▶ industrial use is scarce and mostly for analysis (GSPNs)
- Intuitive interface languages (e.g., Control Nets) are needed for bringing analysis capabilities to the field

Essential Bibliography

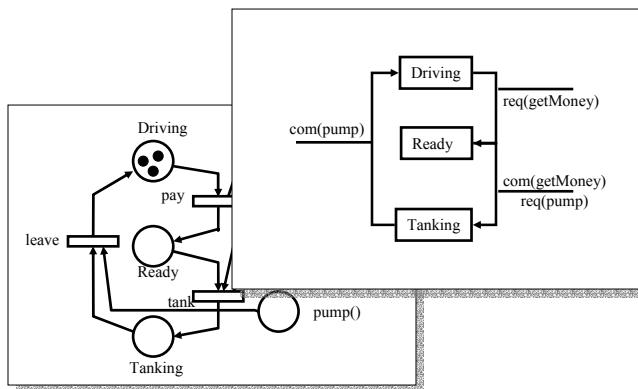
- K. Jensen,
Coloured Petri Nets,
Springer Verlag 1997, vol. 3
- M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli,
G. Franceschini,
Modelling with generalized stochastic Petri nets,
John Wiley 1995
- A. Caloini, G. Magnani, M. Pezzè,
*A Technique for Designing Robotic Control
Software Based on Petri Nets*
IEEE-Transactions on CST - Paper No. 96-101

Petri Nets as Semantic Domain

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ **Petri Nets as Semantic Domain**
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

Petri Nets as Semantic Domain

- Can we “hide” Petri nets maintaining the useful properties?
 - ▶ when?
 - ▶ how?



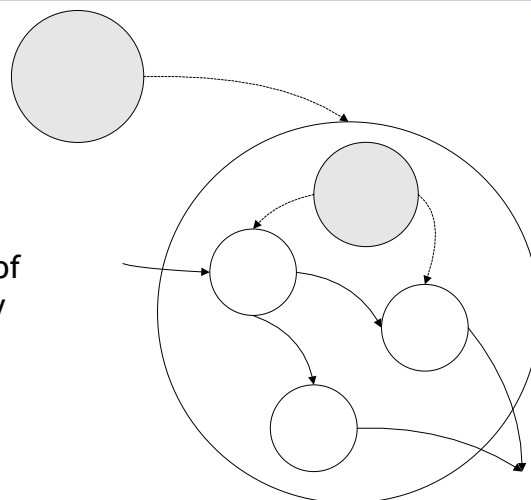
Lack of homogeneity of constructs

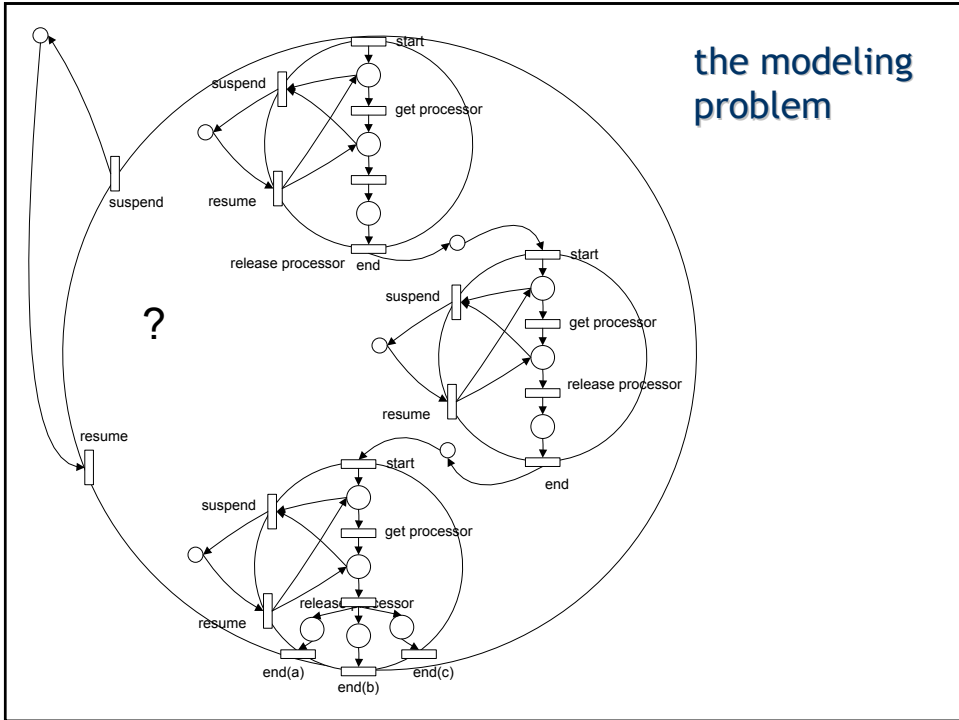
- many notations present constructs that cannot be easily mapped to Petri nets:
 - ▶ hierarchy
 - ▶ history
 - ▶ preemption
 - ▶


Hierarchy

SA/RT:

- P is hierarchically decomposed into P1, P2, P3
- the control status of P_i is determined by both Ctrl_n 2 and Ctrl_n 1
- hard to define a mapping rules







History

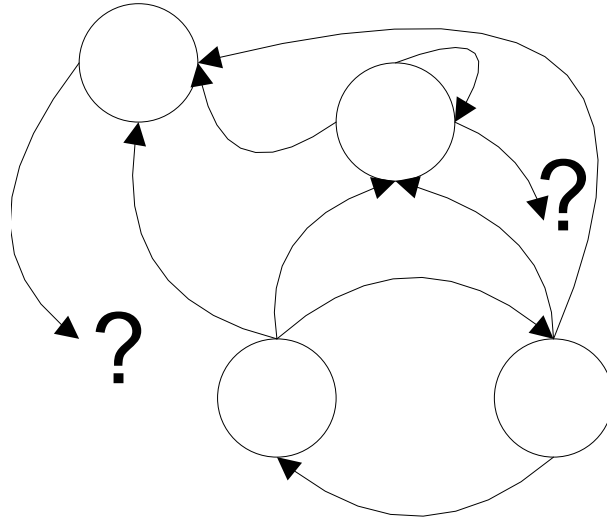
the modeling problem

Statechart history:

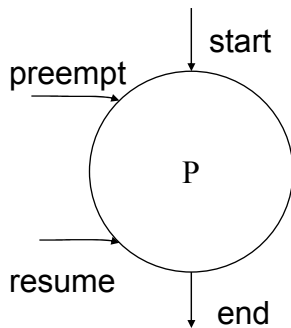
- from S1 I can go to either S21 or S22, depending on the last visited state
- if I go to S22, I can go to either S221 or S222
- hard to define mapping rules

ACPN - September 2003
© Engels Pezzè
Software Engineering and Petri Nets 72

the modeling problem



Preemption

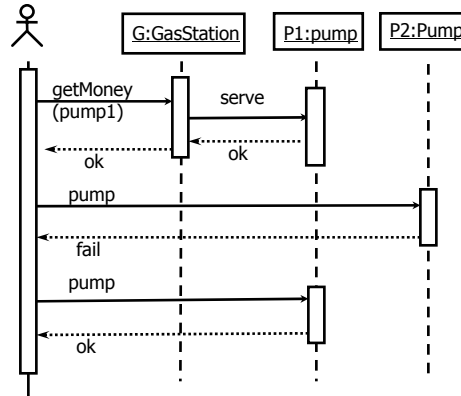
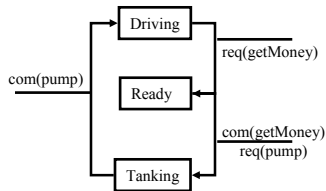
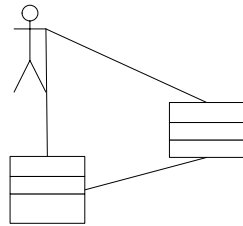


- P can be suspended and resumed
- Petri net transitions are atomic
- to model preemption we need to model
 - ▶ the different actions
 - ▶ and keep track of time

[see real-time](#)

Incompleteness and inconsistencies

UML diagrams



Incomplete and inconsistent specifications

- Inconsistencies
 - ▶ different diagrams may be used at different levels
 - ▶ Petri nets may help in revealing (removing) inconsistencies
- Incompleteness
 - ▶ dealing with incompleteness may be hard:
 - Petri nets may be meaningless/non executable
 - ▶ solving incompleteness may be impossible
 - ▶ Petri nets may help in revealing and removing incompleteness

-uses

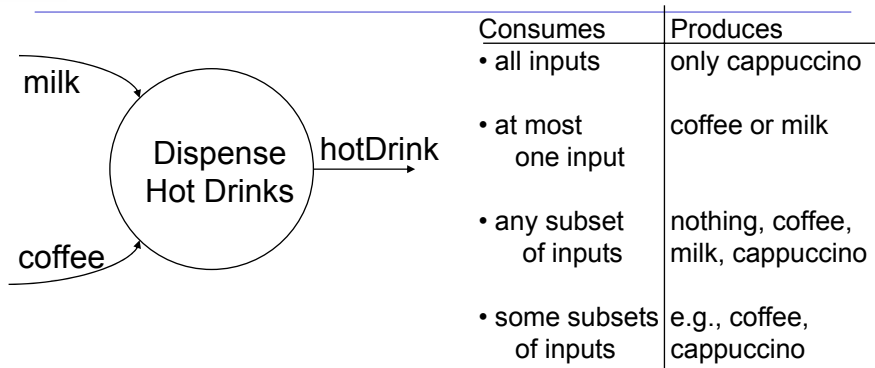
Driver

-uses

Pump

+pump()
 +serve()

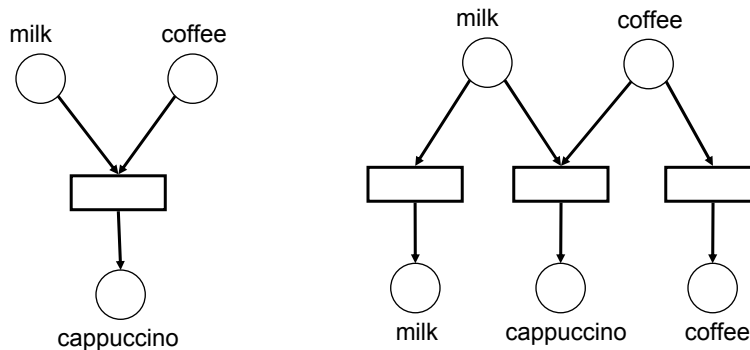
Multiple semantics



All interpretations are possible,
restricting the set of interpretations reduces the freedom of the designer

Similar set of interpretations for multiple outputs, additional orthogonal interpretations for duration, ...: the number of interpretations grows exponentially

different plausible semantics



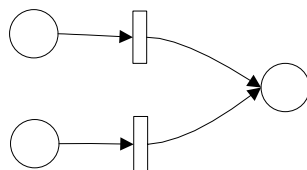
consume all inputs

consume any subset of inputs

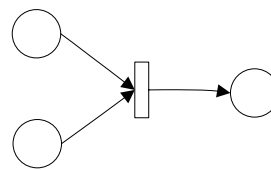
fixed vs flexible mapping

- fixed mappings
 - ▶ provide a single semantics
 - ▶ are hardly extensible
 - ▶ they require stable and commonly agreed interpretations
 - almost never
 - popular notations are ascribed many different semantics
 - experimental notations are instable
- customizable mappings
 - ▶ provide a family of semantics
 - ▶ adapt to new interpretations and extensions
 - ▶ harder to define and maintain

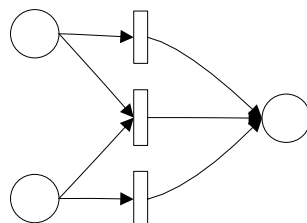
flexible rule based mapping



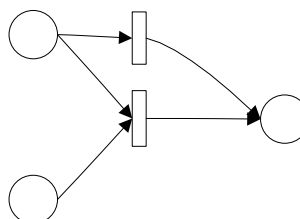
exactly one input



all inputs



all possible subsets



some possible subsets

Summary and Open Research Problems

- Petri nets (as most formal methods) can be used as semantic domain for various kinds of analysis
- need to be paired with suitable specification notations (for supporting the required level of communication)
- mapping may be hard
- informality may require variable mappings
 - ▶ interesting pairing with rule based translation approaches

Essential Bibliography

- M. D. Fraser, K. Kumar, V. K. Vaishnavi, *Informal and Formal Requirements Specification Languages: Bridging the Gap*, IEEE Transactions on Software Engineering, May 1991
- G. Richter, B. Maffeo, *Toward a Rigorous Interpretation of ESML - Extended Systems Modeling Language*, IEEE Transactions on Software Engineering, February 1993
- L. Shi, P. Nixon, *An Improved Translation of SA/RT Specification Model to High-Level Timed Petri Nets*, Proceedings of Formal Methods Europe 96, LNCS 1051
- L. Baresi, A. Orso, M. Pezzè, *Introducing Formal Methods in Industrial Practice*, Proceedings of the 20th International Conference on Software Engineering, 1997
- R.F. Paige, *A Meta-Method for Formal Method Integration*, Proceedings of FME 1997, LNCS1313

Analysis

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ **Analysis**
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

Properties of interest

- models can be used for analyzing many different properties:
 - ▶ consistency and completeness
 - ▶ correct behavior
 - ▶ performance
 - ▶ safety
 - ▶
- Models may depend on the property of interest

Analysis steps

- create a model suited for the property of interest
- express the property of interest in the derived model
- prove the property
- express the results in the application domain

- important interplay between property, model, analysis, design
 - ▶ we may impose design or modeling constraints to facilitate the proof of important properties
 - ▶ e.g., fixed upper bound of loops for easy analysis of timing properties

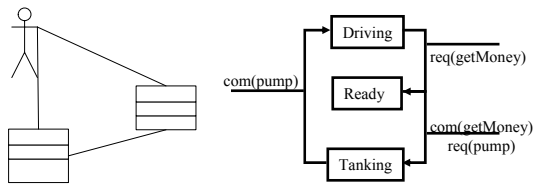
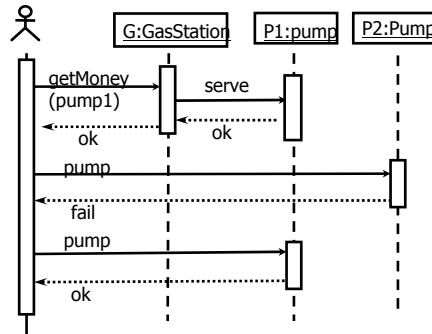
Petri nets for analysis

- Petri nets can be used to
 - ▶ identify inconsistencies and incompleteness
 - ▶ simulate specifications
 - ▶ proof liveness and safety properties
 - ▶ analyze performance
 - ▶ timing analysis (see real-time systems)

Inconsistencies and incompleteness

informal models may be

- incomplete
 - ▶ by design (e.g., UML interaction diagrams)
 - ▶ bad practice (e.g, missing classes - relations)
- inconsistent
 - ▶ semantic inconsistencies
 - ▶ contradictory views



Revealing incompleteness and inconsistencies

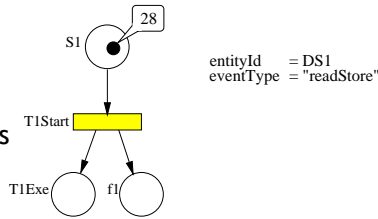
- inconsistencies of the language can be revealed by mapping the language to the formal model
 - ▶ e.g., number of inputs consumed by an SA Process
- inconsistencies in the specification can be revealed by applying the mapping to the specification
 - ▶ e.g., Statechart transition not appearing in the interaction diagrams
- incompleteness in the specification can be revealed by applying the mapping to the specifications
 - ▶ e.g., Statechart transitions not appearing in the class diagram

IMPORTANT: both mapping and model analysis may be non-trivial (the Petri net model of a specification with deadlock may be live)

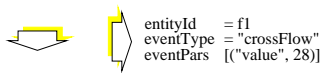
simulation

we need

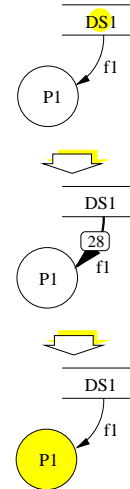
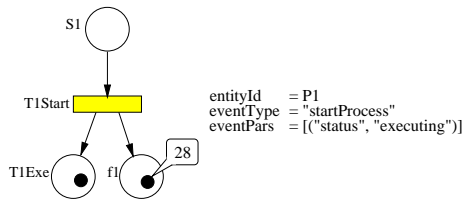
- a mapping from the specifications to the model



- a mapping from the model to the spec

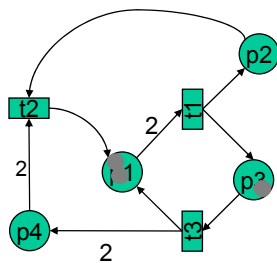


- they are not independent



proving liveness and safety properties

Matrix analysis: rows -> transitions - columns -> places



$$M_0 = (2, 0, 1, 0)$$

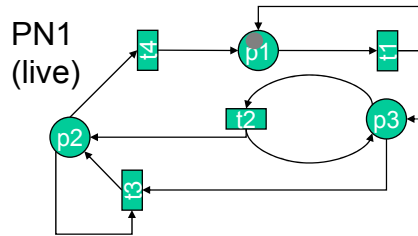
$$u = (0, 0, 1)$$

$$M_1 = (3, 0, 0, 2)$$

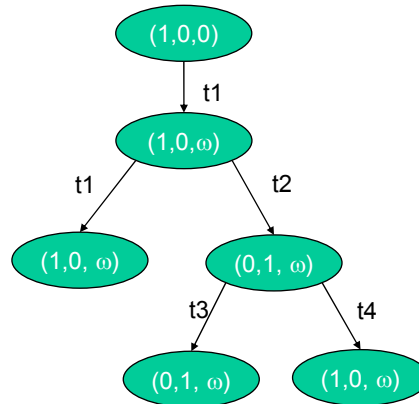
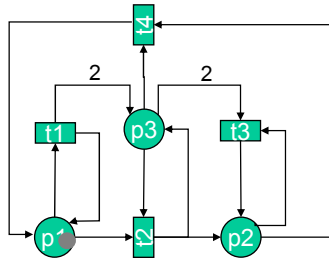
easy to apply
works only for PT nets
approximate results

$$\begin{pmatrix} 3 \\ 0 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

reachability analysis



PN2 (not live)

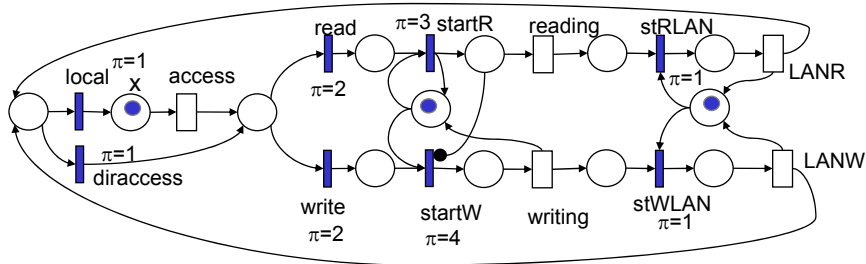


same coverage tree

reachability analysis can help in proving...

- boundedness
- safeness
- liveness
- reachability
- temporal logic properties (with model checking)
- but result must be correctly interpreted

Performance analysis



- numerical computation
- simulation
- stochastic analysis

Summary and Open Research Problems

- Petri nets (as most formal methods) can be used for various kinds of analysis
- need to match properties of interest with analysis results

Essential Bibliography

- T. Murata, Petri Nets: Properties, Analysis, and Applications, Proceedings of the IEEE, April 1989.
- M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, *Modelling with generalized stochastic Petri nets*, John Wiley 1995

Part II

Domain Specific Problems

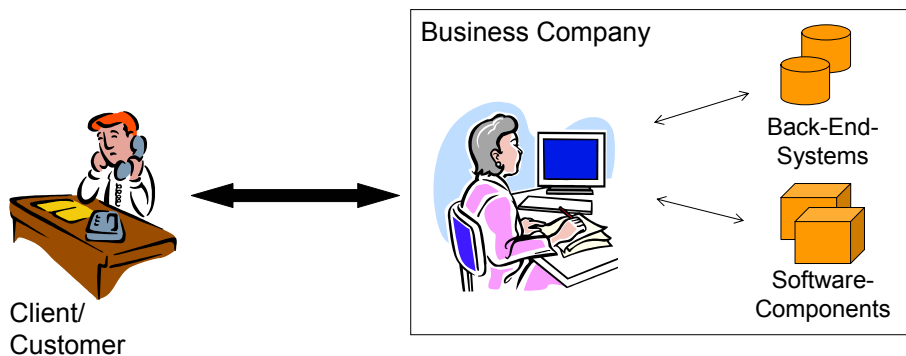


credits
This material is taken from the Tutorial on Software Engineering and Petri presented at ICATPN 2003 by Gregor Engels and Mauro Pezzè

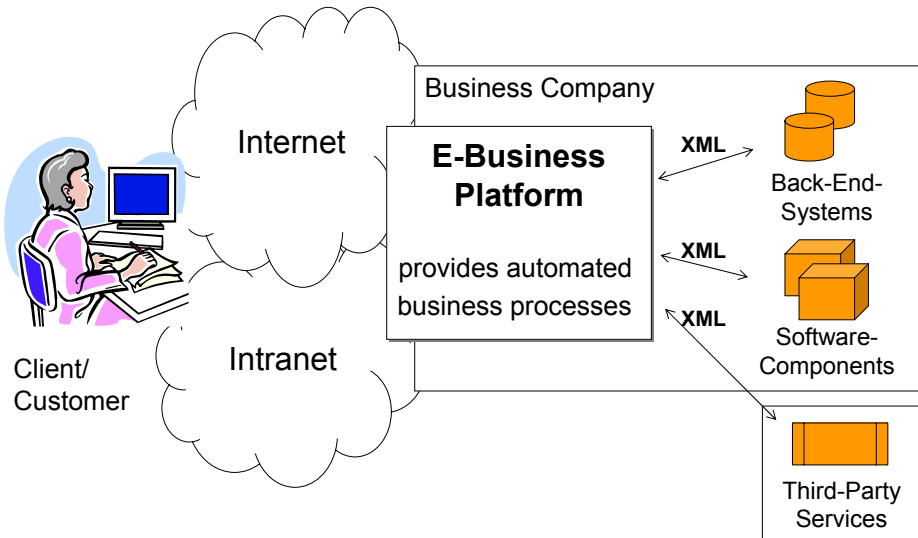
Workflow / Business Processes

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ **Workflow / Business Processes**
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

Human-Driven Integration



Process-Driven Integration



Notions

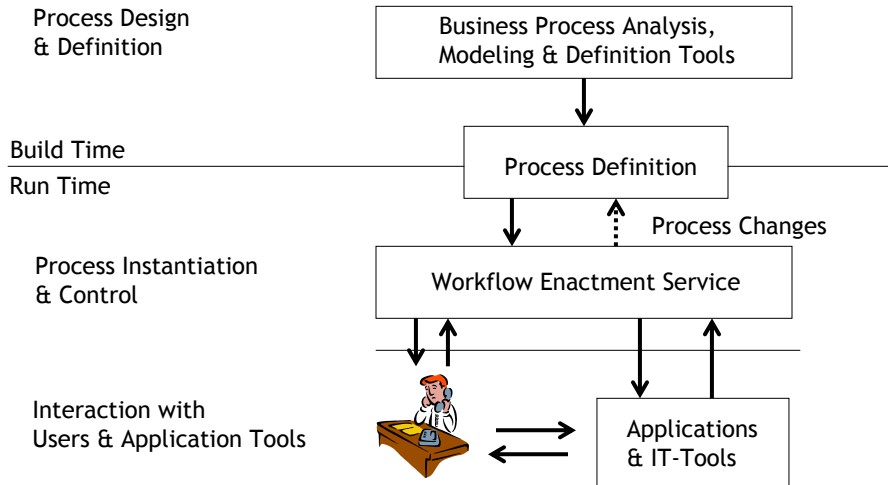
Workflow Management Coalition (WfMC)

Definitions:

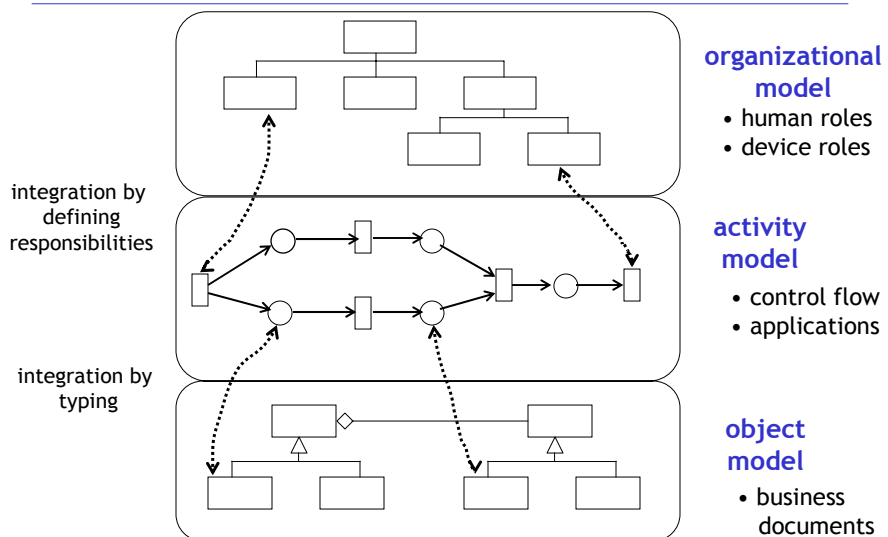
A **workflow** is the computerized facilitation or automation of a business process, in whole or part.

A **workflow management system** is a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic.

Workflow System Characteristics



Process Models: 3 building blocks



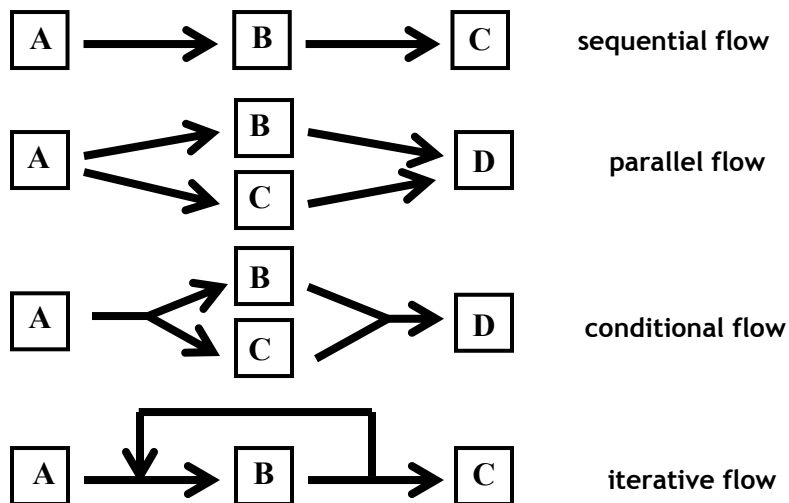
Problems

- **integrated** modeling of all aspects
 - ▶ organizational structure
 - ▶ activities
 - ▶ objects

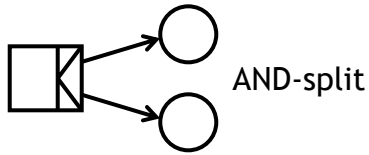
- **expressive** modeling means

- **structuring** means
 - ▶ modularity
 - ▶ reuse

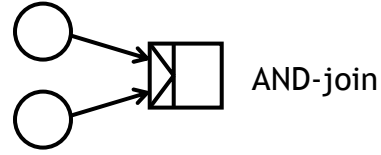
WfMC activity flow classification



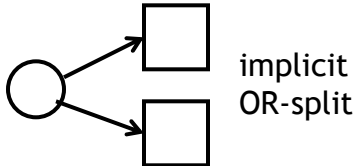
Petri net building blocks for workflow modeling



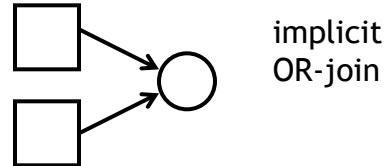
AND-split



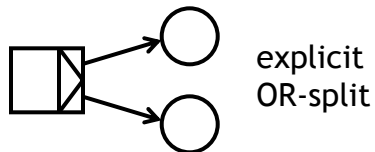
AND-join



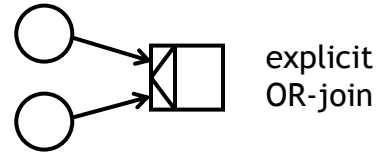
implicit
OR-split



implicit
OR-join



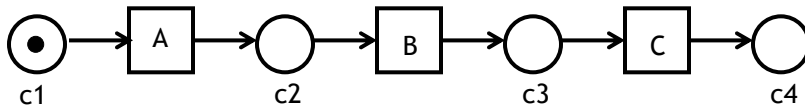
explicit
OR-split



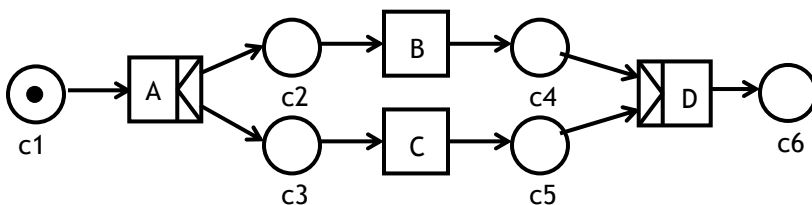
explicit
OR-join

Petri net models for WfMC flow types

sequential flow

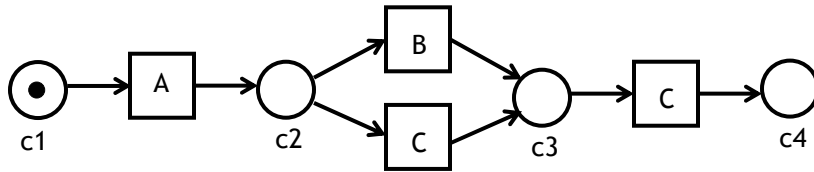


parallel flow

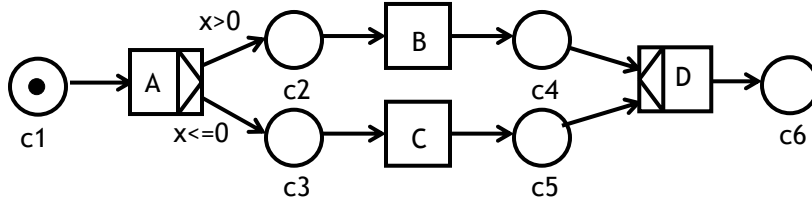


Petri net models for WfMC flow types

conditional flow with non-deterministic choice

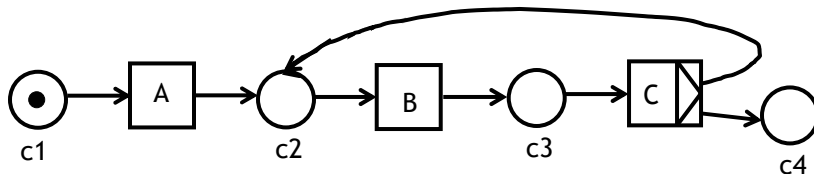


conditional flow with explicit choice based on workflow attribute x



Petri net models for WfMC flow types

iteration



Triggering

Note: Enabling of a task does not mean that the task is executed (immediately)

Solution: **Triggers** are used to start the execution of an enabled task

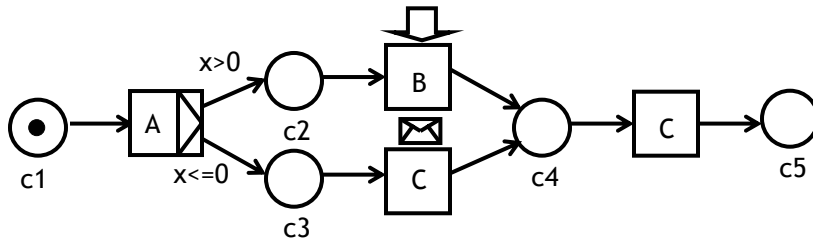
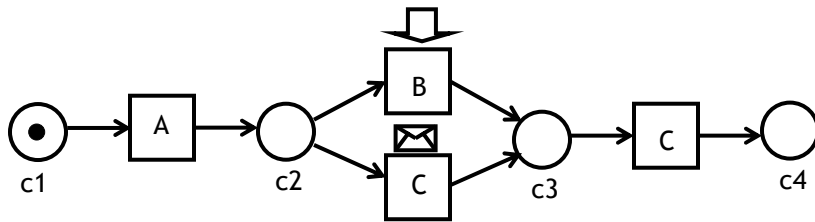
Triggering types

Four types of triggers:

- ▶ **automatic**: an enabled task is triggered the moment it is enabled
- ▶ **user**: an enabled task is triggered by a human participant
- ▶ **message**: an enabled task is triggered by an external event (i.e. message)
- ▶ **time**: an enabled task is triggered by a clock



Combining triggers and OR-splits



Workflow Patterns

• Design Patterns

- ▶ GoF: E. Gamma, R. Helm, R. Johnson, J. Vlissides - 1995
- ▶ Basic idea: „Each pattern describes a problem which occurs over and over again in our environment ...“
- ▶ Objectives:
 - Reuse of knowledge
 - Structuring of descriptions
 - improved readability, understandability of descriptions

• Workflow patterns

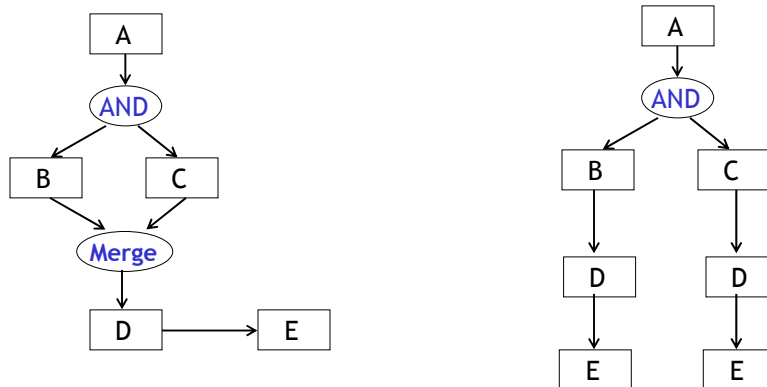
- ▶ W.M.P. van der Aalst
- ▶ <http://tmitwww.tm.tue.nl/research/patterns/>

Workflow Pattern Types

- Basic Control Flow Patterns
 - ▶ sequence, parallel split, synchronization, exclusive choice, simple merge
- Advanced Branching and Synchronization
 - ▶ multi-choice, synchronizing merge, multi-merge, discriminator
- Structural
 - ▶ arbitrary cycles, implicit termination
- Multiple Instances
 - ▶ with or without design or runtime knowledge
- State-based
 - ▶ deferred choice, interleaved parallel routing, milestone
- Cancellation

Petri Nets as Specification Means

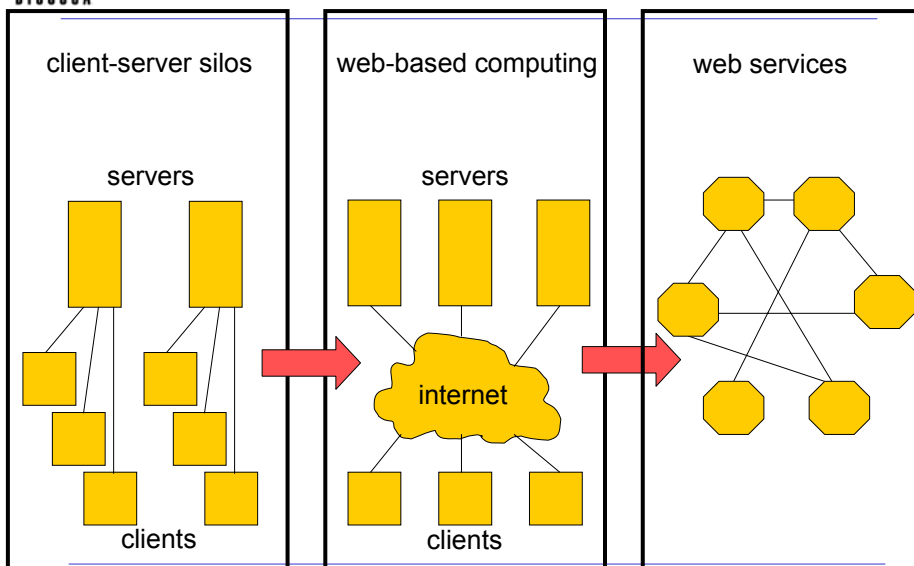
example of Petri Net model for the workflow pattern „multi-merge“



Domain-Specific Workflows: Software Process

- Software Development Processes
 - ▶ Process improvement
 - Standards: ISO 9000, Capability Maturity Model (CMM), Total Quality Management (TQM)
 - ▶ Process Modeling Languages
 - Object-oriented: SOCCA, UML
 - Rule-based: ESCAPE, OIKOS
 - Petri net-based: SLANG, FUNSOFT
 - ▶ Challenges / Open Issues
 - guidance vs. control
 - granularity of description
 - integration of languages, methods, and tools

Domain-Specific Workflows: Web Services



Summary

- Applications of Petri nets in several domains
 - ▶ workflow modeling
 - ▶ business process re-engineering
 - ▶ software process modeling
 - ▶ web service process modeling
- development of domain-specific Petri net features
- development of domain-specific patterns

Open Research Problems

- **integrated** modeling of all aspects
 - ▶ organizational structure
 - ▶ activities
 - ▶ objects
- **expressive, domain-specific** modeling means
- **structuring** means
 - ▶ modularity
 - ▶ reuse
- **role of Petri nets** as
 - ▶ specification means (i.e., workflow modeling languages)
 - ▶ semantic domain (i.e., semantic and analytical base)

Essential Bibliography

- W.M.P. van der Aalst, K.M. van Hee: *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, 2002
- W.M.P. van de Aalst: *The Application of Petri Nets to Workflow Management*. The Journal of Circuits, Systems, and Computers, 8(1):21-66, 1998
- W. Deiters, V. Gruhn: *Process Management in Practice applying the FUNSOFT Net Approach to Large-Scale Processes*, Automated Software Engineering 5, 7-25, 1998
- J.-C. Derniame, B. A. Kaba, D. Wastell (eds.): *Software Process: Principles Methodology, and Technology*, LNCS 1500, Springer, Berlin 1998
- Workflow Patterns Home Page,
<http://tmitwww.tm.tue.nl/research/patterns/>

Multimedia

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ **Multimedia**
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

Multimedia Applications

- **multimedia application**
 - ▶ computer-aided, integrated creation, manipulation, representation, storage, and communication of independent media objects
 - ▶ at least one continuous (time-dependent) and one discrete (time-independent) media object.

- **media types**
 - ▶ time independent (or discrete) media
 - e.g., text, graphics, pictures
 - ▶ time dependent (or continuous) media
 - e.g., animation, audio, music, video

Sample Multimedia Applications

- **kiosk systems**
 - ▶ on-line information terminals / desks / towers

- **electronic commerce**
 - ▶ on-line information, reservation and/or ordering systems

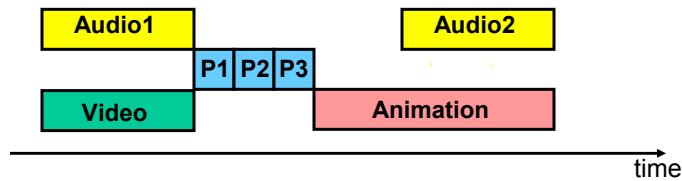
- **electronic teaching (e-Learning)**
 - ▶ on-line teaching and/or exercise systems

- **infotainment**
 - ▶ on-line television, video, game

- **electronic books, journals, lexicon**
 - ▶ on-line hypertext- / hypermedia-system

Relationship Types between Media Objects

1. logical, application-dependent relationships
2. spatial relationships
3. temporal relationships
 - time interval model



Time Interval Relations

- A **time interval** x is determined by a start point B_x and an end point E_x
- There are **13 basic relations** between 2 time intervals

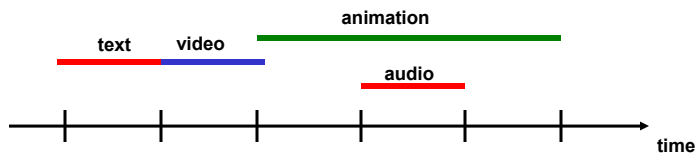
Relation	Symbols	Point-of-time notation	Examples
x before y	< >	$B_x < E_x < B_y < E_y$	
x meets y	m mi	$B_x < E_x = B_y < E_y$	
x overlaps y	o oi	$B_x < B_y < E_x < E_y$	
x finishes y	f fi	$B_x < B_y < E_y = E_x$	
x during y	d di	$B_x < B_y < E_y < E_x$	
x starts y	s si	$B_x = B_y < E_x < E_y$	
x equals y	=	$B_x = B_y < E_x = E_y$	

Petri Nets as Specification Means

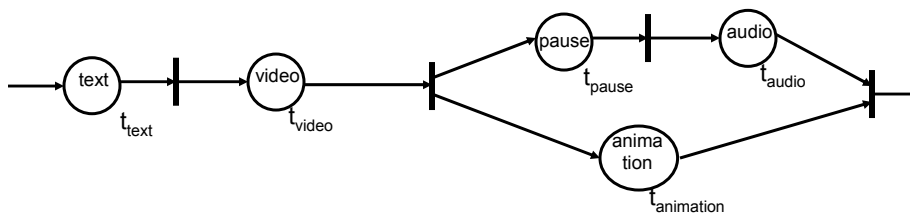
- basic idea: usage of timed Petri nets for specifying synchronization of media objects
- Definition: **Object Composition Petri Net (OCPN)**
 - ▶ directed graph $C = (T, P, A, D, R, M)$ with
 - $T = \{t_1, \dots, t_n\}$ transitions
 - $P = \{p_1, \dots, p_m\}$ places
 - $A \subseteq \{T \times P\} \cup \{P \times T\}$ connection relation
 - $D : P \rightarrow \text{Real}$ mapping of places to time intervals
 - $R : P \rightarrow \{r_1, r_2, \dots, r_n\}$ mapping of places to media objects
 - $M : P \rightarrow \text{integer}$ mapping of places to number of tokens

OCPN example

Time-interval model



OCPN model

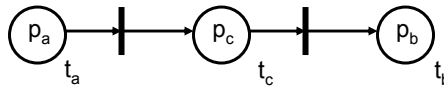


Firing Rule of OCPN

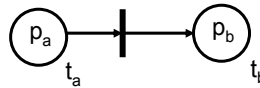
- 1) A transition t fires as soon as each input place has an unlocked token
- 2) After having fired, a transition t removes from each input place a token and adds a new token to each output place
- 3) After having received a token, a place p is active and the token is locked for the duration $D(p)$.

Modeling of time interval relations with OCPN

p_a before p_b

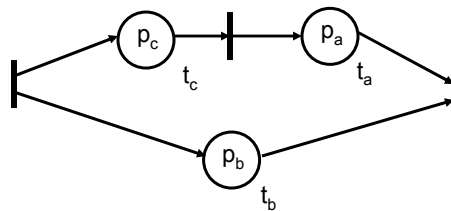


p_a meets p_b



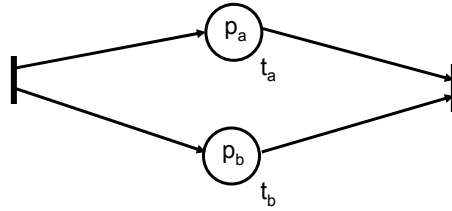
p_a during p_b

$$t_b > t_c + t_a$$



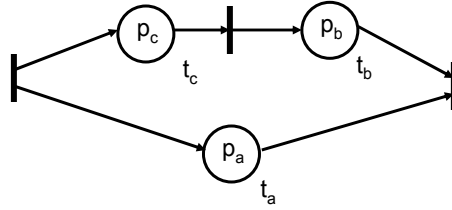
Modeling of time interval relations with OCPN

p_a starts p_b



p_a overlaps p_b

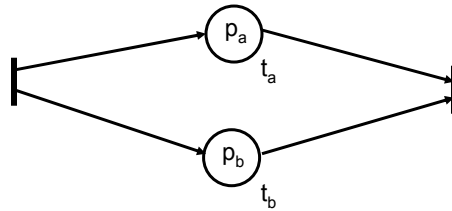
$$t_a < t_c + t_b$$



Modeling of time interval relations with OCPN

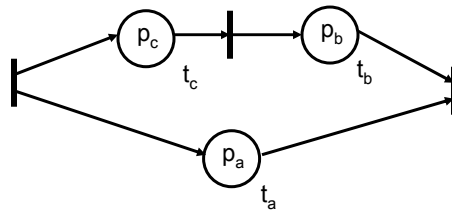
p_a equals p_b

$$t_a = t_b$$



p_a finishes p_b

$$t_a = t_c + t_b$$



Open Research Problems

- Integrated model of
 - ▶ Domain objects and links (logical relationships)
 - ▶ Layout (spatial relationships)
 - ▶ Media synchronization (temporal relationships)
- Coupling of Petri Net models with object models
- Code generation

Essential Bibliography

- Th. D.C. Little, A. Ghafoor: *Synchronisation and Storage Models for Multimedia Objects*, IEEE Journal on Selected Areas in Communications, Vol. 8, No. 3, April 1990, pp. 413 - 427
- G. Engels, St. Sauer. Object-Oriented Modeling of Multimedia Applications. In S.K. Chang (ed.): *Handbook of Software Engineering and Knowledge Engineering*, Vol. 2, World Scientific, 2002, pp. 21 - 52
- M. Vazirgiannis. *Interactive Multimedia Documents - Modeling, Authoring, and Implementation Experiences*. LNCS 1564, Springer, Berlin, 1999.
- Th. Wahl, K. Rothermel: *Representing Time in Multimedia Systems*, Proc. International Conference on Multimedia Computing and Systems, Boston, USA, May 1994, pp. 538-543

Real-Time and embedded systems

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ **Real-Time**
 - ▶ Distributed and Mobile
- Open Issues and Final Discussion

real-time embedded systems

- **real-time** system: correct functioning depends on the time (deadline) at which the results are produced
- deadlines depend on the application served by the system (**embedded**)
- **hard** real-time: late results are wrong
 - ▶ fly-by-wire
- **soft** real-time: late results produce a degraded behavior
 - ▶ letter sorting machine
- the temporal behavior of **software** depend on **hardware**

Characteristics

- importance of **timing properties**
 - ▶ deadline satisfaction (hard real time)
 - ▶ performance analysis (soft real-time)
- software cannot be abstracted from **hardware**
- analysis must model not only the system, but also the **controlled application** (and the interfaces: sensors and actuators)
- complete models include **several components**: real-time operating system, scheduler,...

Problems

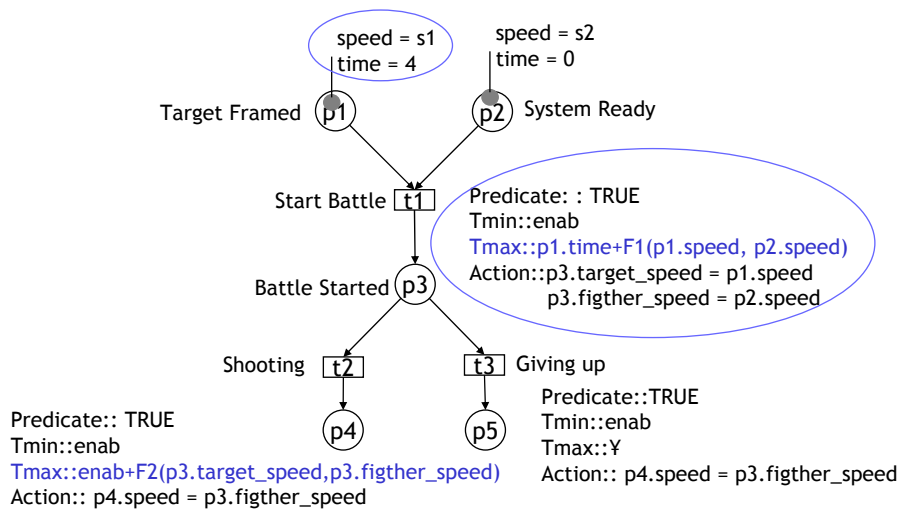
- Early validation of timing requirements
 - ▶ **early modeling**
 - ▶ **analysis and verification**
- Verification of correspondence between code and analyzed models
- Analysis of hardware and system interactions
 - ▶ **detailed models**
- Safety analysis
 - ▶ **ad hoc models**

Early modeling

Time Petri nets

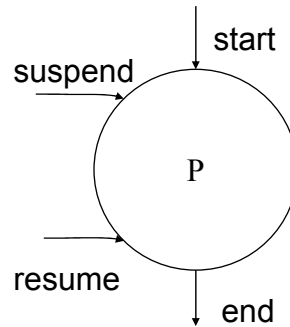
- different timing models:
 - ▶ delay on places
 - ▶ delay on transitions
 - ▶ firing time on transition
 - unique ↔ multiple
 - fixed ↔ variable
 - absolute ↔ relative
- All models are substantially equivalent
- ...but satisfy different needs

High-Level time Petri nets: time + data

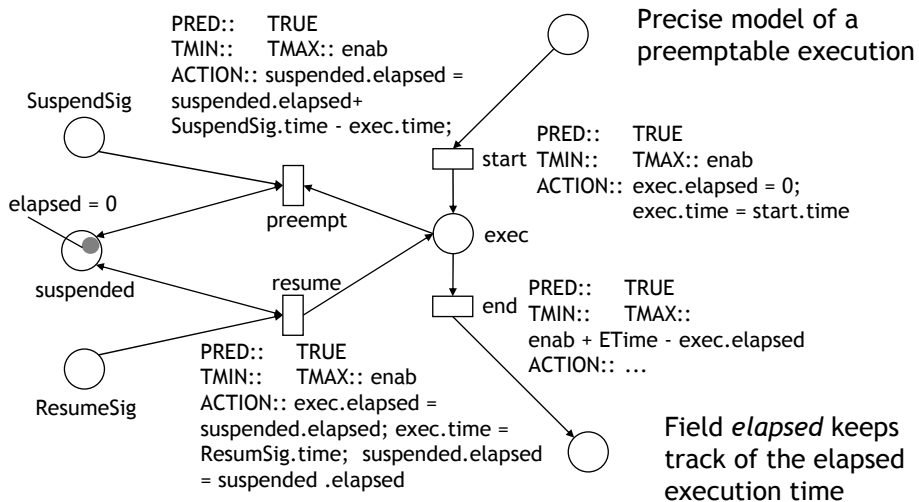


Detailed models

- models of hardware/software interactions
- example: the presence of a limited number of processors limits the parallelism
many systems are composed of preemptive processes



TPN Model of a preemptive process



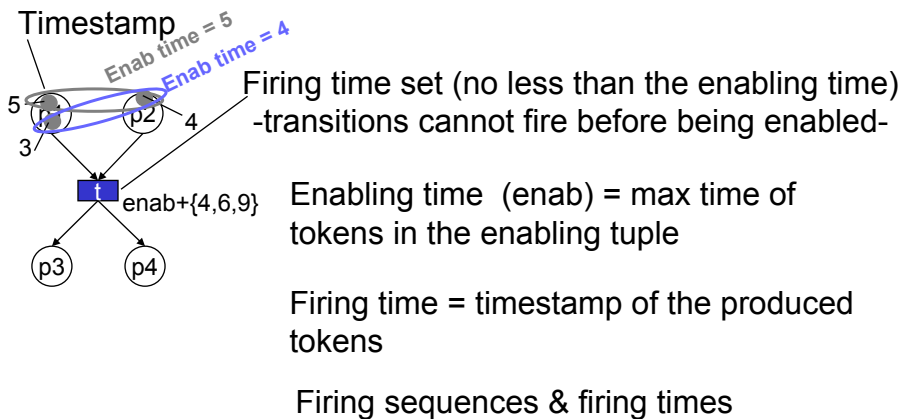
NOTE: end should put 0 back in suspended.elapsed

Petri nets as modeling means

- time extensions can model all aspects of real-time systems
- models suffer from the same problems of un-timed nets:
 - ▶ difficult to
 - write
 - read
 - maintain
 - ▶ not suitable for communication

Towards analysis and verification

a semantic model for time

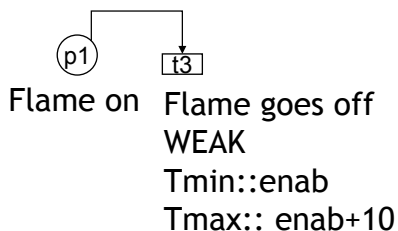


(Monotonic) Weak time semantics (M)WTS

- transitions can fire only within their firing time sets
- transitions cannot fire before being enabled
- enabled transition may not fire even if not disabled by other firings before their maximum firing time

- useful to model partially specified events
 - ▶ events that depend on non modeled components
 - e.g., an external human decision, an unexpected failure

Example of WTS: gas burner

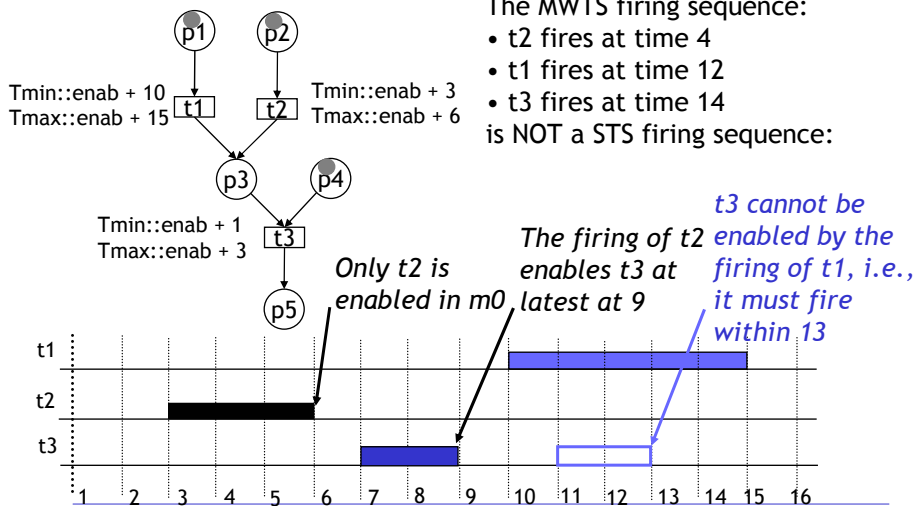


A fault (flame blows off due to the wind) is possible, but may never occur (WTS)

Strong time semantics STS

- transitions can fire only within their firing time sets
- transitions cannot fire before being enabled
- Transitions must fire within their firing sets (unless disabled by other firings before their maximum firing time)
- the most common semantics
- the default semantics in many time Petri net models

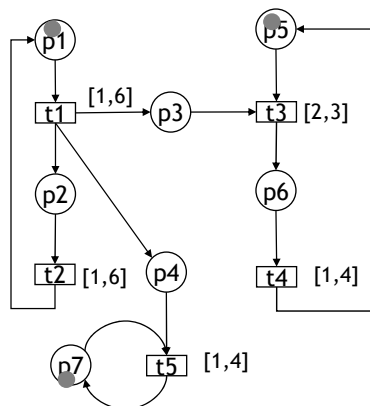
STS \neq WTS



Time reachability analysis

- Time reachability graph (TPNs):
 - ▶ state class = marking + firing domain
 - firing domain = set of inequalities
 - ▶ there exists a normal form to identify visited state
 - ▶ if firing intervals have rational boundaries the set of states is finite
 - ▶ the time reachability graph is a coverage:
 - all timed firing sequences are captured in the time reachability graph
 - some sequences of states identified in visiting the time reachability graph may not be time firing sequences

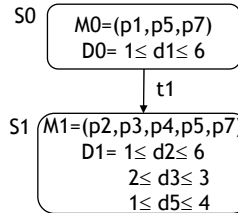
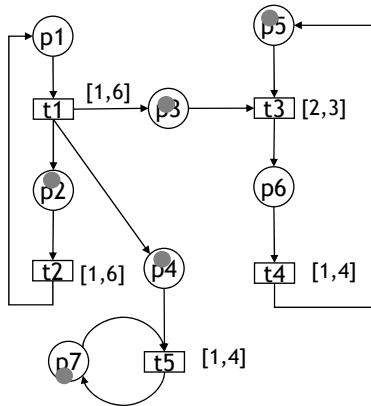
The initial state class



$$S_0 \quad \begin{cases} M_0 = (p_1, p_5, p_7) \\ D_0 = 1 \leq d_1 \leq 6 \end{cases}$$

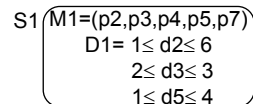
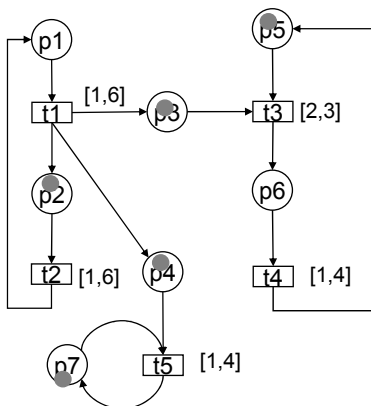
Places p1, p5 and p7 are marked, transitions t1 can fire within 1 and 6 time units from the time of the symbolic marking

The state class after the firing of t1



State class produced by the firing of transition t1 in the initial state class S0: it represents the firing intervals of the transitions enabled in all markings reachable from S0 relative to the firing time of t1

Enabling of t2 in the state class S1



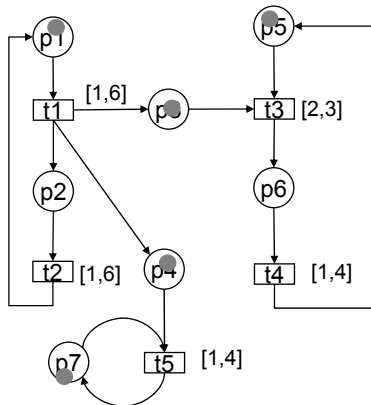
t2 can fire in S1 if

$$\begin{cases} 1 \leq d2 \leq 6 \\ 2 \leq d3 \leq 3 \\ 1 \leq d5 \leq 4 \\ d2 \leq d3 \\ d2 \leq d5 \end{cases}$$

has a solution

Let us assume that t2 fires at time $d2F$ within 1 and 3 (set of possible solutions)

Building the state class produced by the firing of t_2



New bounds for the new state:

$$2 \leq d'3 + d2F \leq 3$$

$$1 \leq d'5 + d2F \leq 4$$

equivalent to:

$$2 - d2F \leq d'3 \leq 3 - d2F$$

$$1 - d2F \leq d'5 \leq 4 - d2F$$

with

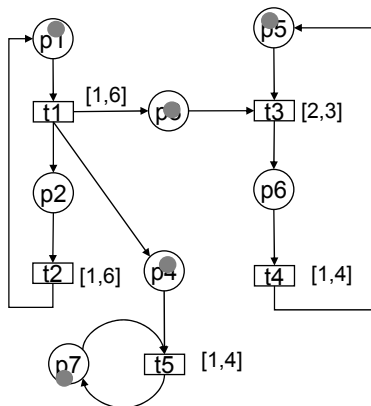
$$1 \leq d2F \leq 3$$

first two equations can be written as:

$$2 - d'3 \leq d2F \leq 3 - d'3$$

$$1 - d'5 \leq d2F \leq 4 - d'5$$

Finding new equations



eliminating $d2F$ we obtain:

$$2 - d2F \leq d'3 \leq 3 - d2F$$

$$1 \leq d2F \leq 3$$

$$\Rightarrow 0 \leq d'3 \leq 2$$

$$1 - d2F \leq d'5 \leq 4 - d2F$$

$$1 \leq d2F \leq 3$$

$$\Rightarrow 0 \leq d'5 \leq 3$$

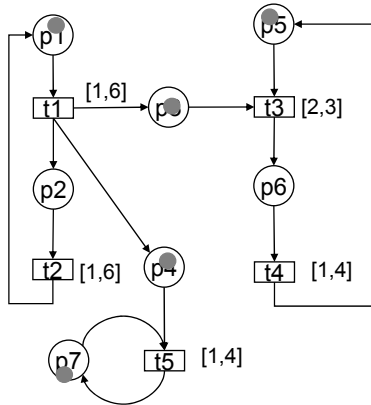
$$2 - d'3 \leq d2F \leq 3 - d'3$$

$$1 - d'5 \leq d2F \leq 4 - d'5$$

$$\Rightarrow d'5 - d'3 \leq 2$$

$$d'3 - d'5 \leq 2$$

The state class after the firing of t2



S2

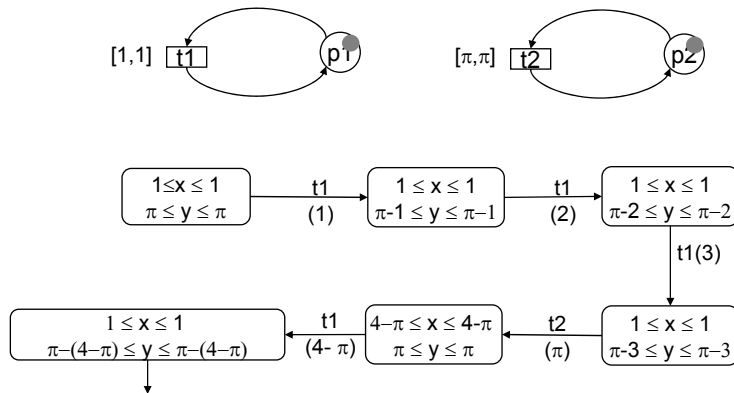
$$M1 = (p1, p3, p4, p5, p7)$$

$$D1 = \begin{aligned} &0 \leq d'3 \leq 2 \\ &0 \leq d'5 \leq 3 \\ &d'5 - d'3 \leq 2 \\ &d'3 - d'5 \leq 2 \\ &1 \leq d'1 \leq 6 \end{aligned}$$

The first 4 equations represent the translation of the former equations with respect to the firing time $d2F$ of $t2$, the new equation represent the newly enabled transition

Proving temporal properties

Time reachability graphs are finite (but approximate) if boundaries are rational numbers

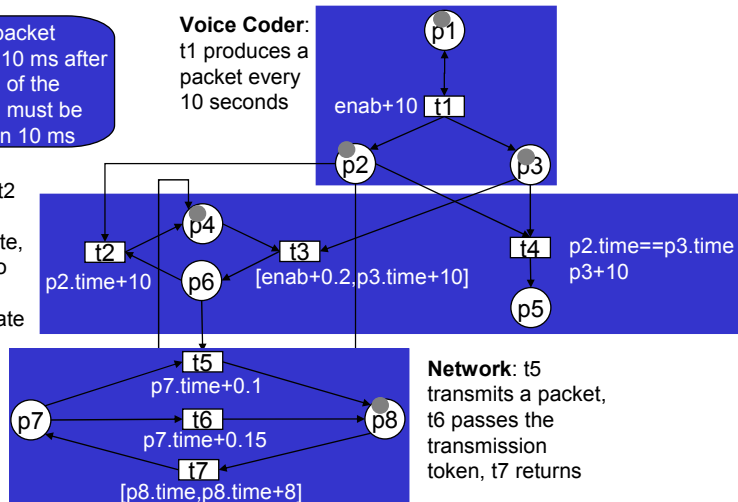


Scaling up (HLTPNs)

PROPERTY: a packet produced within 10 ms after the transmission of the previous packet, must be transmitted within 10 ms

Voice Coder: t1 produces a packet every 10 seconds

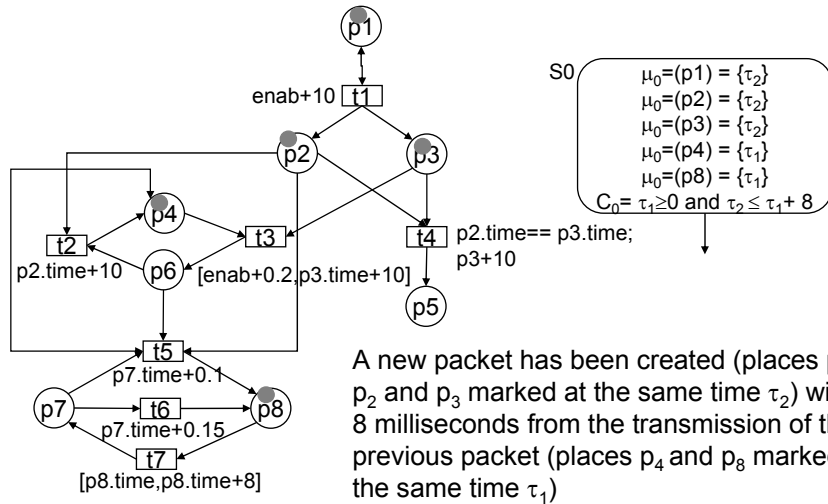
Interface: t2 moves to *receive* state, t3 moves to *ready to transmit* state



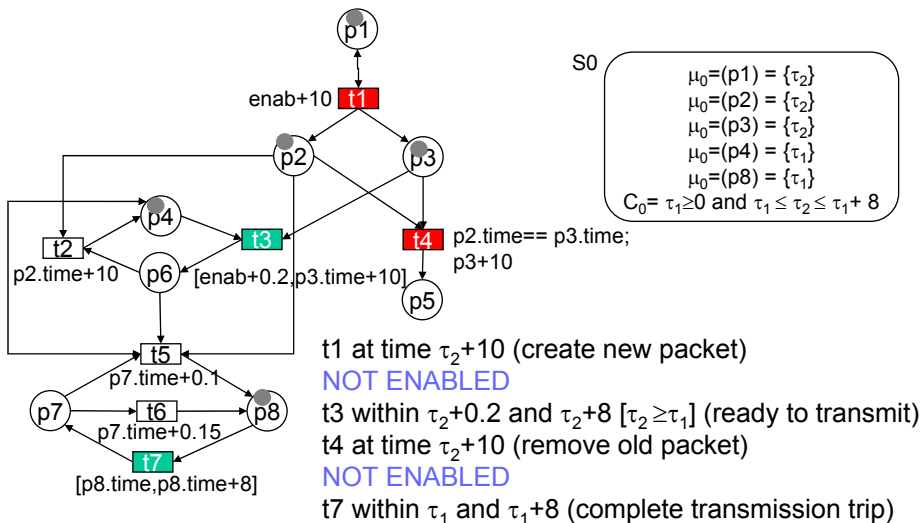
Symbolic state

- A symbolic state represents a set of possible states with the same P/T marking
- a symbolic state is a pair $[\mu, C]$, where
 - ▶ μ = symbolic marking: associates symbols (representing sets of times of tokens) to places
 - ▶ C = symbolic constraint: equations coding the relations among symbolic times

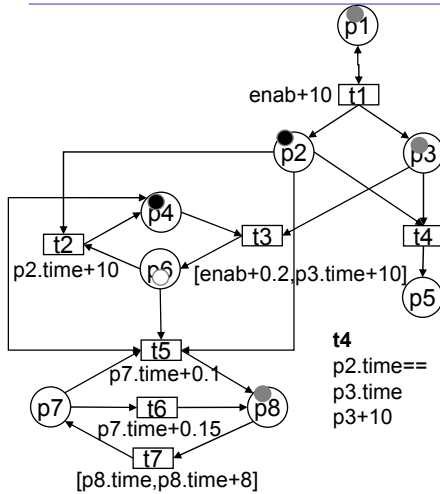
The initial symbolic state



STS enablings in S0



The symbolic state after the firing of t3



symbolic marking after the firing of t3:

$$\mu_1=(p1) = \{\tau_2\}$$

$$\mu_1=(p3) = \{\tau_2\}$$

$$\mu_1=(p6) = \{\tau_3\}$$

$$\mu_1=(p8) = \{\tau_1\}$$

symbolic constraint:

$$C_1 = \tau_1 \geq 0 \text{ and } \tau_2 \leq \tau_1 + 8$$

("inherited" from S0)

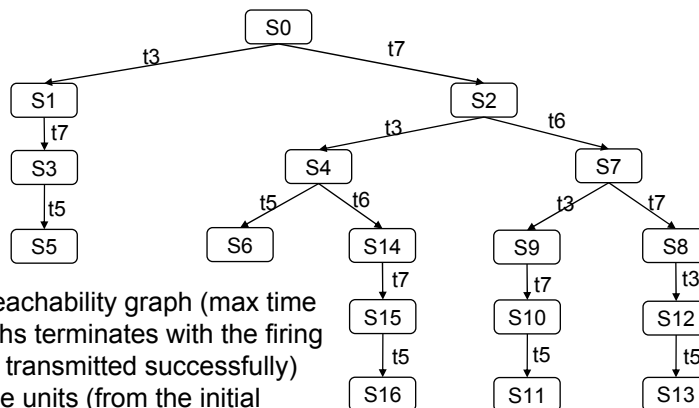
and $\tau_3 \geq \max\{\tau_2, \tau_1\} + 0.2$ and $\tau_3 \leq \tau_2 + 10$
(t3 fired within its time interval)

and $\tau_3 \leq \tau_1 + 8$

(t3 cannot fire before the maximum firing time of any other enabled transition (t7) axiom 5)

[other bounds (trivially true in this case, exclude any enabled transition from not being able to fire (axiom 4))]

Proving temporal Properties



Finite time reachability graph (max time = 10): all paths terminates with the firing of t5 (packet transmitted successfully) within 10 time units (from the initial market == packet generated within 8 time units from their previous transmission)

Petri nets as analysis means

- different reachability analysis techniques for different time Petri net extensions to deal with different characteristics and prove specific properties
- A great potential for analysis
- Timed analysis has not been fully developed yet

Software and system safety

- possible approaches:
 - ▶ hazard analysis: identify states that can lead to an accident
 - fault tree analysis
 - ▶ safety requirements specification and analysis: early analysis on specifications
 - formal specifications
 - ▶ design for safety: reduce hazards by using specific design techniques
 - ▶ testing: verifying hypothesis on the produced code

Future trends

- SOC (system on a chip):
 - ▶ the differentiation between hardware and software components is pushed forward in the development process
 - ▶ new design methodologies and new support for proving temporal properties is needed
- COTS (commercial off the shelf):
 - ▶ both hardware and software components will be increasingly used in future real time systems
- Internet connectivity:
 - ▶ real-time systems will make extensive use of the Internet

Safety future approaches

- integration of informal and formal methods
- safe product families and safe reuse
- test and evaluation of safety critical systems
- runtime monitoring

Open Research Problems

- (SOC)
 - ▶ new design methodologies and models
 - ▶ new support for proving temporal properties
- (COTS)
 - ▶ support for modeling and for modular analysis
- Internet connectivity:
 - ▶ modeling and analyzing “predictable” communication

Bibliography (Real-Time Systems)

- Nissanke, *Realtime Systems*, Prentice Hall, 1997.
- Kopetz, *Software Engineering for Real-Time: A roadmap*, The Future of Software Engineering, Finkelstein ed. ACM Press 2000
- Leveson, *Safeware: System Safety and Computers*, Addison Wesley 1995.
- Lutz, *Software Engineering for Safety: A roadmap*, The Future of Software Engineering, Finkelstein ed. ACM Press 2000

Bibliography (Timed Petri Nets)

- P. M. Merlin, D. J. Farber, Recoverability of Communication Protocols - Implications of a Theoretical Study, IEEE Transactions on Communications, September 1976.
- B. Berthomieu, M. Diaz, Modeling and Verification of Time Dependent Systems using Time Petri Nets, IEEE Transactions on Software Engineering, March 1991.
- C. Ghezzi, D. Mandrioli, S. Morasca, M. Pezzè, A Unified High-Level Petri Net Formalism for Timed Critical Systems, IEEE Transactions on Software Engineering, February 1991.
- N.G. Leveson, J.L. Stolzy, *Safety Analysis Using Petri Nets*, IEEE Transaction on Software Engineering, March 1987.

Distributed and Mobile Systems

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ **Distributed and Mobile**
- Open Issues and Final Discussion

Distributed and mobile systems

- distributed systems:
 - ▶ fixed network of nodes
 - ▶ reliable and predictable communication means
 - ▶ stable environment: possible failures of nodes can be repaired
- mobile systems:
 - ▶ the network is not fixed
 - ▶ program may evolve and change structure
 - ▶ communication facilities may vary
 - ▶ unstable environment: nodes may appear and disappear
 - ▶ pushed by new technology
 - raises many interesting new problems
 - technological problems: providing adequate support for mobility
 - design problems: how to identify processes, how to establish a communication
 - modeling problems: how to model the new mobile framework
 - analysis and verification problems: hoe to test and verify mobile systems

Physical vs logical mobility

- physical mobility: movement of mobile hosts in the environment (e.g., palm, laptop,..)
- logical mobility: mobile units (code and state) that migrate among hosts

new problems

- meeting of unknown nodes (devices)
 - ▶ *discovering who else is there*
- coordination
 - ▶ *different forms of synchronization (e.g., coordination of groups of moving devices)*
 - ▶ *explicit and implicit coordination mechanisms*
 - ▶ *different notions of co-location*
 - *physical*
 - *logical*
 - *programs on the same node may be physically contiguous, but logically far apart*
- communication
 - ▶ *differences in bandwidth or power may lead to one way communication*

Modeling challenges

- need to model space and components (code fragments or physical devices) that move through it
- modeling components:
 - ▶ unit of execution (mobile agent)
 - ▶ but with code-on-demand the unit of execution does not actually move (more difficult to model)
- modeling location
 - ▶ implicit in the structure of the system
 - ▶ give explicitly (tight with the model of component)
- modeling context
 - ▶ context-aware computing is an important characteristics of mobile systems
 - ▶ includes: resources, services, other components available in the current location

Open Research Problems

- modeling location
 - ▶ implicit in the structure of the system
 - ▶ give explicitly (tight with the model of component)
- modeling context
 - ▶ context-aware computing is an important characteristics of mobile systems
 - ▶ includes: resources, services, other components available in the current location

Bibliography

- Roman, Picco, Murphy, *Software Engineering for Mobility: A roadmap*, The Future of Software Engineering, Finkelstein ed. ACM Press 2000
- Fuggetta, Picco, Vigna, *Understanding Code Mobility*, IEEE TSE, 1998
- Cardelli, Gordon, *Mobile Ambients*, Theoretical Computer Science 2000.

Open Issues and Final Discussion

- Part I: Specification and Analysis
 - ▶ Software Specification
 - ▶ Petri Nets as Specification Means
 - ▶ Petri Nets as Semantic Domain
 - ▶ Analysis
- Part II: Domain-specific Problems
 - ▶ Workflow / Business Processes
 - ▶ Multimedia
 - ▶ Real-Time
 - ▶ Distributed and Mobile
- **Open Issues and Final Discussion**

Open Issues and Final Discussion

- Specification and Analysis
 - ▶ trade-off for modeling languages (understandability, expressiveness, compositionality, modularity, completeness, (in)formality, analyzability)
 - ▶ role of UML in the future
 - ▶ integration / coupling of modeling languages
 - ▶ intuitive Petri nets models
 - ▶ flexible mappings to Petri nets as semantic domain
 - ▶ pairing Petri nets with suitable notations
 - ▶ matching analysis and problems

Open Issues and Final Discussion

- Workflow / Business Processes
 - ▶ integrated modeling of all aspects (organizational structure, activities, objects)
 - ▶ expressive, domain-specific modeling means
 - ▶ structuring means (modularity, reuse)
 - ▶ role of Petri nets as specification means and semantic domain
- Multimedia
 - ▶ Integrated model of logical, spatial, and temporal relationships
 - ▶ Coupling of Petri Net models with object models
 - ▶ Code generation
- Real-Time
 - ▶ (SOC) new design methodologies and models and new support for proving temporal properties
 - ▶ (COTS) new support for modeling and for modular analysis
 - ▶ Internet connectivity: modeling and analyzing “predictable” communication
- Distributed and Mobile
 - ▶ modeling location
 - ▶ modeling context