3. Modelling Dynamic Behavior with Petri Nets

1

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
Gruppe Softwaretechnologie
http://st.inf.tu-dresden.de
Version 12-1.0, 10/17/12

1) Basics
   1) Elementary Nets
   2) Colored Petri Nets
2) Patterns in Petri Nets
3) Refactorings
4) Composability of Colored Petri Nets
5) Parallel Composition with CPN
6) Application to modelling

Obligatory Readings

2

- Balzert 2.17
- Or Ghezzi Chap 5
- or (not enough in Pfleeger):
- W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of workflow task structures: A petri-net-based approach. Information Systems, 25(1): 43-69, 2000.
- Kurt Jensen, Lars Michael Kristensen and Lisa Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. Software Tools for Technology Transfer (STTT). Vol. 9, Number 3-4, pp. 213-254, 2007.
- J. B. Jörgensen. Colored Petri Nets in UML-based Software Development – Designing Middleware for Pervasive Healthcare. www.pervasive.dk/publications/files/CPN02.pdf
- Web portal "Petri Net World" http://www.informatik.uni-hamburg.de/TGI/PetriNets/

Literature

3

- K. Jensen: Colored Petri Nets. Lecture Slides http://www.daimi.aau.de/~kjensen Many other links and informations, too
  - www.daimi.aau.dk/CPnets the home page of CPN. Contains lots of example specifications. Very recommended
- K. Jensen, Colored Petri Nets. Vol. I-III. Springer, 1992-96. Landmark book series on CPN.
- T. Murata. Petri Nets: properties, analysis, applications. IEEE volume 77, No 4, 1989.
- W. Reisig. Elements of Distributed Algorithms – Modelling and Analysis with Petri Nets. Springer. 1998.
- W. Reisig, G. Rozenberg: Lectures on Petri Nets I+II, Lecture Notes in Computer Science, 1491+1492, Springer.
- J. Peterson. Petri Nets. ACM Computing Surveys, Vol 9, No 3, Sept 1977
- http://www.daimi.au.dk/CPnets/intro/example_indu.html

Relationship of PN and other Behavioral Models

4

- P.D. Bruza, Th. P. van der Weide. The Semantics of Data-Flow Diagrams. Int. Conf. on the Management of Data. 1989
  - http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.9398
- E.E.Roubtsova, M. Aksit. Extension of Petri Nets by Aspects to Apply the Model Driven Architecture Approach. University of Twente, Enschede,the Netherlands
- Other courses at TU Dresden:
  - Entwurf und Analyse mit Petri-Netzen
  - Lehrstuhl Alg. u. log. Grundlagen d. Informatik
  - Dr. rer. nat. W. Nauber
  - http://wwwtcs.inf.tu-dresden.de/~nauber/eapn10add.html

Goals

- Understand untyped and Colored Petri nets (CPN)
- Understand that CPN are a verifiable and automated technology for safety-critical systems
- PN have subclasses corresponding to finite automata and data-flow graphs
- PN can be refined, then reducible graphs result

The Initial Problem

- You work for PowerPlant Inc. Your boss comes in and says:

Our government wants a new EPR reactor, similarly, in the way Finland has it. How can we produce a verified control software? We need a good modelling language. Assembler would be too bad...

UML does not work...

How do we produce software for safety-critical systems?

Interesting Projects with Safety-Critical, Parallel Embedded Software

- Arial
  - The WITAS UAV unmanned autonomously flying helicopter from Linköping http://www.ida.liu.se/~marwz/papers/ICAPS06_System_Demo.pdf
- Automotive
  - Prometheus: driving in car queues on the motorway
    - http://www.springerlink.com/content/j06n312r36805683/
- Trains
  - www.railcab.de Autonomous rail cabs
  - www.cargocab.de Autonomous cargo metro
    - http://www.cargocap.de/files/cargocap_presse/2005/2005_01_12%20kruse.pdf
  - http://www.rubin-nuernberg.de/ Autonomous mixed metro

Application Areas of Petri Nets

- Model introduced by C.A. Petri in 1962.
  - Ph.D. Thesis: "Communication with Automata".
  - Over many years developed within GMD (now Fraunhofer, FhG)
  - PNs describe explicitly and graphically: Conflict/non-deterministic choice, concurrency
- Reliable software (quality-aware software)
  - PetriNets can be checked on deadlocks, liveness, fairness, bounded resources
- Safety-critical software that require proofs
  - Control software in embedded systems or power plants
- User interface software
  - Users and system can be modeled as separate components
- Hardware synthesis
  - Software/Hardware co-design

Application Area I: Behavior Specifications in UML

- Instead of describing the behavior of a class with a statechart, a CPN can be used
- CPN have several advantages:
  - They model parallel systems naturally
  - They are compact and modular, can be reducible
  - They lend themselves to aspect-oriented composition, in particular of parallel protocols
  - They can be used to generate code, also for complete applications
  - UML statecharts, data flow diagrams, and activity diagrams are special instances of CPN
- Informal: for CPN, the following features can be proven
  - Liveness: All parts of the net do never get into a dead lock, i.e., can always proceed
  - Fairness: all parts of the net are equally “loaded” with activity
  - K-boundedness: the data that flows through the net is bound by a threshold
  - Deadlock-freeness: the net does not stop (deadlock)

Application Area II: Contract checking (Protocol Checking) for Components

- Petri Nets describe behavior of components (dynamic semantics)
  - They can be used to check whether components fit to each other
- Problem: General fit of components is undecidable
  - The protocol of a component must be described with a decidable language
  - Due to complexity, context-free or -sensitive protocol languages are required
- Algorithm:
  - Describe the behavior of two components with two CPN
  - Link their ports
  - Check on *liveness* of the unified CPN
  - If the unified net is not live, components will not fit to each other…
- Liveness and fairness are very important criteria in safety-critical systems

3.1 Basics of PN

Petri Net Classes

Predicate/Transition Nets: simple tokens, no hierarchy.

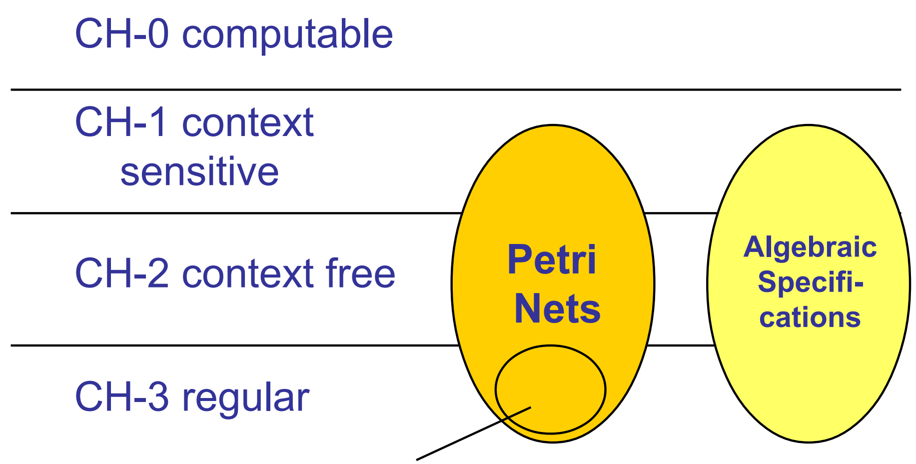
Place-Transition Nets: multiple tokens

High Level Nets: structured tokens, hierarchy

There are many other variants, e.g., with timing constraints

Language Levels

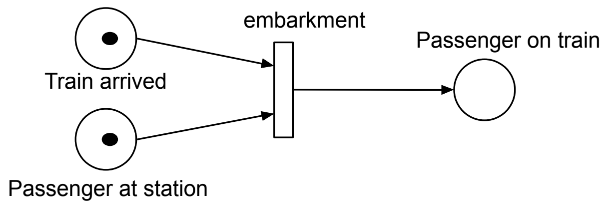
- PN extend finite automata with indeterminism
  - Asynchronous execution model (partial ordering)



Finite state machines are PN with finite reachability graph

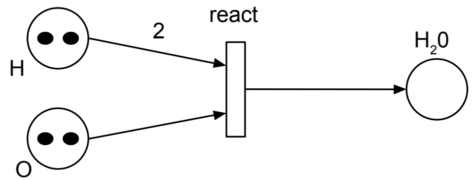
Elementary Nets: Predicate/Transition Nets

- A **Petri Net (PN)** is a directed, bipartite graph over two kinds of *nodes,* namely *places* (circles) and *transitions* (bars or boxes)
- An *elementary PN* is with boolean tokens, i.e., one token per place (bound of place = 1)
  - aka basic, predicate/transition nets (PTN), condition/Event nets
  - The presence of a token in a place means that the condition or predicate is true
  - The *firing* of a transition means that from the input predicates the output predicates are concluded
  - Thus elementary PN can model simple forms of logic



Integer Place/Transitions-Nets

- An *integer PN* is a directed, weighted, bipartite graph over places and transitions with *integer* tokens, i.e., places may contain several tokens, and a *capacity* (*bound* = k)
  - *k* tokens in a place indicate that *k* data items are available
  - *M(p)* is the number of tokens in place *p*
- A *marking* assigns to each place a nonnegative integer
  - A marking is denoted by *M*, an m-vector where m is the number of places.
  - A PN has a *initial marking*, M_0.
- Arcs have *cardinalities (weights)* to show how many tokens they transfer

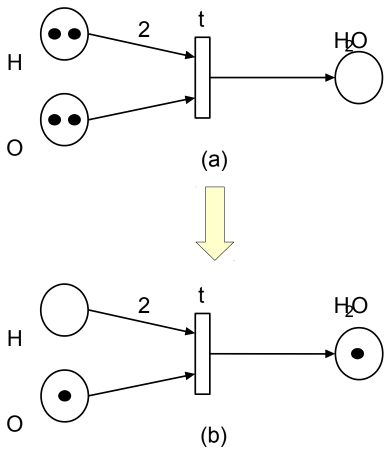


Here: initial marking $M_0(2,2,0)$

Formal Transition Enabling and Firing

In a PN a state is changed according to the following *transitions firing rule*:

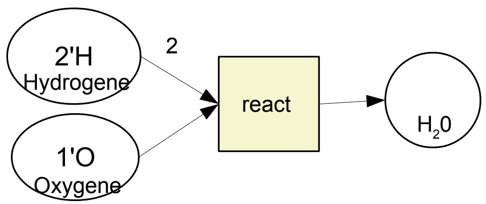
- A transition t is *enabled* if
  - each input place *p* of *t* is marked with at least *w(p,t)* tokens, where *w(p,t)* is the weight of the arc from *p* to *t*
  - The output place can be filled
- An enabled transition may or may not fire.
- A *firing* of an enabled transition removes *w(p,t)* tokens from each input place *p* to *t*, and adds *w(t,p)* tokens to each output place *p* of *t*, where *w(t,p)* is the weight of the arc from *t* to *p*.



(a) *t* is enabled.
(b) *t* has been fired.

High-Level Nets

- A *high-level PN (colored PN)* allows for *typed* places and arcs
  - For types, any DDL can be used (e.g., UML-CD)
- High-level nets are modular
  - Places and transitions can be refined
  - A Colored Petri Net is a reducible graph
- The upper layers of a reducible CPN are called *channel agency nets*
  - Places are interpreted as channels between components



3.1.1 Elementary Nets (Predicate/Transition Nets)

Meaning of Places and Transitions in Elementary Nets

- Predicate/Transition (Condition/Event-, State/Transition) Nets:
  - Places represent conditions, states, or predicates
  - Transitions represent the firing of events:
    - if a transition has one input place, the event fires immediately if a token arrives in that place
    - If a transition has several input places, the event fires when all input places have tokens
- A transition has *input* and *output* places (pre- and postconditions)
  - The presence of a token in a place is interpreted as the condition is true

Formal Definition of a Place/Transition Net

- A PN is a 5-tuple, P = (P, T, F, W, M_0) with

$P = \{p_1, p_2, ..., p_m\}$ is a finite set of places,

$T = \{t_1, t_2, ..., t_m\}$ is a finite set of transitions,

$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),

$W : F \rightarrow \{1, 2, 3, ...\}$ is a weight function,

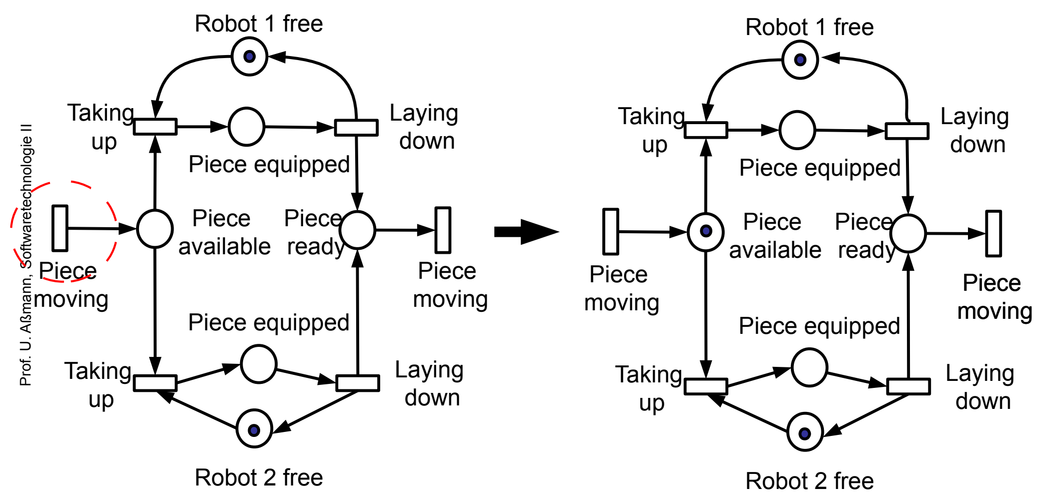
$M_0 : P \rightarrow \{0, 1, 2, 3, ...\}$ is the initial marking, (if img(P) = {0,1}, we have a elementary net, otherwise an integer net)

$P \cap T = \emptyset, \; P \cup T \neq \emptyset$

A PN structure N = (P, T, W) without any specific initial marking is denoted N

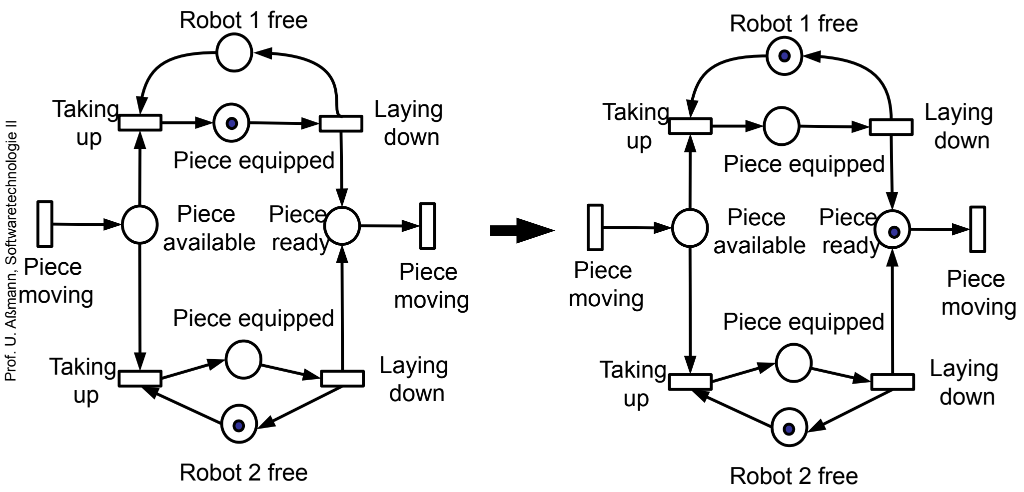
A PN with the given initial marking is denoted by (N, M_0)

Example of 2 Robots as Predicate/Transition Net



Example of 2 Robots as Predicate/Transition Net

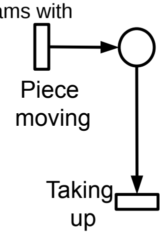
- Places represent predicates; tokens show validity

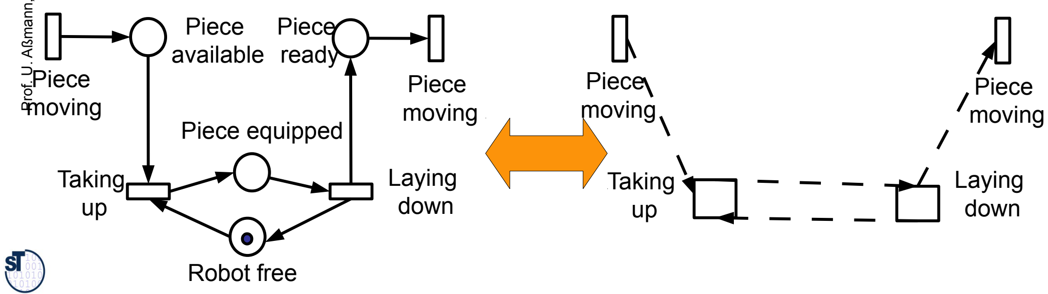


3.1.2 Special Nets

Marked Graphs (MG)

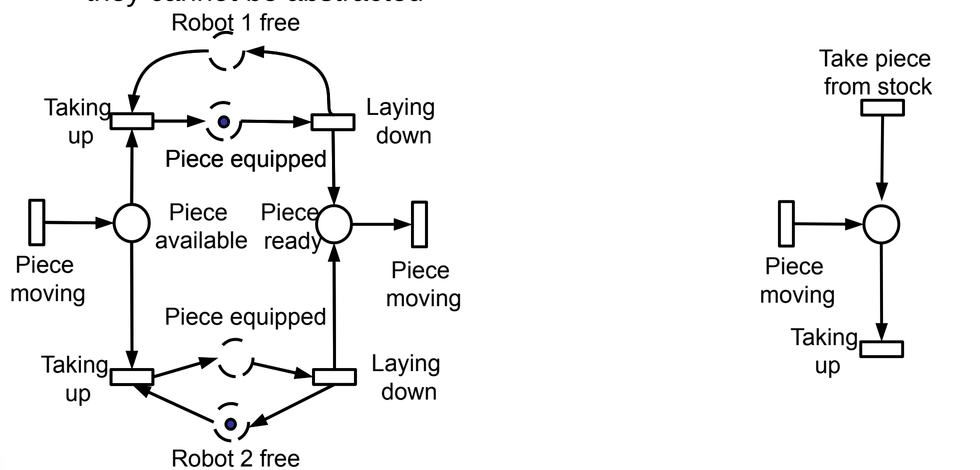
- A *Marked Graph (MG)* is an PN such each place is the input to only one transition and the output of only one transition. MG provide d*eterministic parallelism without confusion*
  - Then the places can be abstracted (identified with one flow edge)
  - Transitions may split and join, however
- Marked Graphs correspond to a special class of data-flow graphs (Data flow diagrams with restricted stores, DFD)
  - Transitions correspond to processes in DFD, places to stores
  - States can be *merged* with the ingoing and outcoming arcs → DFD without stores
  - Restriction: Stores have only one producer and consumer
  - But activities can join and split
- All theory for CPN holds for marked graph - DFD, too [BrozaWeide]
- Bsp. Robot is a DFD (but not the assembly line):





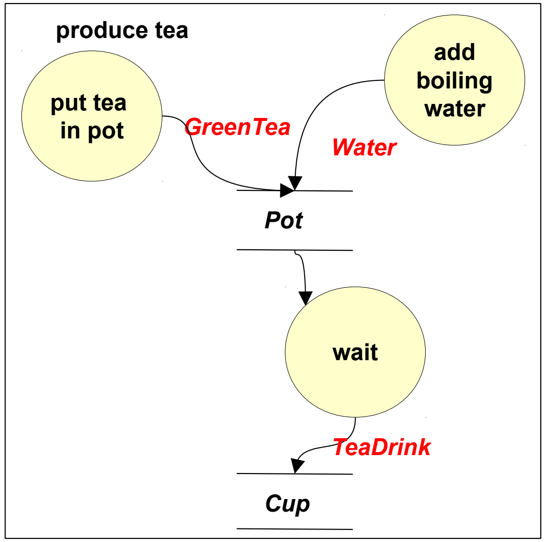
More General Data-Flow Diagrams

- General DFD without restriction can be modeled by PN, too. Then, places cannot be abstracted; they correspond to stores with 2 feeding or consuming processes
- Example: the full robot has places with 2 ingoing or outgoing edges, they cannot be abstracted



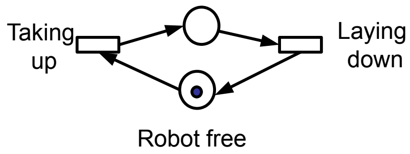
For DFD, Many Notations Exist

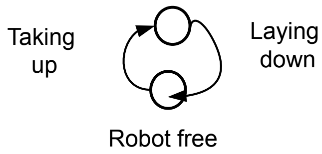
- Notation from Structured Analysis [Balzert]



State Machines are PN with Cardinality Restrictions

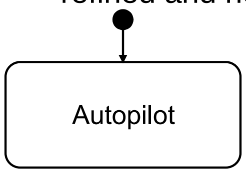
- A *Finite State Machine* PN is an elementary PN such that each transition has only one input and one output place
  - Then, it is equivalent to a finite automaton or a *statechart*
  - From every class-statechart that specifies the behavior of a class, a State Machine can be produced easily
    - Flattening the nested states
  - Transitions correspond to transitions in statecharts, states to states
  - Transitions can be *merged* with the ingoing and outcoming arcs
  - In a FSM there is only one token
- All theory for CPN holds for Statecharts, too
- Ex. Robot is an FSM (but not with incoming data flow):

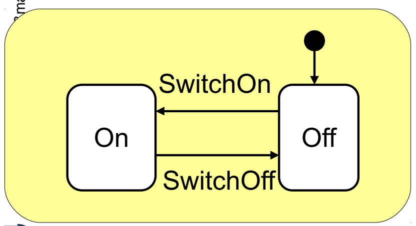


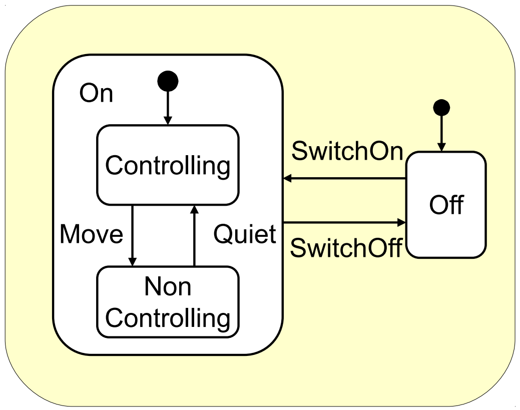


Hierarchical StateCharts from UML

- States can be nested in StateCharts
- This corresponds to StateMachine-PN, in which states can be refined and nested







3.1.2 Colored Petri Nets as Example of High Level Nets

Modularity, Refinement, Reuse

Preparing "reducible graphs"

Colored Petri Nets, CPN

- Colored (Typed) Petri Nets (CPN) refine Petri nets:
  - Tokens are typed (colored)
  - Types are described by data structure language, such as Java, ML, UML class diagrams
  - but may also be data dictionaries, grammars
  - Concept of time can be added
- Full tool support
  - Fully automated code generation in Java and ML (in contrast to UML), e.g., DesignCPN of Aarhus University http://www.daimi.aau.dk
  - Prover proofs features about the PN
  - Net simulator allows for debugging
- Much better for safety-critical systems than UML, because proofs can be done

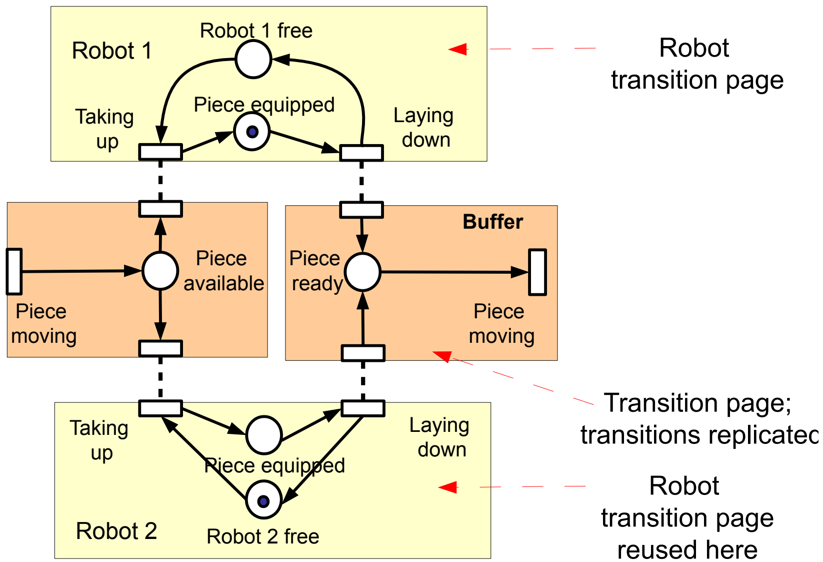
Annotations in CPN

- Places are annotated by
  - Token types
    - `(STRING x STRING)`
  - Markings of objects and the cardinality in which they occur:
    - `2'("Uwe","Assmann")`
- Edges are annotated by
  - Type variables which are unified by unification against the token objects
    - `(X,Y)`
  - Guards
    - `[ X == 10]`
  - if-then-else statements
    - `if X < 20 then Y := 4 else Y := 7`
  - switch statements
  - boolean functions that test conditions

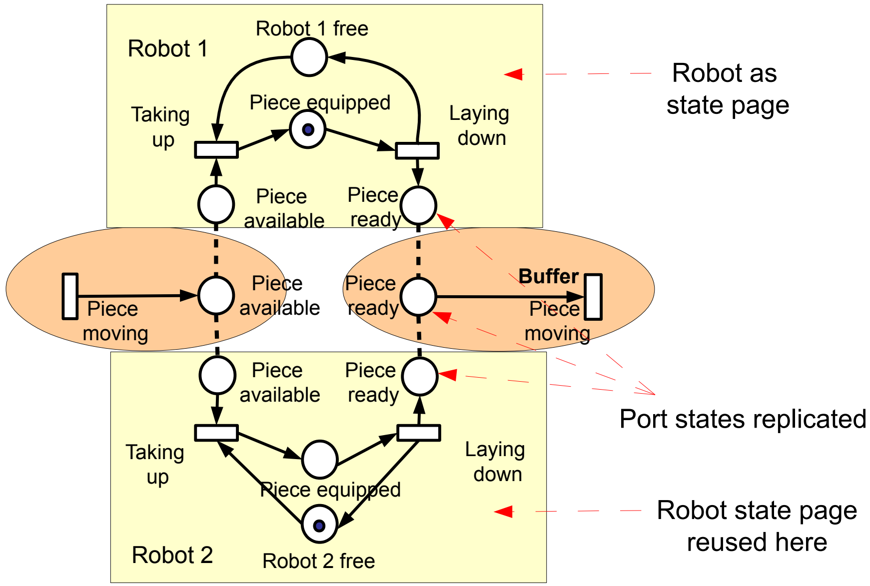
CPN are Modular

- A subnet is called a *page (module)*
  - Every page has ports which mark in- and out-going transitions (into a place) or in- and outgoing places (into a transition)
- *Transition page*: interface contains transitions (transition ports)
- *Place page (state page)*: interface contains place (place ports)
- *Net class*: a named page that is a kind of "template" or "class"
  - It can be instantiated to a net "object"
- Reuse of pages and templates possible
  - Libraries of CPN "procedures" possible

Robots with Transition Pages, Coupled by Transition Ports



Robots with Place (State) Pages, Coupled by Replicated State Ports

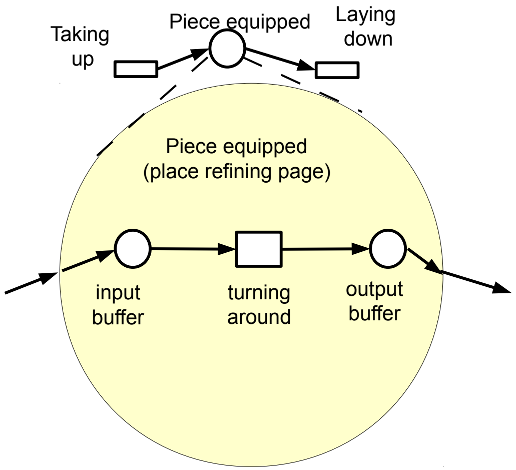


CPN are Hierarchical

- Places and transitions may be hierarchically **refined**
  - Two pointwise refinement operations:
    - Replace a transition with a transition page
    - Replace a state with a state page
  - Refinment condition: Retain the embedding (embedding edges)
- CPN can be arranged as hierarchical graphs (reducible graphs, see later)
  - Large specifications possible, overview is still good
  - Subnet stemming from refinements are also place or transition pages

Point-wise Refinement Example

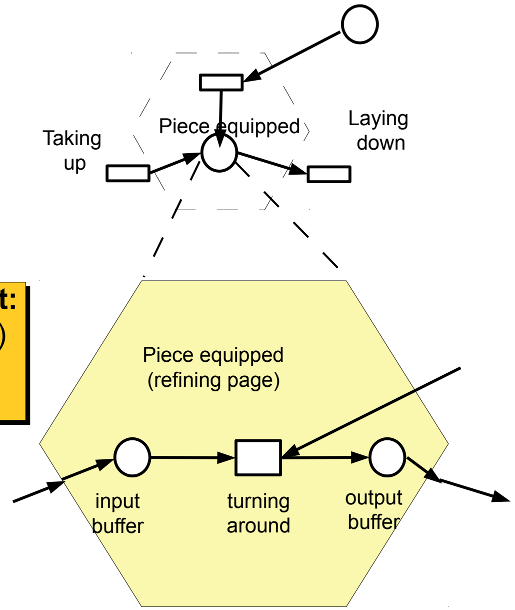
- Pointwise refinement:
  - *Transition refining page*: refines a transition, transition ports
  - *Place refining page (state refining page)*: refines a place, place ports



Law of syntactic refinement: The graph interface (attached edges) of a refined node must be retained by the refining page.

Region (Hyperedge) Refinement Example

- Hyperedges and regions in PN can be refined



Law of syntactic region refinement: The graph interface (attached edges) of a refined region must be retained by the refining region.

Industrial Applications of CPN

- Large systems are constructed as reducible specifications
- ..have 10-100 pages, up to 1000 transitions, 100 token types
- Example: ISDN Protocol specification
  - Some page templates have more than 100 uses
  - Corresponds to millions of places and transitions in the expanded, non-hierarchical net
  - Can be done in several person weeks

3.2 Patterns in Petri Nets

Analyzability:

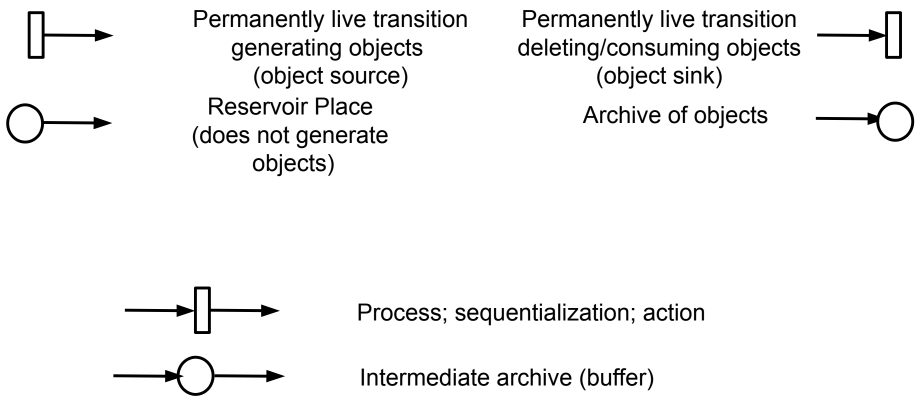
Petri Nets can be analyzed for patterns (by pattern matching)

Modelling of Parallelism and Synchronization

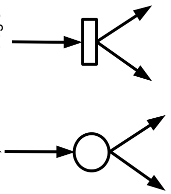
Petri Nets have a real advantage when parallel processes and synchronization must be modelled

Many concepts can be expressed as *PN patterns*

Simple PN Buffering Patterns



Parallelism Patterns

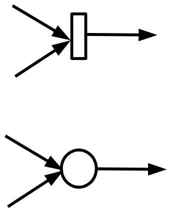


Replication and distribution of objects; forking off parallelism (AND-split)

Forking off parallelism indeterministically (conflict, XOR split)

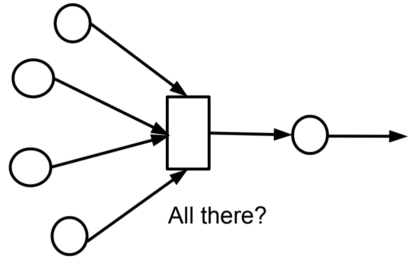
Joining parallelism synchronization barrier (AND-join)

Collecting objects from parallel processes (OR-join)

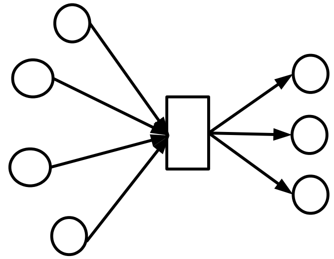


Examples for Building Blocks

Synchronization barrier

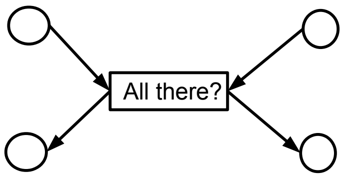


Bridges: Transitions between phases

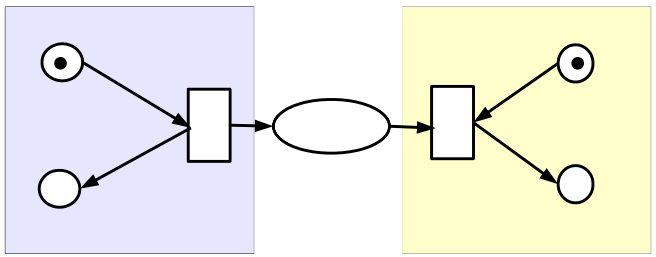


Patterns for Parallelism

Coupling processes with parallel continuation

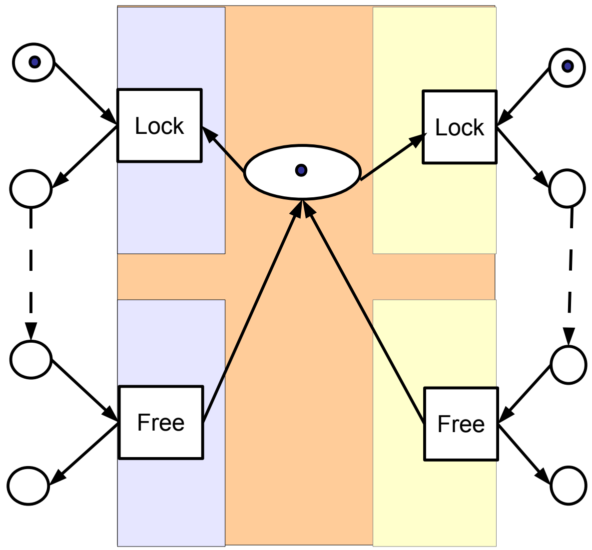


Producer/Consumer with buffer (CSP channel)

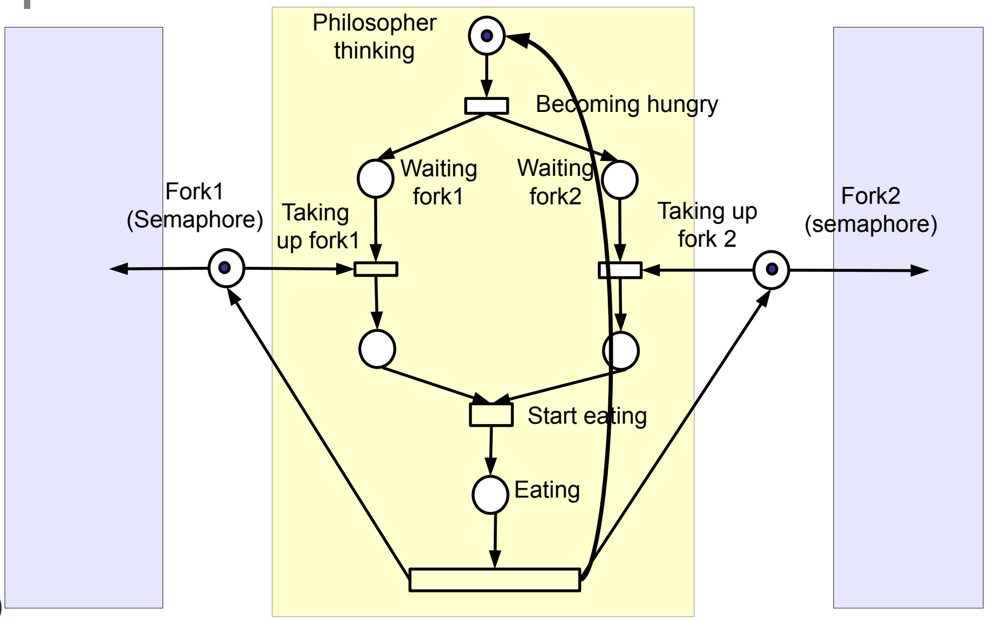


Semaphores For Mutual Exclusion

Binary or counting semaphores: depends on the capacity of the semaphore place



Dining Philosophers



Advantage

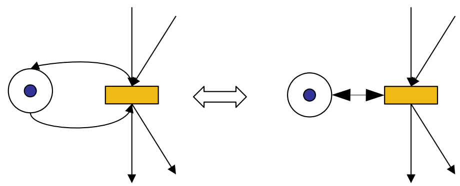
- Patterns can be used to model specific requirements
- PN can be checked for patterns by Pattern Matching (Graph Rewriting)
  - Patterns can be restructured (refactorings)
  - Patterns can be composed (composition)
- Further semantic analysis of PN: Parallel, indeterministic systems can be checked for
  - Absence of deadlocks: will the parallel system run without getting stuck?
  - Liveness: will all parts of the system work forever?
  - Fairness: will all parts of the system be loaded equally?
  - Bounded resources: will the system use limited memory, and how much? (important for embedded systems)
  - Whether predicates hold in certain states (model checking)

3.3 Refactorings (Reduction Rules) for Petri Nets

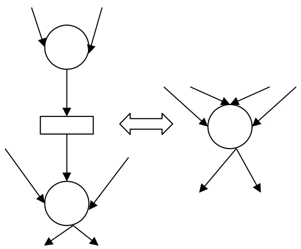
.. in the form of graph rewrite rules

Special Restructuring Patterns (Refactorings)

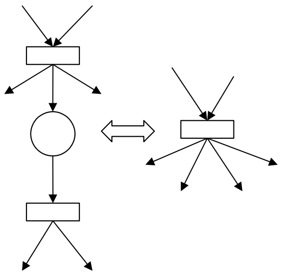
- *Source transitions* are always enabled, i.e., generate tokens (*token generator)*
- *Sink transitions* are always enabled and swallow tokens (*token sink)*
- A *self-loop* is a pair of a place *p* and a transition *t* if *p* is both output and input place of *t*
  - A PN without any self-loops is *pure.* Its arc relation is irreflexive



Simple Reduction Rules

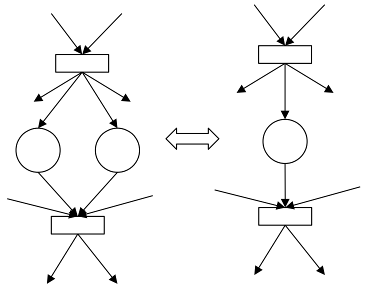


1) Fusion of Series Places (FSP)
(Bridge elimination)

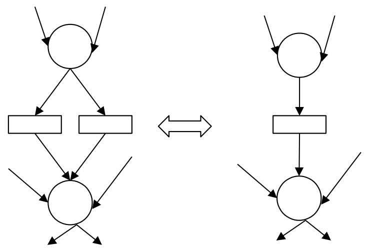


2) Fusion of Series Transitions (FST)
(Intermediate buffer elimination)

Simple Reduction Rules

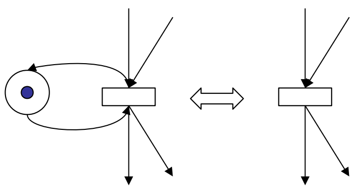


3) Fusion of Parallel Places (FPP)

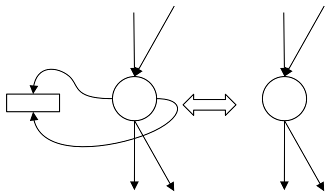


4) Fusion of Parallel Transitions (FPT)

Simple Reduction Rules



5) Elimination of Self-loop Places (ESP)



6) Elimination of Self-loop Transitions (EST)

All transformations preserve liveness, safeness and boundedness.

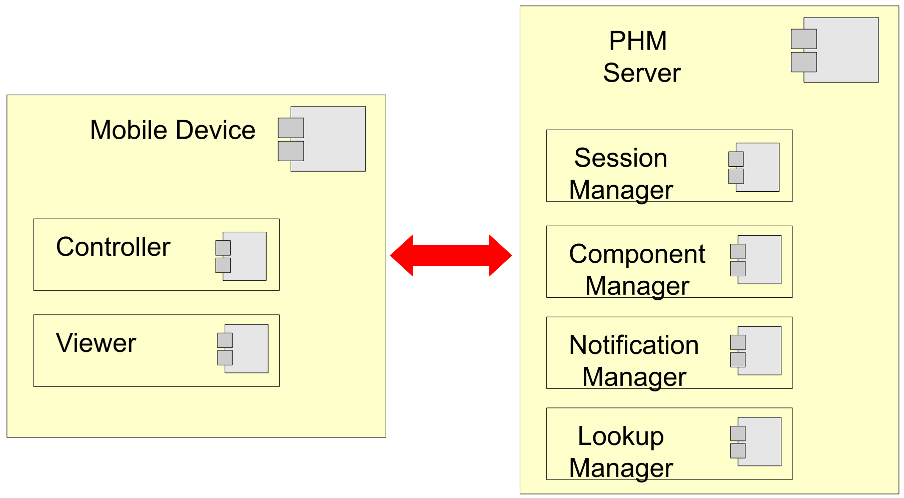
3.4 Composability of CPN

Case Study for Composition: Pervasive Healthcare Middleware (PHM)

- in development at the Pervasive Computing Center, University of Aarhus
- Basic idea:
  - Specify the structure of an application with UML
  - and the behavior with CPN, describing the behavior of the classes/objects (object lifecycle)
  - Glue behavior together with page glueing mechanism
- Electronic patient records (EPR) replace the papers
  - First version in 2004, on stationary PC
  - Next versions for pervasive computing (PDA, wireless):
    - Hospital employees will have access to the patient's data whereever they go, from Xray to station to laboratories
  - For instance, medication plans are available immediately

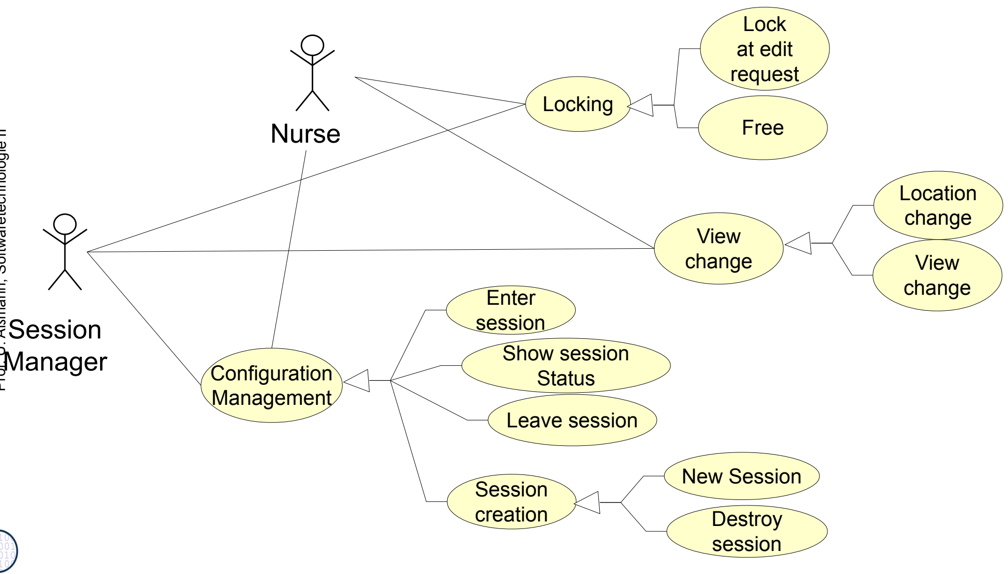
The PHM Architecture

- A *session* is entered by several mobile devices that collaborate

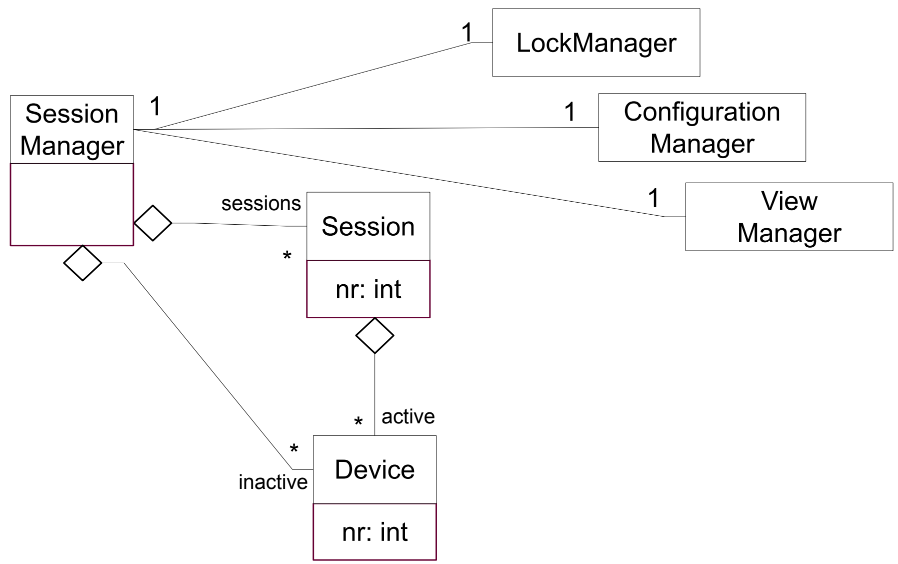


Session Manager Use Cases

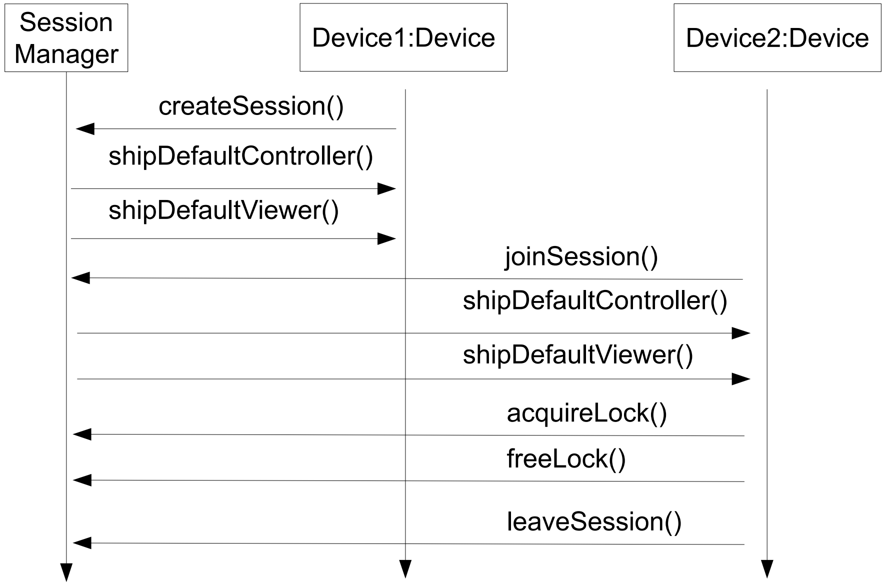
- The *session manager* manages all mobile devices that collaborate in a certain scenario



Class Diagram Session Manager

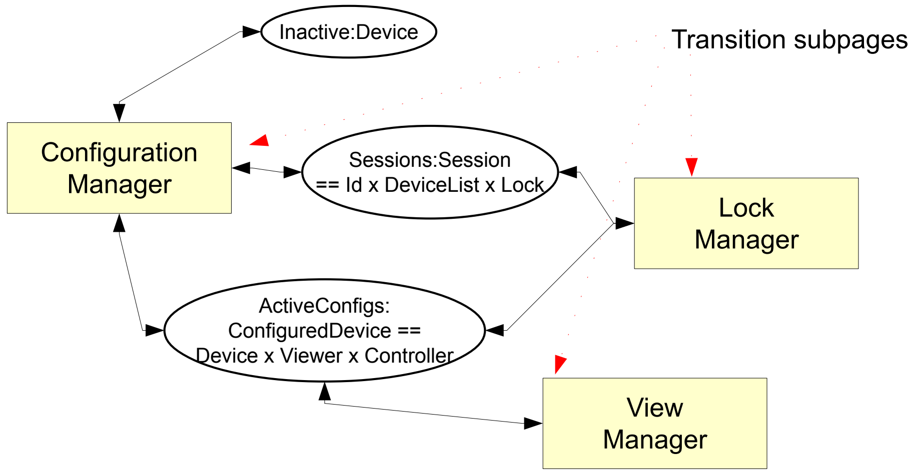


Sequence Diagram Session Manager



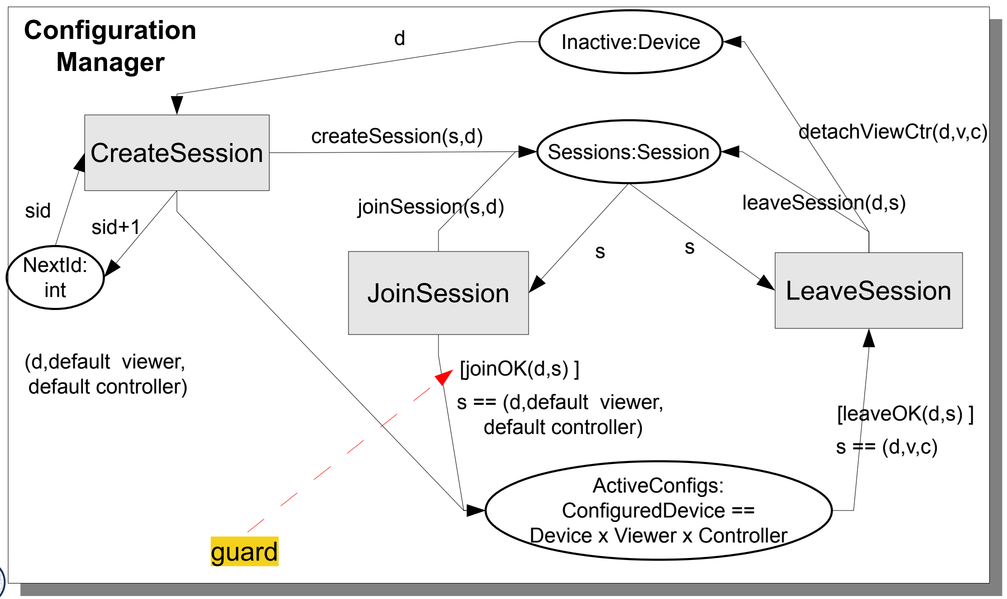
Session Manager Top-Level CPN

- Double arrows indicate that arrows run in both directions
- Basic Types
  - `Session ::= SessionId DeviceList LockType`
  - `ConfiguredDevice ::= Device Viewer Controller`

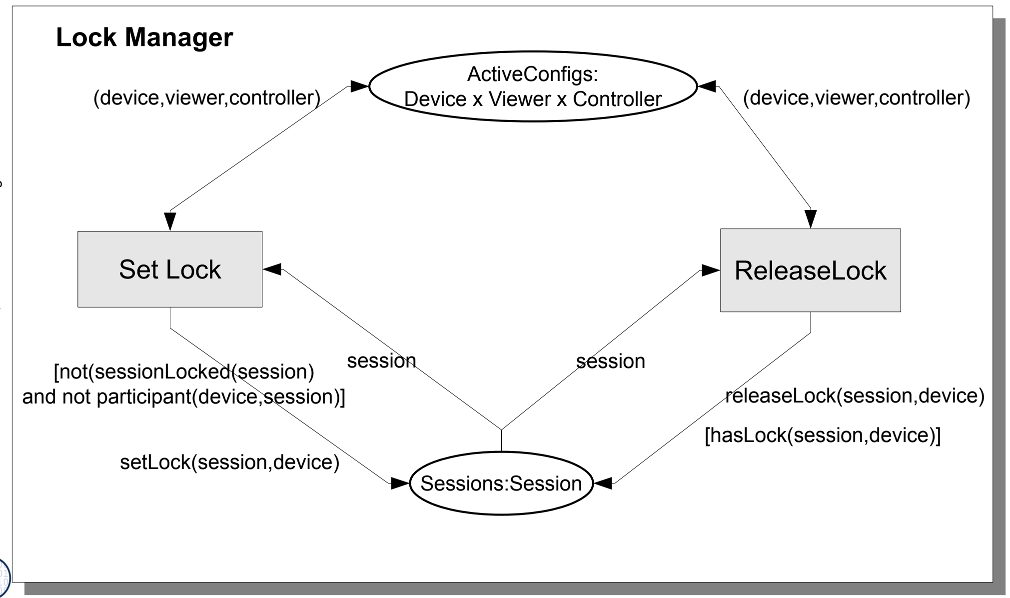


Configuration Manager Page

- Page is fused along common names of nodes

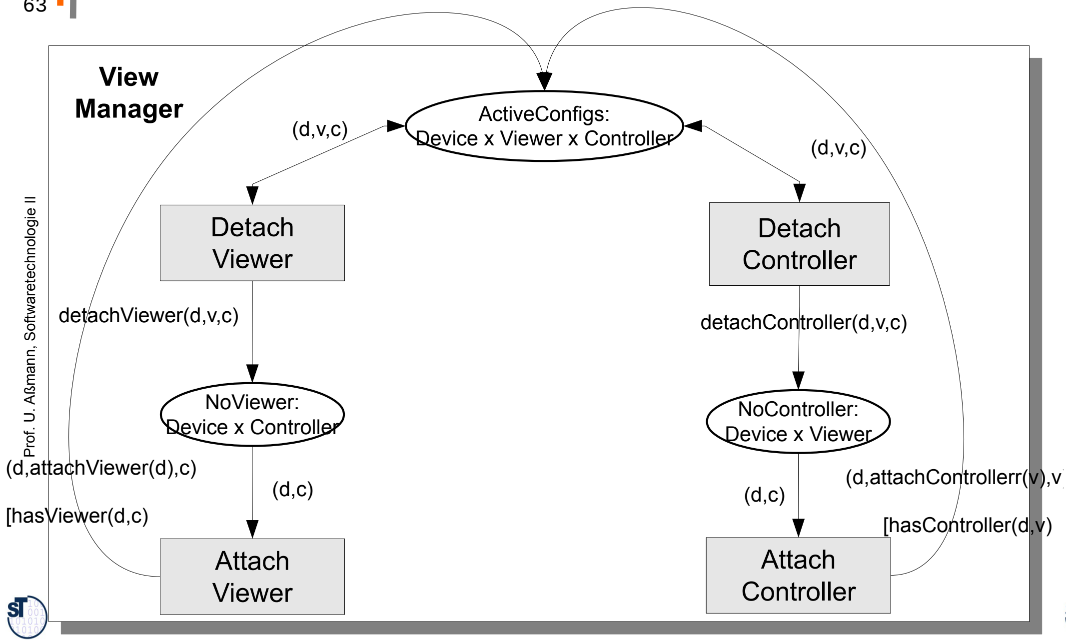


Lock Manager Page



View Manager Page

63



Remarks

64

- The CPN pages are attached to UML classes, i.e., describe their behavior
  - States and transitions are marked by UML types
- Every subpage is coupled to others (composed with others)
  - via common states (*fusing/join states*)
    - The union of the pages via join states is steered by OR, i.e., the pages add behavior, but do not destroy behavior of other pages
  - Via common transitions (*fusing/join transitions)*
    - The union of the pages via join transitions is steered by AND, i.e., the pages add behavior and synchronize with transitions of other pages
- Transitions are interpreted as coarse-grain events
  - On the edges, other functions (actions) are called
  - Hence, CPN are *open:* if something is too complicated to model as a PN, put it into functions

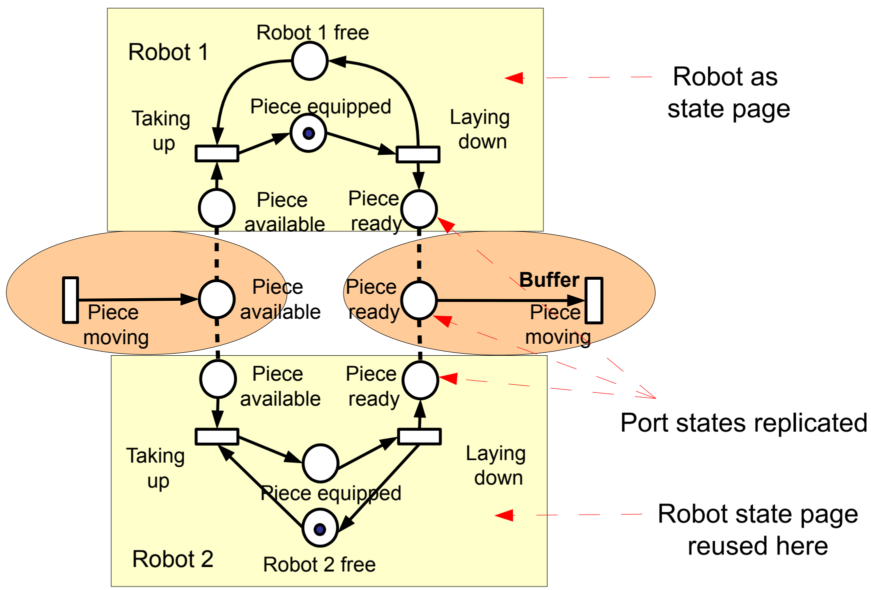
Coupling of Place and Transition Pages

65

- **Port state coupling (**or **fuse, merge, composition):** Place pages are coupled to other place pages via common states (port states)
  - The union of the pages is steered by OR, i.e., the pages add behavior, but do not destroy behavior of other pages
- **Port transition coupling:** Transition pages are coupled to other transition pages via common transitions (port transitions)
  - The union of the pages is steered by AND, and every page changes the behavior of other page
  - Events must be available on every incoming edge of a transition
  - The transitions of the combined net only fire if the transitions of the page components fire

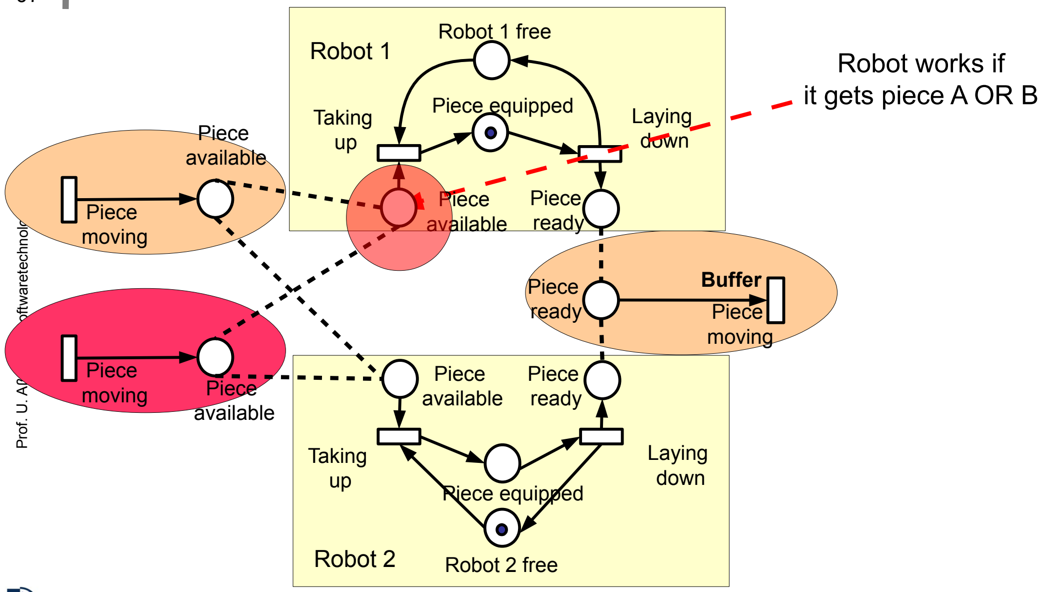
Robots with State Pages, Coupled by Replicated State Ports

66

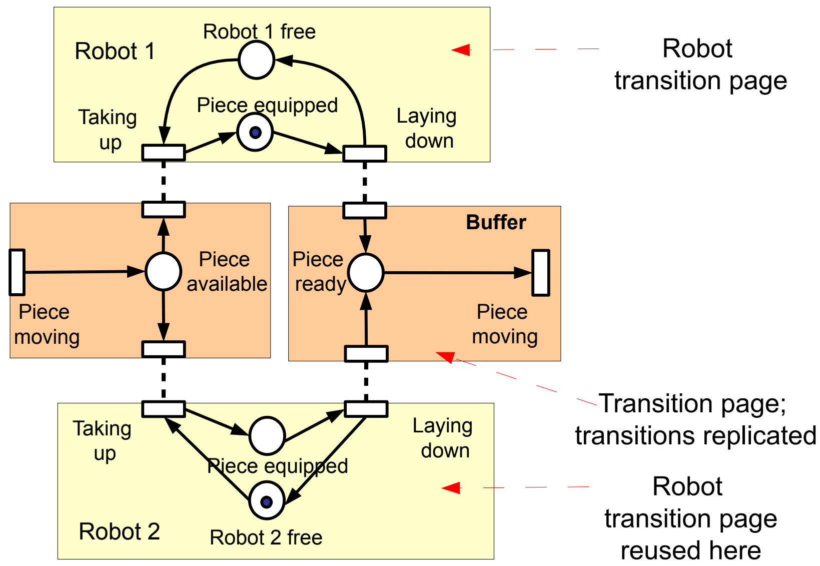


[Helmut Balzert]

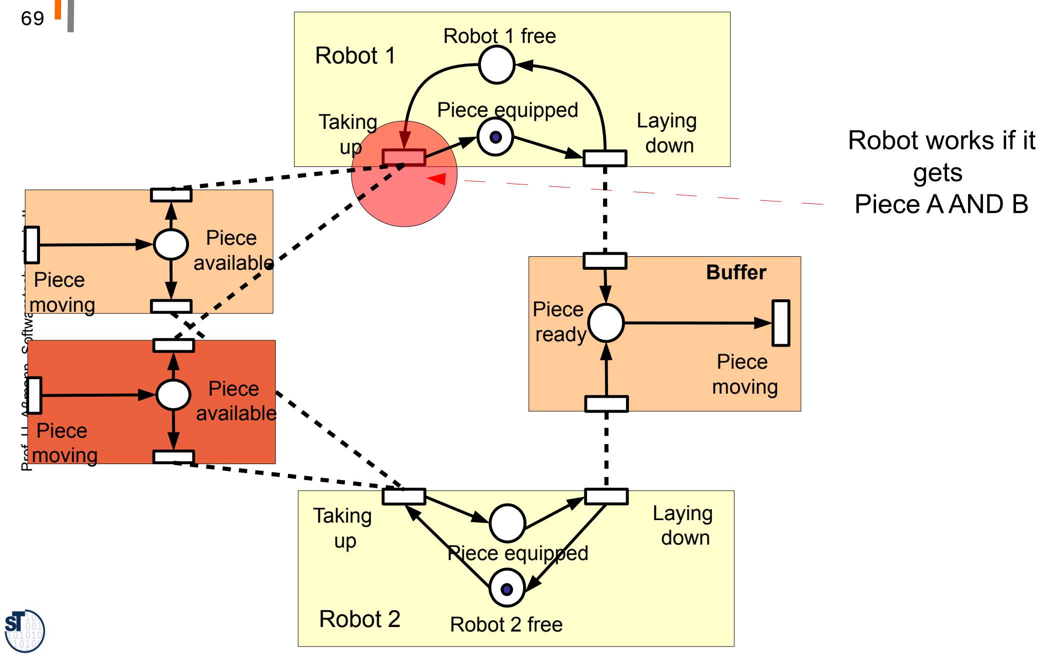
A Robot OR-composed View



Robots with Transition Pages, Coupled by Transition Ports



A Robot AND-composed view



Advantages of CPN for the PHM

- The PHM is a distributed and mobile scenario
  - Devices can fail (battery empty, wireless broken, etc)
  - The resulting CPN can be checked on deadlock, i.e., will the PHM session manager get stuck?
- Compact specification
  - Usually, CPN are much more compact than statecharts
- Variability
  - The pages are modular, i.e., can be exchanged for variants easily (e.g., other locking scheme)

3.4 Parallel Composition of Colored Petri Nets

71

Parallel composition of PN

72

- Complex synchronization protocols can be abstracted to a pattern (als called transition page or a place page)
- When joining PN with AND (i.e., joining transition pages), synchronization protocols can be overlayed to existing sequential specifications

Unforeseeable Extensible Workflows

73

- Workflows are described by Colored Petri Nets (CPN) or languages built on top of CPN:
  - YAWL language [van der Aalst]
  - Workflow nets
- We can use the extension of CPN for workflow composition, enriching a workflow *core* with a workflow *aspect:*
  - **Place extension (State extension):** adding more edges in and out of a place (state):
    - OR-based composition: Core OR view: Core-place is ORed with Aspect-Place
  - **Transition extension (Activity extension):** adding more edges in and out of a transition (activity)
    - AND-based composition: Core-transition is ANDed with Aspect-transition

Weaving Patterns for Synchronization Protocols with AND Composition

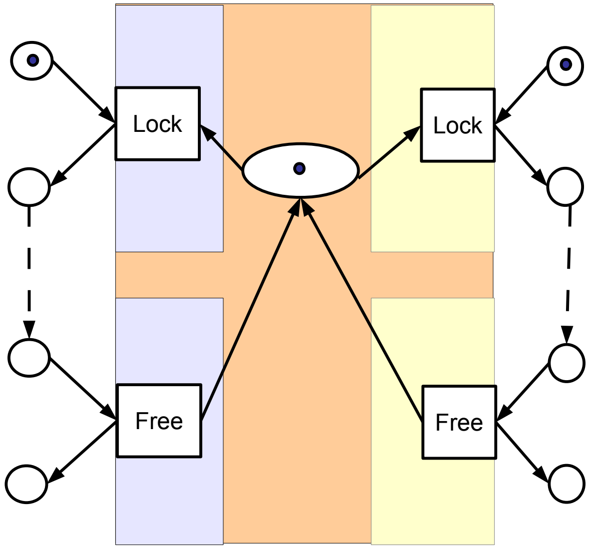
74

- Complex synchronization protocols can be abstracted to a transition page
- Weaving them with AND, they can be overlayed to existing sequential specifications

Semaphores For Mutual Exclusion Revisited

75

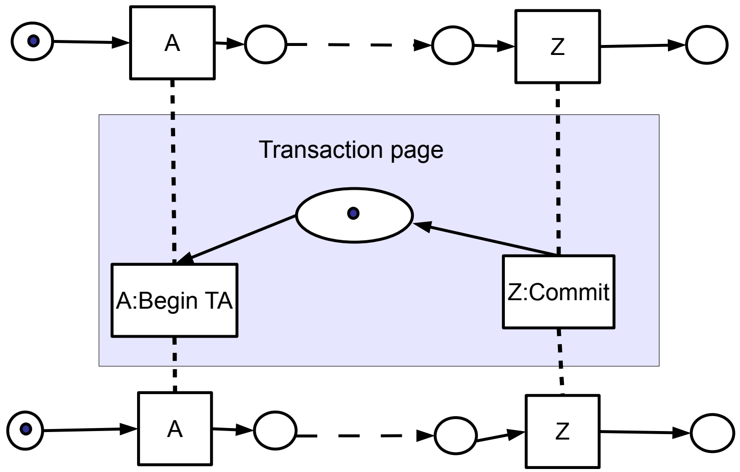
- Forms a synchronisation aspect via ANDed Lock transitions



Transaction Protocols as AND-Aspects

76

- Crosscut between processes (cores) and transaction protocol (aspect)



Insight

77

- AND-Merge and OR-Merge of CPN are sufficient basic composition operators for building complex aspect weavers for workflow languages built on CPN

AND-weaving for synchronization

OR-weaving for functional extension

3.5 The Application to Modelling

78

Petri Nets Generalize UML Behavioral Diagrams

Activity Diagrams

- Activity Diagrams are similar to PN, but not formally grounded
  - Without markings
  - No liveness analysis
  - No resource consumption analysis with boundness
  - No correspondence to UML statechart, although for PN holds that PN with finite reachability graphs correspond to finite automata
- I.e., it is difficult to prove something about activity diagrams, and difficult to generate (parallel) code from them.

Data-flow diagrams

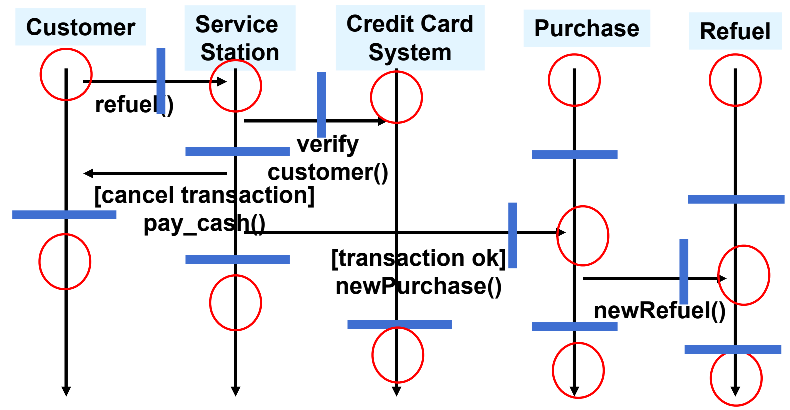
- DFD are special form of activity diagrams, and correspond to Marked Graphs

Statecharts

- Finite automata are restricted form of Petri nets
- The hierarchical structuring in Statecharts is available in High-Level Petri Nets (e.g., CPN)

Petri Nets Generalize UML Sequence Diagrams

- The object life lines of a sequence diagram can be grouped into state such that a PN results
- All of a sudden, liveness conditions can be studied
  - Is there a deadlock in the sequence diagram?
  - Are objects treated fair?



A Simple Modelling Process for Safety-Critical Software with CPN

- **Elaboration**: Identify active and passive parts of the system
  - Active become transitions, passive to places
- **Elaboration**: Find the relations between places and transitions
- **Elaboration**: How should the tokens look like: boolean? Integers? Structured data?
  - Use UML class diagrams as token type model
- **Restructure**: Group out subnets to separate "pages"
- **Refactor**: Simplify by reduction rules
- **Verify**: Analyse the specification on liveness, boundedness, reachability graphs, fairness. Use a model checker to verify the CPN
- **TransformRepresentation**: Produce views as statecharts, sequence, collaboration, and activity diagrams..

How to Solve the Reactor Software Problem?

- Specify with UML and CPN
  - Verify it with a model checker
  - Let a prototype be generated
  - Test it
  - Freeze the assembler
- Then, verify the assembler, because you should not trust the CPN tool nor the compiler
  - Any certification agency in the world will require a proof of the assembler!
- However, this is much simpler than programming reactors by hand...

The Gloomy Future of PN

- PN will become the major tool in a future CASE tool or integrated development environment
  - Different views on the PN: state chart view, sequence view, activity view, collaboration view!
- Many isolated tools for PN exist, and the world waits for a full integration into UML
- CPN will be applied in scenarios where parallelism is required
  - Architectural languages
  - Web service langauges (BPEL, BPMN, ...)
  - Workflow languages
  - Coordination languages

The End

- Thanks to Björn Svensson for help in making slides, summarizing [Murata]