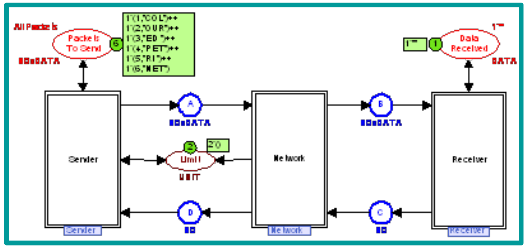
State Space Exploration of Coloured Petri Nets and the ASAP Model Checking Platform

Petri Nets 2010 Tutorial

Lars M. Kristensen
Department of Computer Engineering
Bergen University College, Norway
lmkr@hib.no

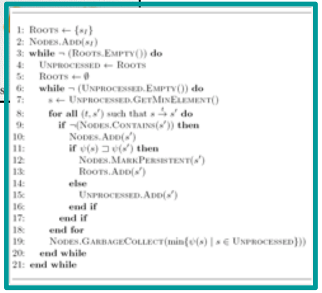
Michael Westergaard
Computer Science Department,
University of Aarhus, Denmark
mw@cs.au.dk



ASAP

ASCoVeCo State space
Analysis Platform

Copyright © 2007, 2008 CPN Group, University of Aarhus



```
Roots ← {$s_I$}
Nodes.Add($s_I$)
while ¬ (Roots.Empty()) do
  Unprocessed ← Roots
  Roots ← $\emptyset$
  while ¬ (Unprocessed.Empty()) do
    $s$ ← Unprocessed.GetMinElement()
    for all $(t, s')$ such that $s \xrightarrow{t} s'$ do
      if ¬(Nodes.Contains($s'$)) then
        Nodes.Add($s'$)
        if $\psi(s) \sqsupset \psi(s')$ then
          Nodes.MarkPersistent($s'$)
          Roots.Add($s'$)
        else
          Unprocessed.Add($s'$)
        end if
      end if
    end for
    Nodes.GarbageCollect(min{$\psi(s) \mid s \in$ Unprocessed})
  end while
end while
```

ISBN: 978-972-8692-51-3

Tutorial Outline (1)

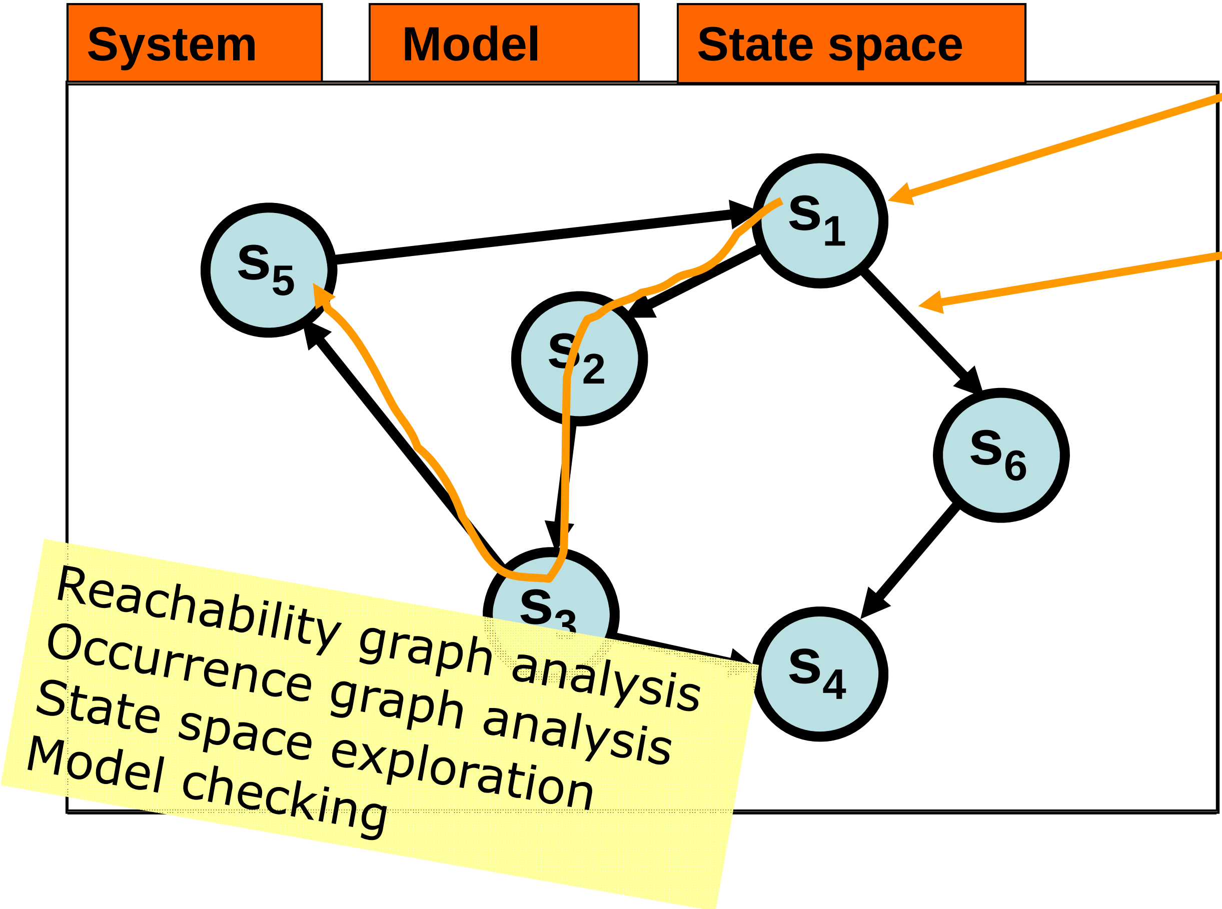
- **Part 1: Introduction**
  - State space methods for Coloured Petri Nets (CPNs) and the research area.
  - Example of a practical application: verification of an edge router discovery protocol with Ericsson.
  - Overview of the ASAP model checking platform.
- **Part 2: User perspective**
  - Managing verification projects.
  - Creating and executing verification jobs: The JoSEL language.
  - Safety properties and LTL model checking.

Tutorial Outline (2)

- **Part 3: Advanced state space methods**
  - Compact in-memory storage: the comback method
  - State deletion: the sweep-line method.
  - State space partitioning for external memory and distributed model checking.
- **Part 4: Research perspective**
  - Extending ASAP with new state space methods.
  - Benchmarking and profiling.
  - Status and outlook.

State Space Exploration

- **One of the main approaches to model-based verification of finite-state concurrent systems:**



Reachability graph analysis
Occurrence graph analysis
State space exploration
Model checking

Nodes → set of reachable states

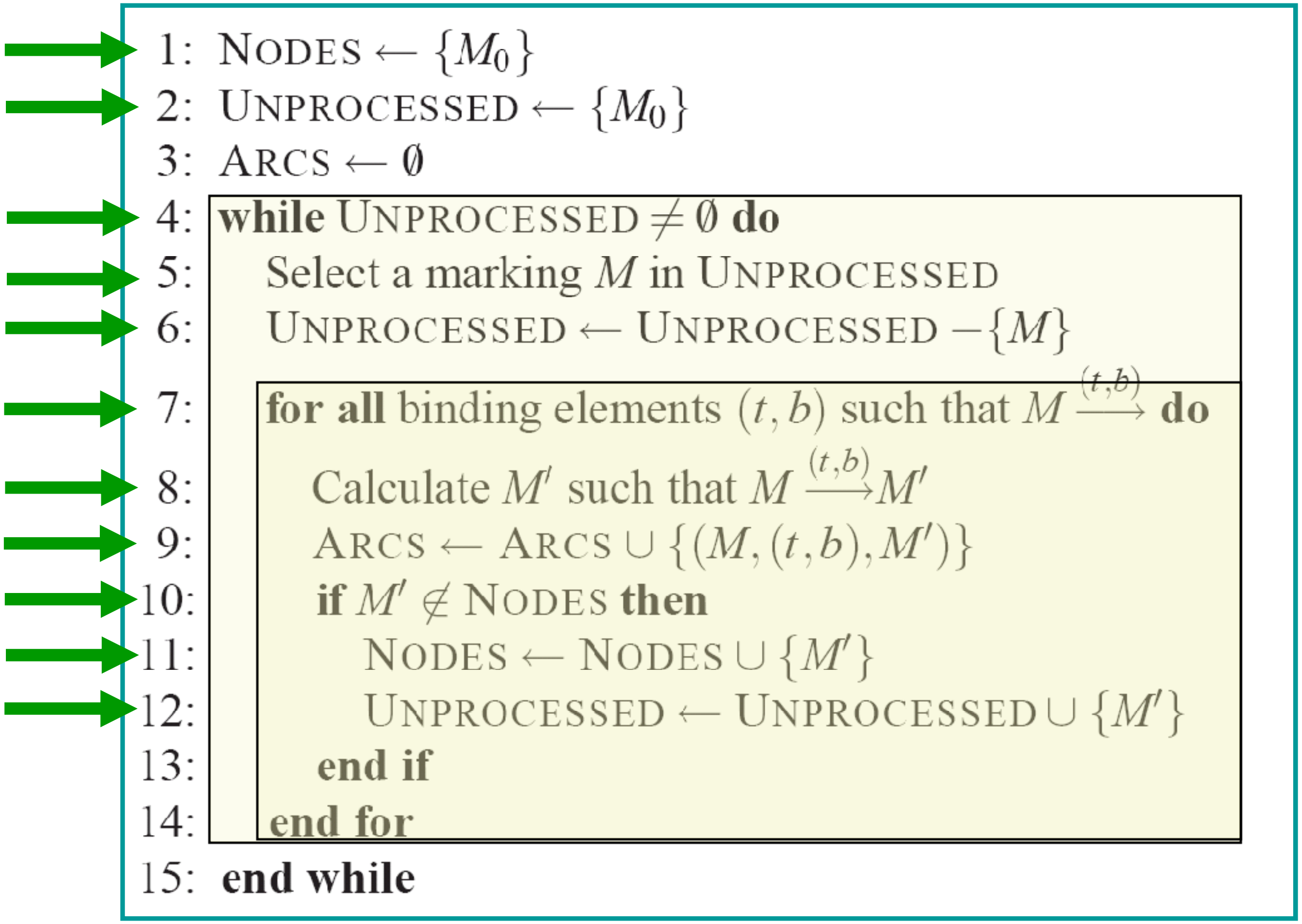
Arcs → occurrences of events

Paths → Set of execution sequences

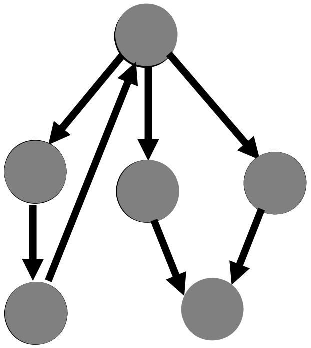
Guarantees complete coverage of executions:

Error detection + Verification

Explicit State Space Exploration



```
$\textsc{Nodes} \leftarrow \{M_0\}$
$\textsc{Unprocessed} \leftarrow \{M_0\}$
$\textsc{Arcs} \leftarrow \emptyset$
while $\textsc{Unprocessed} \neq \emptyset$ do
    Select a marking $M$ in $\textsc{Unprocessed}$
    $\textsc{Unprocessed} \leftarrow \textsc{Unprocessed} - \{M\}$
    for all binding elements $(t,b)$ such that $M \xrightarrow{(t,b)}$ do
        Calculate $M'$ such that $M \xrightarrow{(t,b)} M'$
        $\textsc{Arcs} \leftarrow \textsc{Arcs} \cup \{(M,(t,b),M')\}$
        if $M' \notin \textsc{Nodes}$ then
            $\textsc{Nodes} \leftarrow \textsc{Nodes} \cup \{M'\}$
            $\textsc{Unprocessed} \leftarrow \textsc{Unprocessed} \cup \{M'\}$
        end if
    end for
end while
```



State Space Exploration Methods

- **Advantages:**
  - Highly automatic support by computer tools (construction and analysis algorithms).
  - Much of the underlying mathematics can be hidden.
  - Rich set of behavioural properties can be analysed.
  - Counter examples and diagnostics information.
  - Even partial state spaces provide a systematic and effective error-detection technique.

- **Disadvantages:**
  - Verification relative to specific system configuration.
  - Inherent state explosion problem.

State Space Methods Zoo

Directed model checking

Sweep-line method

Bit-state hashing

Symmetry method

State caching

Büchi automata

Stubborn sets

ample sets

Distributed model checking

Equivalence method

Modular state spaces

Comback method

LTL model checking

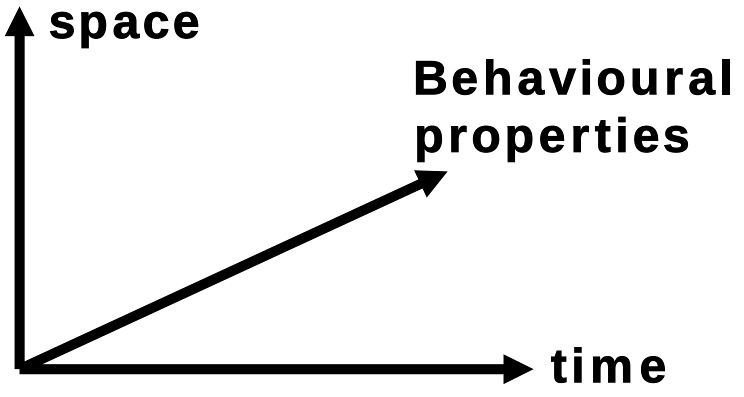
Language equivalence

Hash compaction

Symbolic model checking

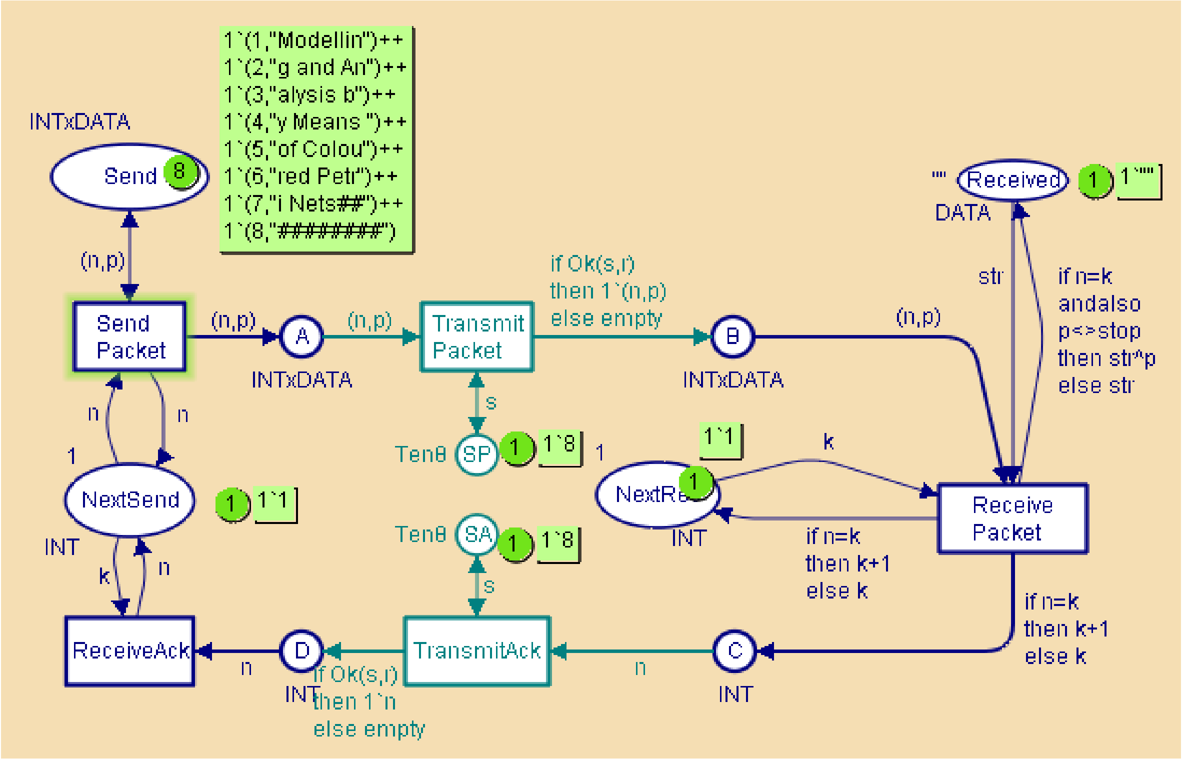
CTL model checking

- **Mostly modelling language independent.**
- **Typically exploits specific system characteristics.**



Coloured Petri Nets (CPNs)

- **Combination of Petri Nets and Standard ML.**



Petri Nets:
- concurrency
- control structures
- synchronisation
- communication
- resource sharing

Standard ML:
- data manipulation
- compact modelling

- **Construction, simulation, and basic state space exploration is supported by CPN Tools.**

CPNs and State Space Methods

- **A main guidelines has been to support the full CPN modelling language:**
  - The rich data types yields state vectors of typically 100-1000 bytes.
  - The complex inscriptions make it infeasible to exploit structural properties.
  - Unfolding to low-level Petri Nets is not an viable option.
  - Calculation of enabling binding element (events) is expensive.
- **Advantages of the CPN modelling language:**
  - The possibility of compact modelling yields smaller state spaces (model level reduction).
  - The hierarchical structure facilitates sharing of sub-states.
  - Petri net locality can be exploited to reduce time spent on calculating enabled binding elements (events).

Practical Applications

- **State space methods for CPNs have been widely used for verification purposes:**
  - Danfoss Flowmeter systems.
  - Bang & Olufsen Beolink System.
  - Scheduling at Australien Defence Force.
  - Ericsson Edge Router Discovery Protocol.
  - Several Internet protocols (e.g., WAP, IOTP, TCP, DCCP, SIP, DYMO).
  - ...
- **For further examples:**

  http://www.cs.au.dk/CPnets/intro/industrial.shtml

An Example Application

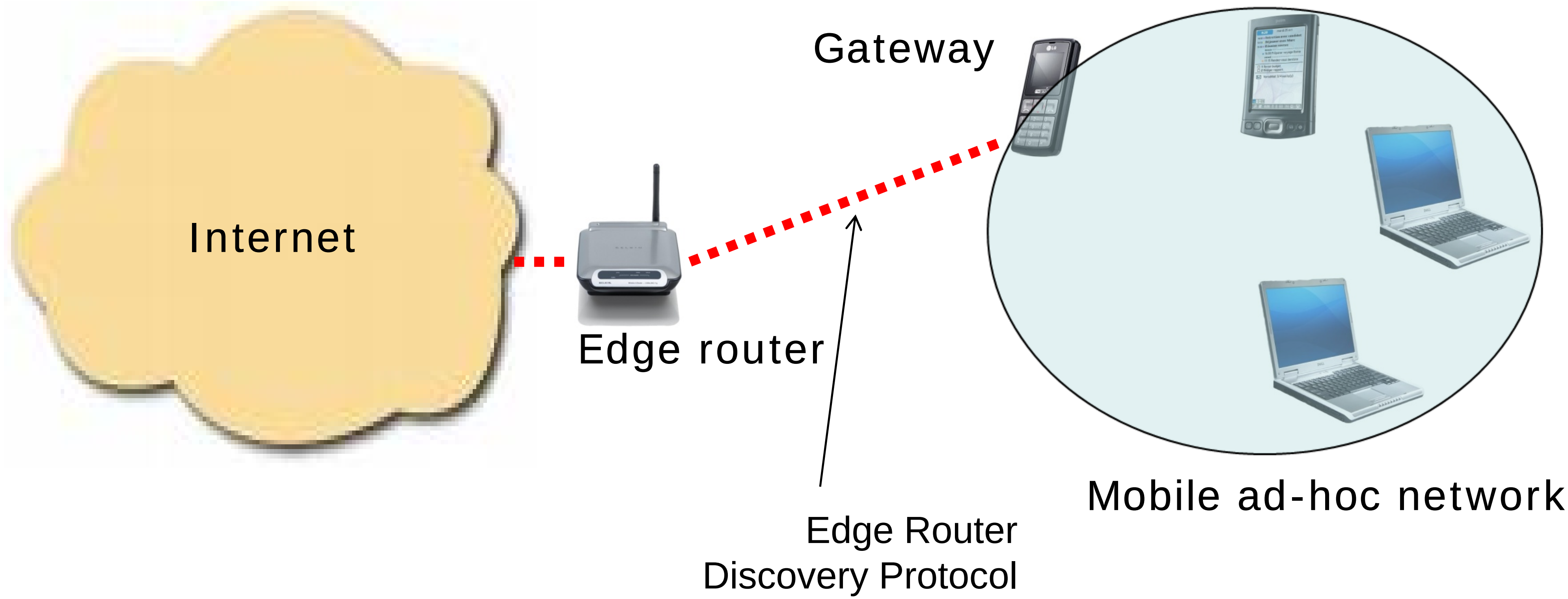
- Design of an Edge Router Discovery Protocol (ERDP) for mobile ad-hoc networks.

ERICSSON

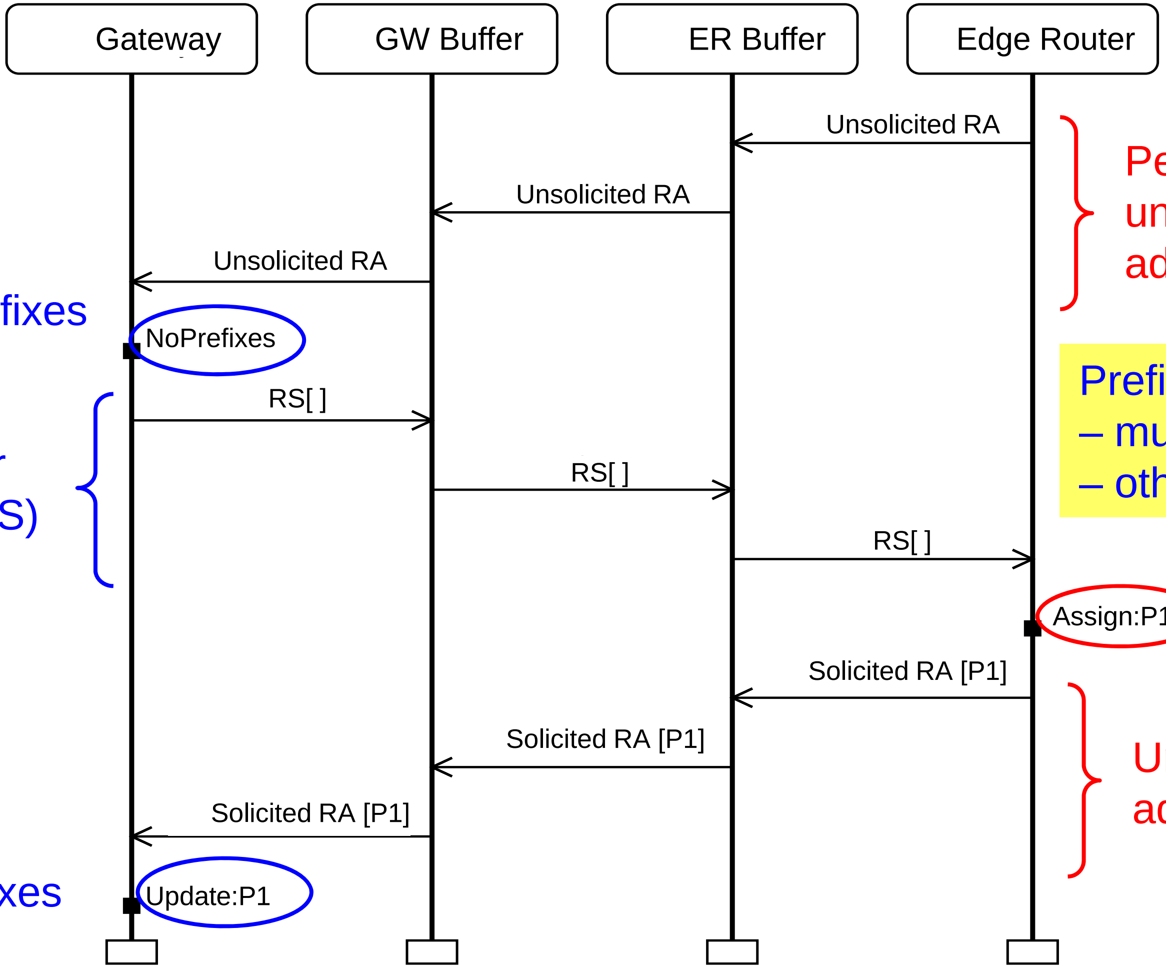
- A CPN model was constructed constituting a formal specification of the ERDP protocol.
- State space exploration was applied to conduct a formal verification of key properties of ERDP.
- Modelling and verification helped in identifying several omissions and errors in the design.

Edge Router Discovery Protocol

- **Protocol for prefix configuration executed between edge routers and gateways:**



Configuration of a gateway



Periodical multi-cast of unsolicited router advertisement (RA)

Currently no assigned prefixes

Prefixes have a limited lifetime
– must be refreshed
– otherwise they will expire

Unicast router solicitation (RS)

New prefix

Unicast solicited router advertisement (RA)

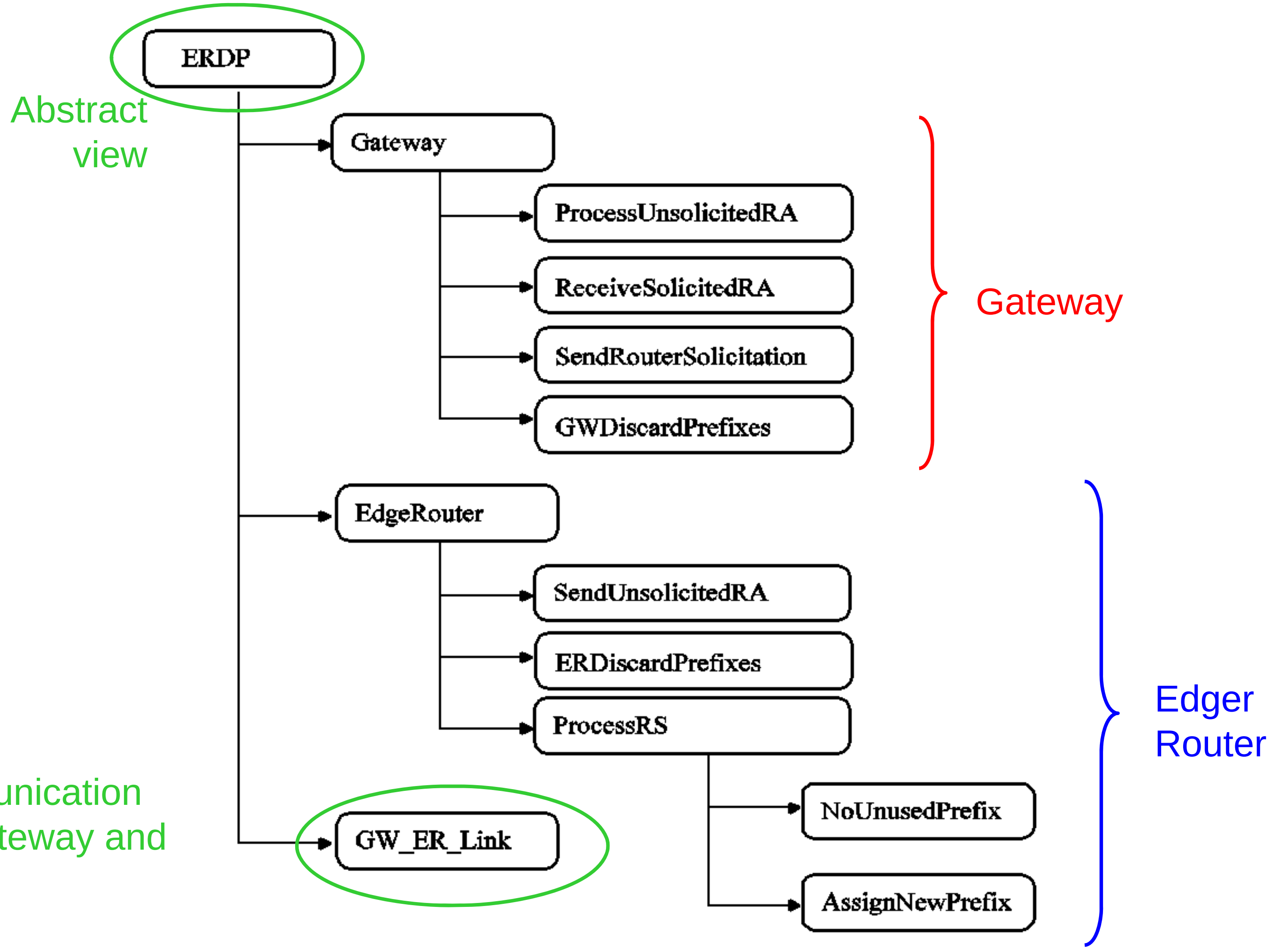
Update prefixes

The Modelling Phase

- **CPN modelling applied for specification of the protocol software design:**
  - First a conventional natural language specification was developed by the protocol software engineers.
  - Next a CPN model reflecting the specification was developed.

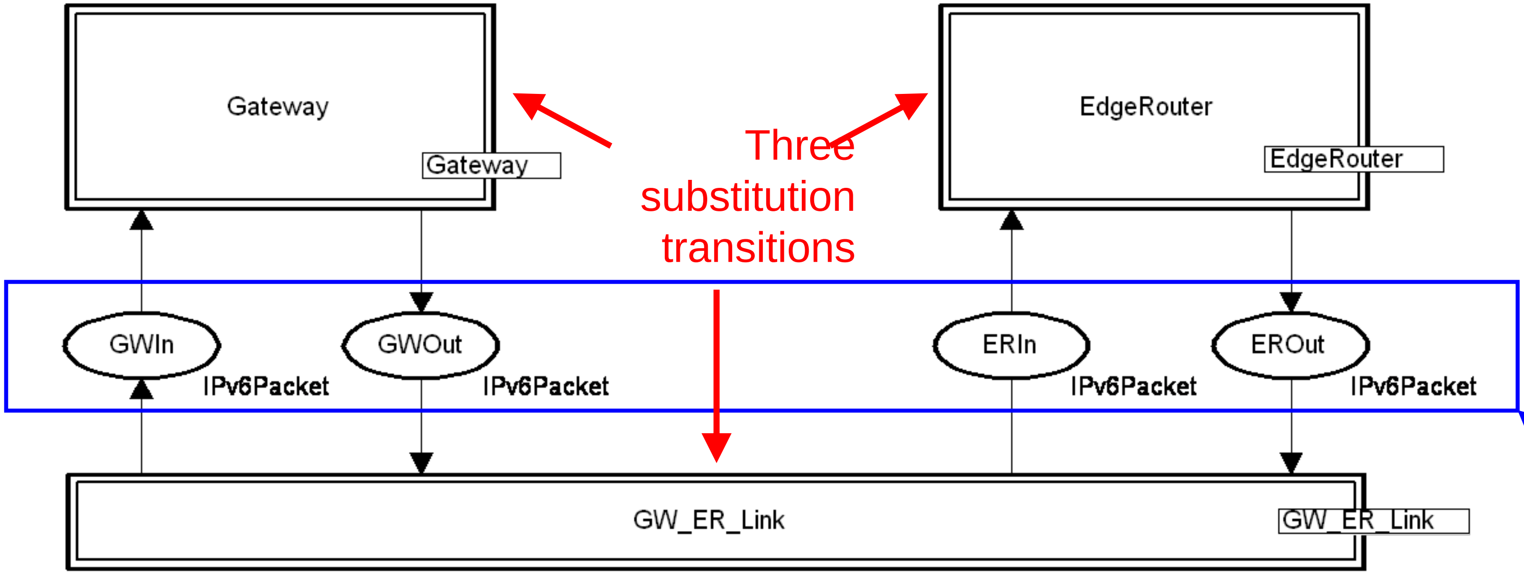
- **The ERDP protocol and the CPN model was then developed in an iterative process:**
  - CPN model discussed and reviewed in each iteration.
  - CPN model used as a basis for discussion of protocol design.
  - Interactive simulation used for detailed investigations of the protocol software.

Module Hierarchy



Wireless communication link between gateway and edge router

ERDP Top-level Module



Three substitution transitions

Four packet buffers

Results from Modelling

- Several software design issues and errors were identified in the modelling phase:

Category	Review 1	Review 2	Total
Incompleteness and ambiguity in the ERDP specification	3	6	9
Errors in the protocol	2	7	9
Simplifications of the protocol	2	0	2
Additions	4	0	4
Total	11	13	24

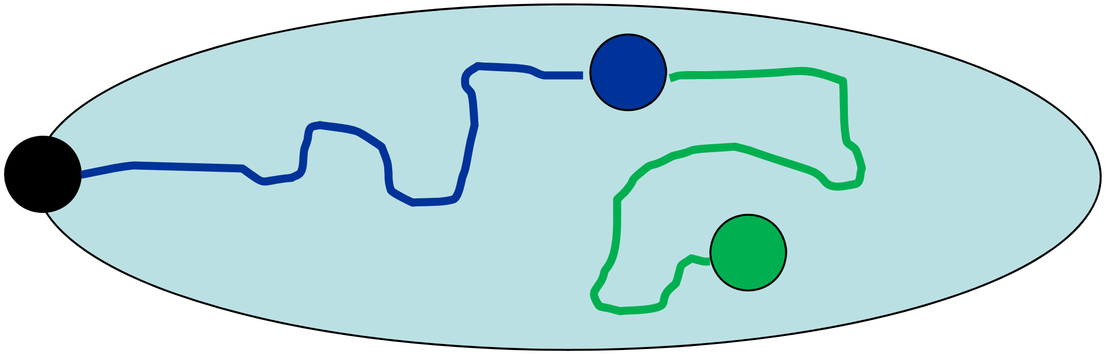
- Approximately 70 person-hours were used on CPN modelling and reviews.

State Space Exploration

- State space exploration was pursued after the three iterations of modelling.
- The first step was to obtain a finite state space:
  - The CPN model above can have an arbitrary number of tokens on the packet buffers.
  - An upper integer bound of 1 was imposed on each of the packet buffers (GWIn, GWOut, ERIn, EROut).
  - This also prevents overtaking among the packets transmitted across the wireless link.
  - The number of tokens simultaneously on the four packet buffers was limited to 2.

Verification of ERDP

- **Key property of the ERDP protocol:**



From any state with a non-configured prefix P it is possible to reach a state where P is consistently configured.

- **Investigated using state space exploration starting from the simplest possible configuration.**

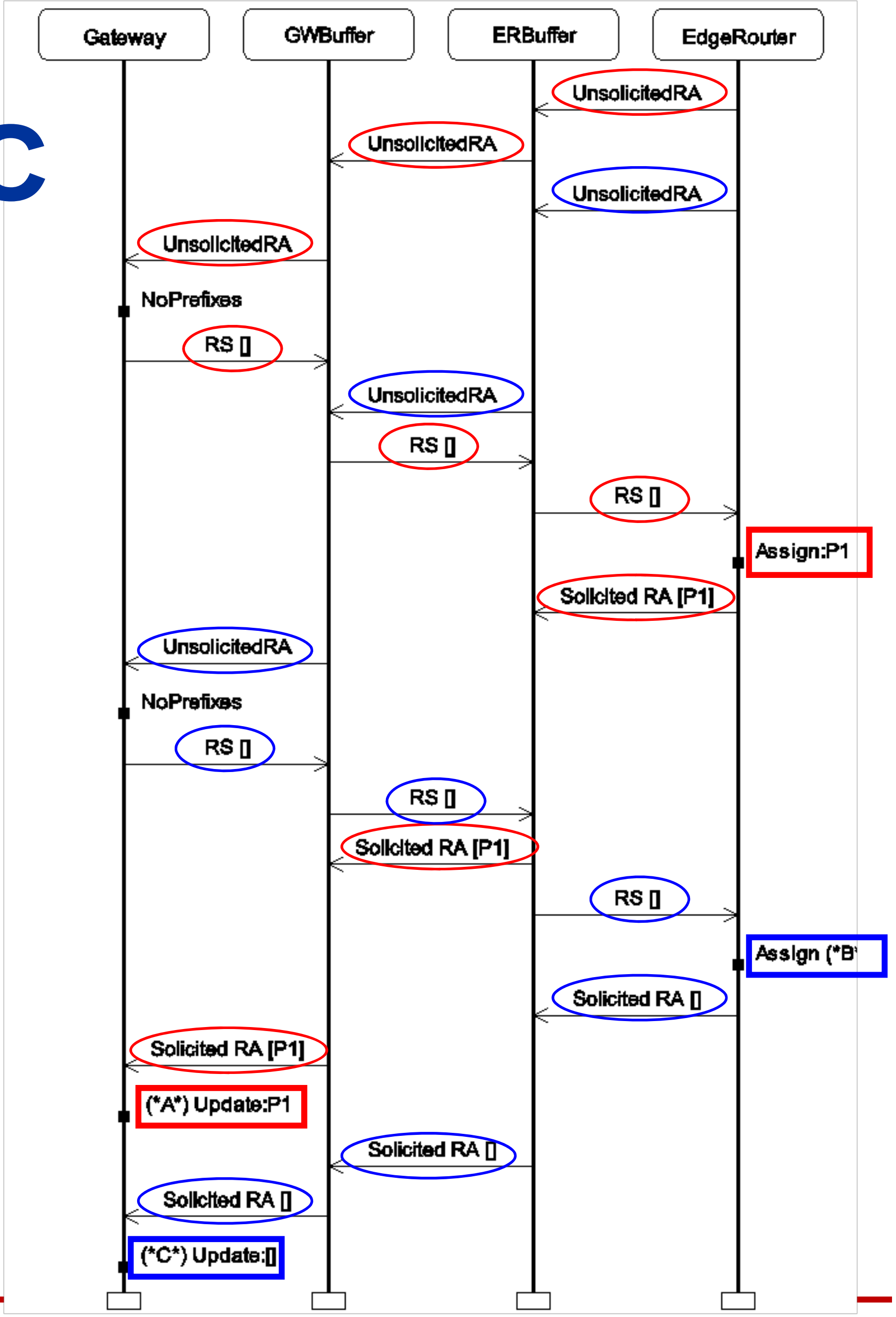
One prefix, no loss, no expiration

- **State space: 46 nodes and 65 arcs.**
- **A single dead marking.**

- **Visual inspection showed that the dead marking is inconsistently configured.**
  - The edge router has assigned a prefix to the gateway.
  - BUT, the gateway is not configured with the prefix.
- **The error-trace was visualised by means of a message sequence chart.**

Error trace MSC

- The edge router sends two unsolicited RAs.
- The first one gets through and we obtain a consistent configuration with prefix P1.
- When the second reaches the edge router there are no unassigned prefixes available.
- A Solicited RA with the an empty list of prefixes is sent.
- The gateway updates its prefixes to be the empty list.



One prefix, no loss, no expiration (rev)

- **The protocol was modified such that the edge router always replies with the list of all currently assigned prefixes.**
- **State space: 34 nodes and 49 arcs.**
- **No dead markings and 11 home markings (constituting a single terminal SCC).**
- **Inspection shows that all home markings are consistently configured with the prefix.**
  - It is always possible to reach a consistently configured state for the prefix.
  - When such a state has been reached, the protocol entities will remain consistently configured.

Results from Verification

- **The verification was conducted in three steps where assumptions were gradually removed.**

- **Step 1 [no packet loss and no expire of prefixes]:**
  - Synchronisation error between edge router and gateway.
  - The error was corrected and the key property was verified.
- **Step 2 [packet loss on wireless link added]:**
  - Synchronisation error when certain unsolicited RAs was lost.
  - Livelock error in processing of router advertisement in gateway.
  - The errors were corrected and the key property was verified.
- **Step 3 [expire of prefixes added]:**
  - Property verified: Consistent configuration always possible.

State Space Statistics

|P|	No loss/No expire		Loss/No expire		Loss/Expire	
1	34	49	68	160	173	531
2	72	121	172	425	714	2,404
3	110	193	337	851	2,147	7,562
4	148	265	582	1,489	5,390	19,516
5	186	337	926	2,390	11,907	43,976
6	224	409	1,388	3,605	23,905	89,654
7	262	481	1,987	5,185	44,450	169,169
8	300	553	2,742	7,181	78,211	300,072
9	338	625	3,672	9,644	130,732	505,992
10	376	697	4,796	12,625	209,732	817,903

- **When a state space has been generated, the verification of the key properties was be done in a few seconds.**

Conclusions

- **Start state space exploration from the simplest possible configurations:**
  - Errors often manifest themselves in the simplest configurations and strongest assumptions.
  - The assumptions are then gradually lifted and larger configurations considered.
- **For the ERDP protocol we did not encounter state explosion.**
- **The key properties could be verified for the number of prefixes that are envisioned to appear in practice.**

Observations

- **State space methods can now be used to validate industrial-sized practical systems.**
- **No state space method will work well on all systems.**
- **Active research area: tomorrow will bring even better state space methods and techniques.**
- **Implications for computer tools:**
  - A computer tool must support a wide range of state space methods.
  - A computer tools must provide a platform for continuously extending the supported methods.

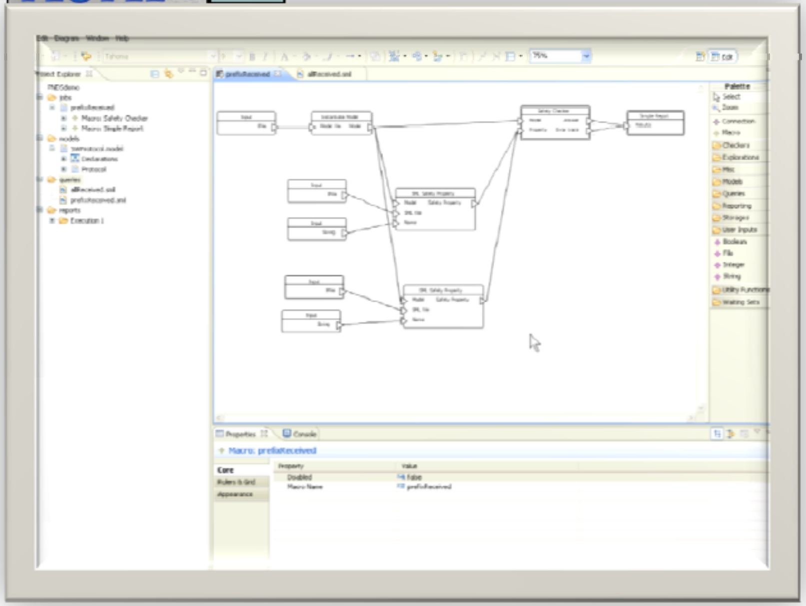
Brief History of State Space Tool Support for Coloured Petri Nets

- **1G - Occurrence Graph Analyzer (OGA) [1992 - 1994]:**
  - d-alone tool based on loading SML images from Design/CPN.
  - l interface and implementation.
  - Su first state space exploration and simple visualisation.
- **2G - Design/CPN e Graph Tool [1994 - 2003]:**
  - Integration of OGA into D
  - Direct support in the graphical
  - Prototype implementations of equiva metry method, time condensed state space, and the swee
- **2.5G - CPN Tools [since 2003]:**
  - Port of Design/CPN Occurrence Graph Tool to CPN Tools.
  - Based on the faster simulation engine used in CPN Tools.

The software architecture made it difficult to support a collection of state space methods in a coherent manner.

The ASAP Platform

- A new model checking computer tool supporting a large collection of state space methods.
- Key features:
  - Graphical specification language for verification jobs.
  - Uniform access to a wide range of state space methods.
  - Allows users to work at different abstraction levels.
  - Support a coherent integration of new state space methods.
  - Verification projects for managing methods, queries, and models.



- **Available via:** www.cs.au.dk/~ascoveco/

ASAP Status

- **ASAP has been developed in the context of the ASCoVeCo Research project.**
- **Implementation started 08/2007.**
- **Current status:**
  - Comback, sweepline, hash compaction, bit-state hashing, full state space exploration, state caching.
  - Safety and LTL (on-the-fly and offline analysis).
  - Reporting facilities, including visualisation of error traces and state spaces.
  - Plug-in architecture for adding new methods.