An Approach to Semi-Automatic Code Generation for the TinyOS Platform using Coloured Petri Nets

Vegard Veiset

Master Thesis

Motivation

Concurrent software systems

- Parallel, Synchronized, Non-deterministic

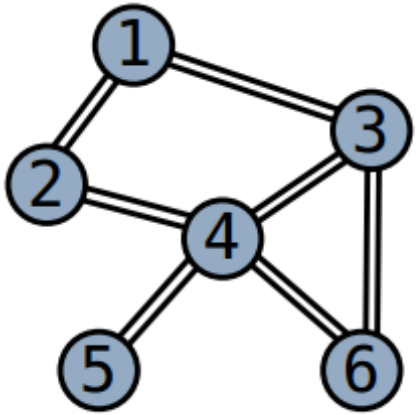
Complex behaviour

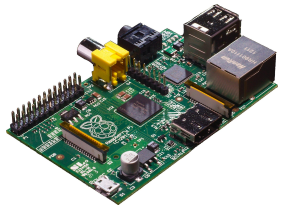
- Model to specify and verify systems

Abstract and platform independent models

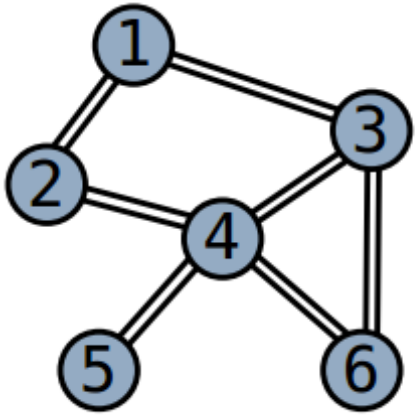
How to get from the model to platform specific code?

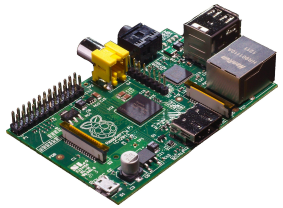
WSN & Roll Protocol



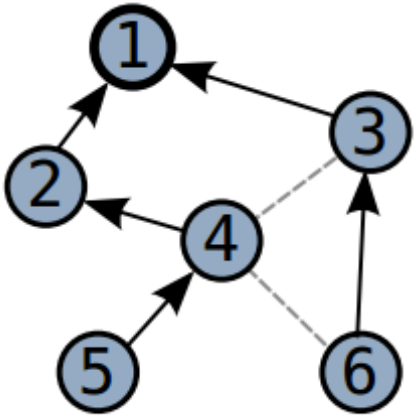


WSN & Roll Protocol

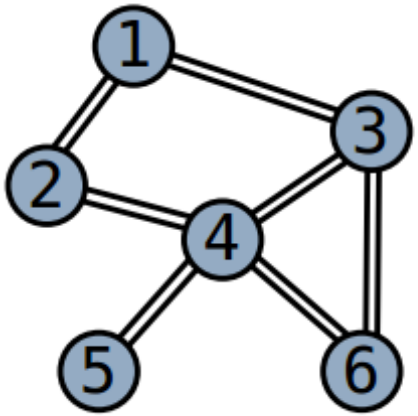


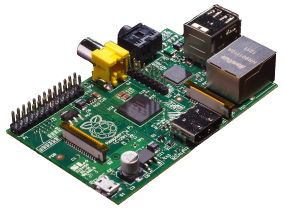


Network as DODAG
(a possible outcome)

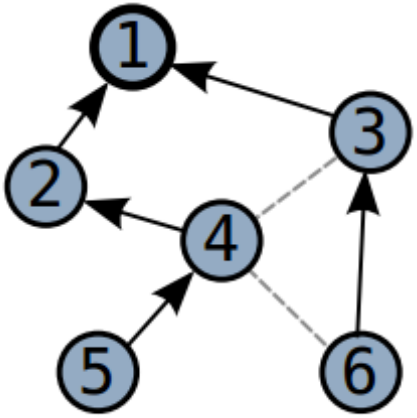


WSN & Roll Protocol

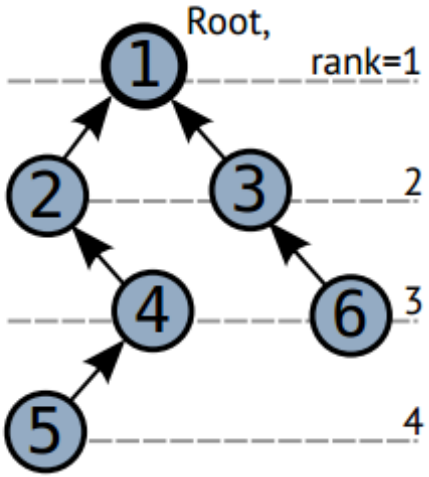




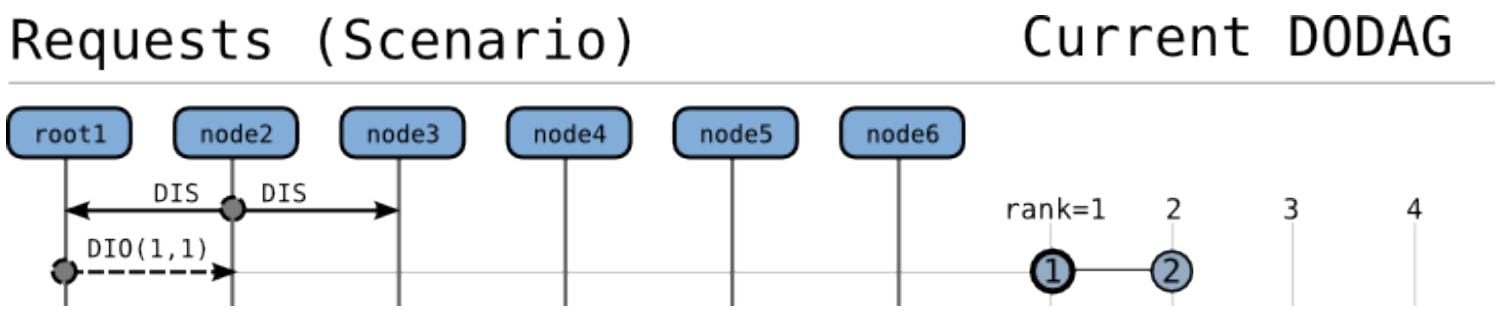
Network as DODAG
(a possible outcome)



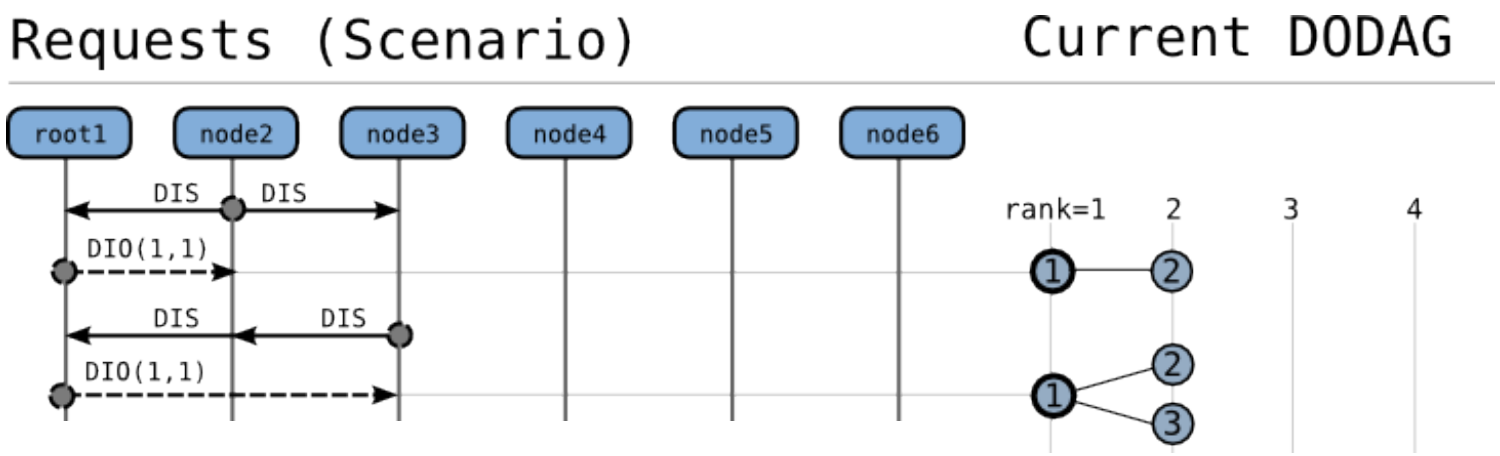
Network as DODAG
(with ranks shown)



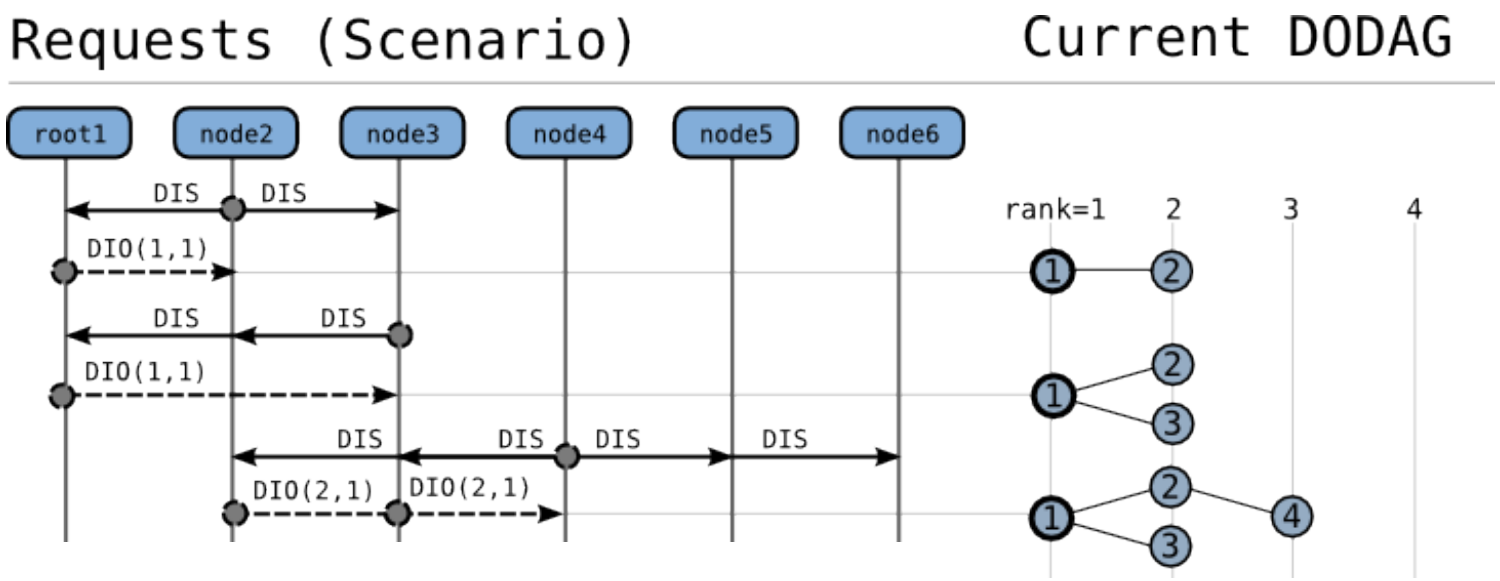
Roll DODAG Creation



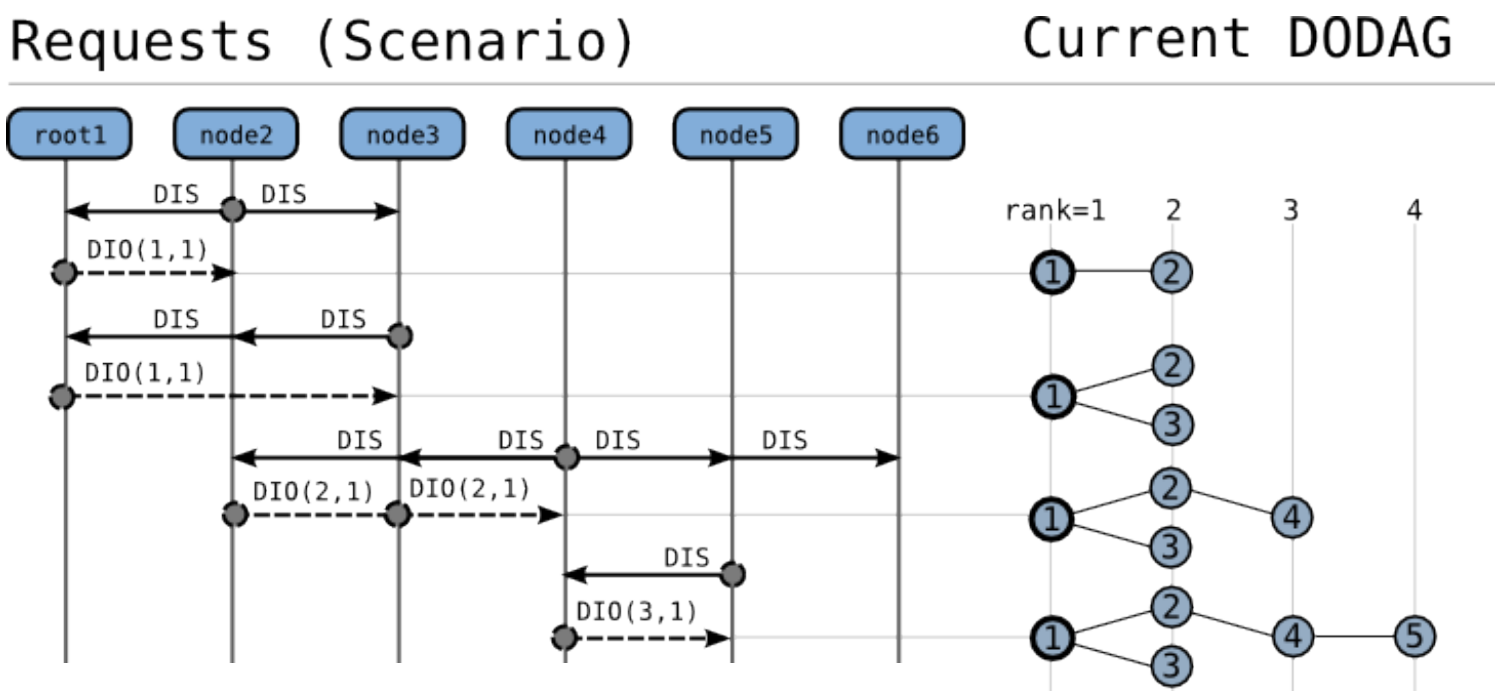
Roll DODAG Creation



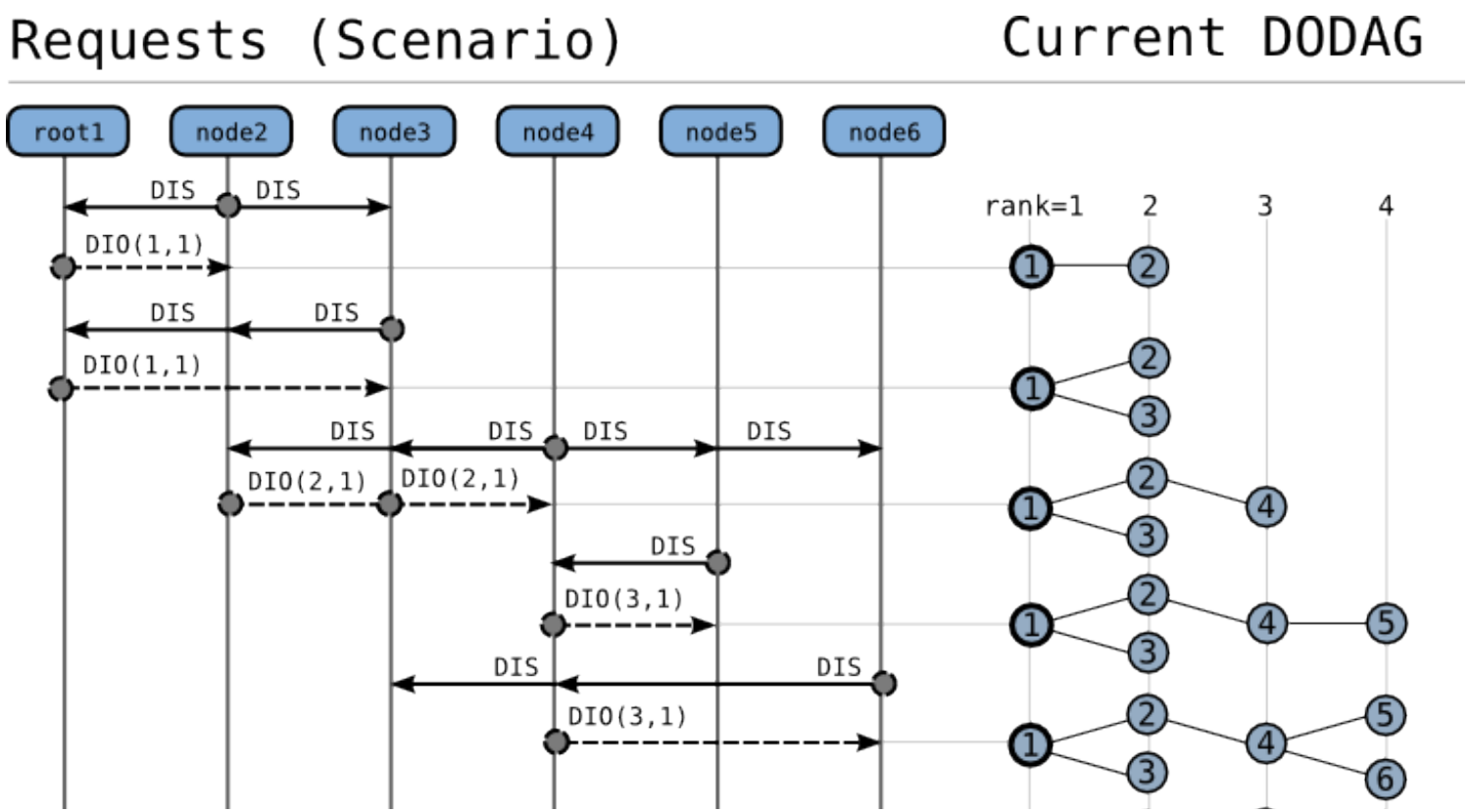
Roll DODAG Creation



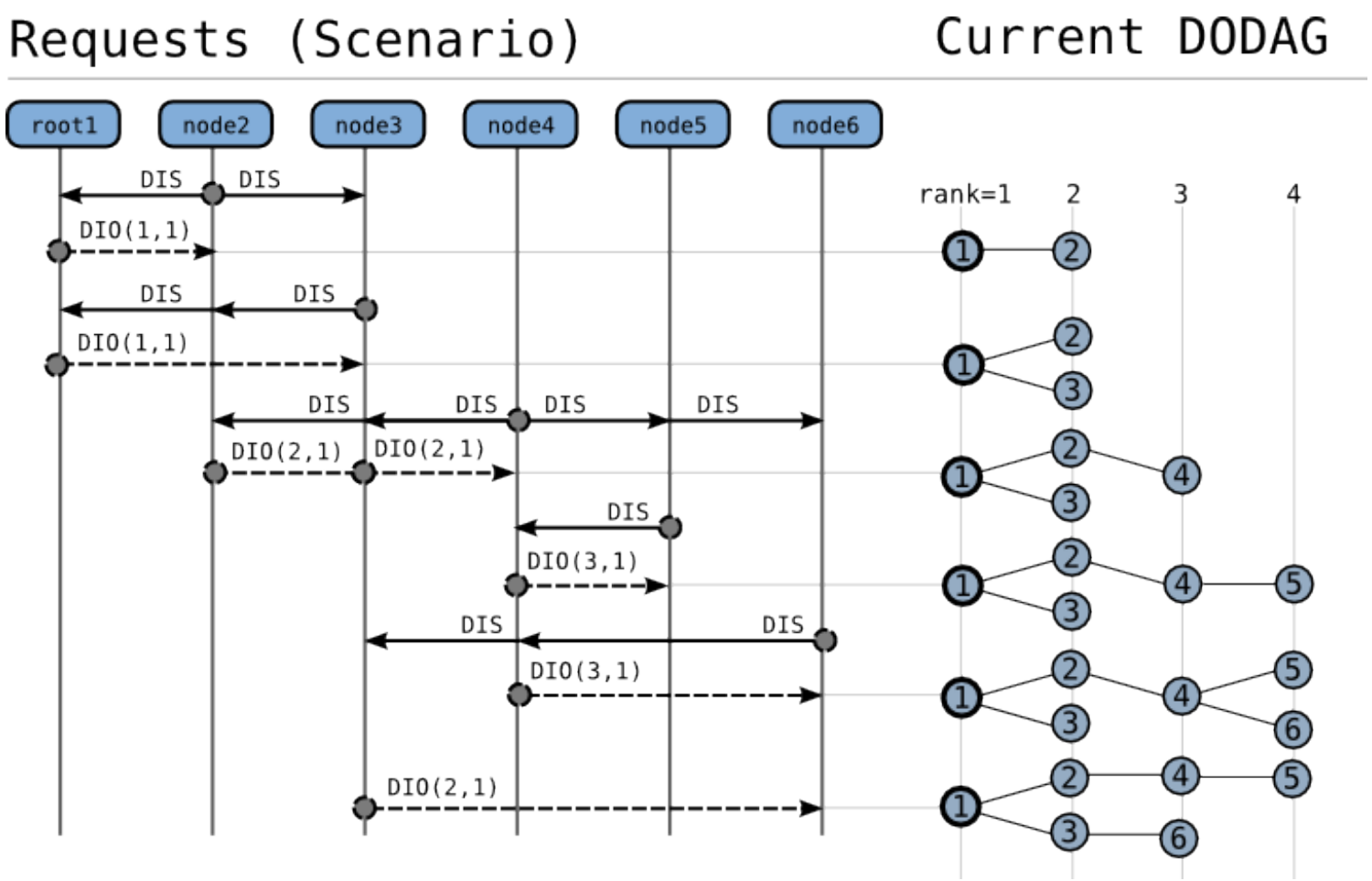
Roll DODAG Creation



Roll DODAG Creation



Roll DODAG Creation

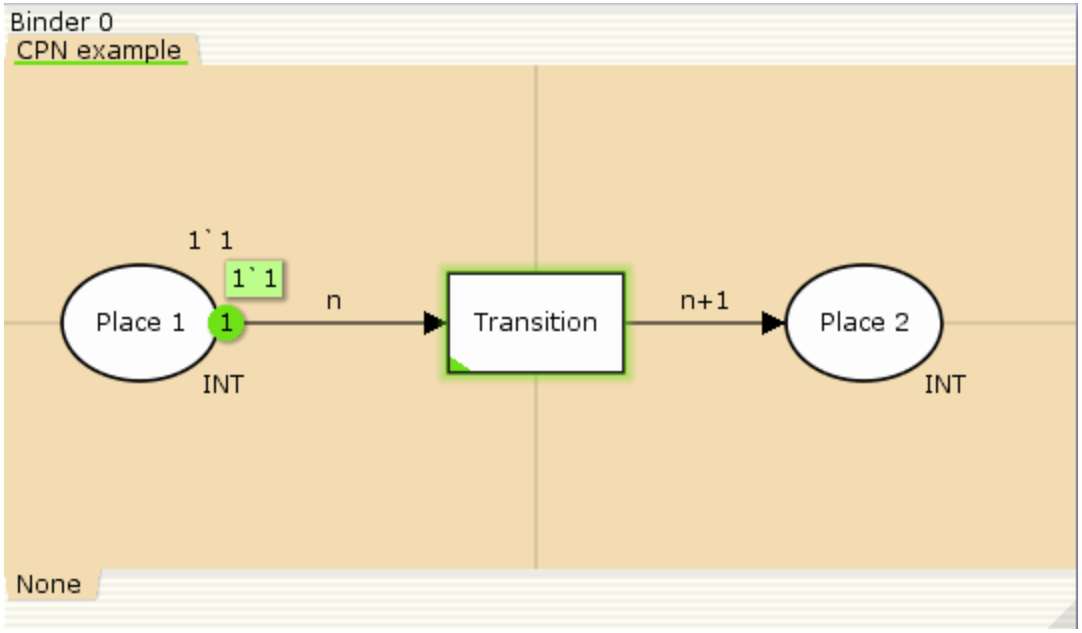


Coloured Petri Net (CPN)

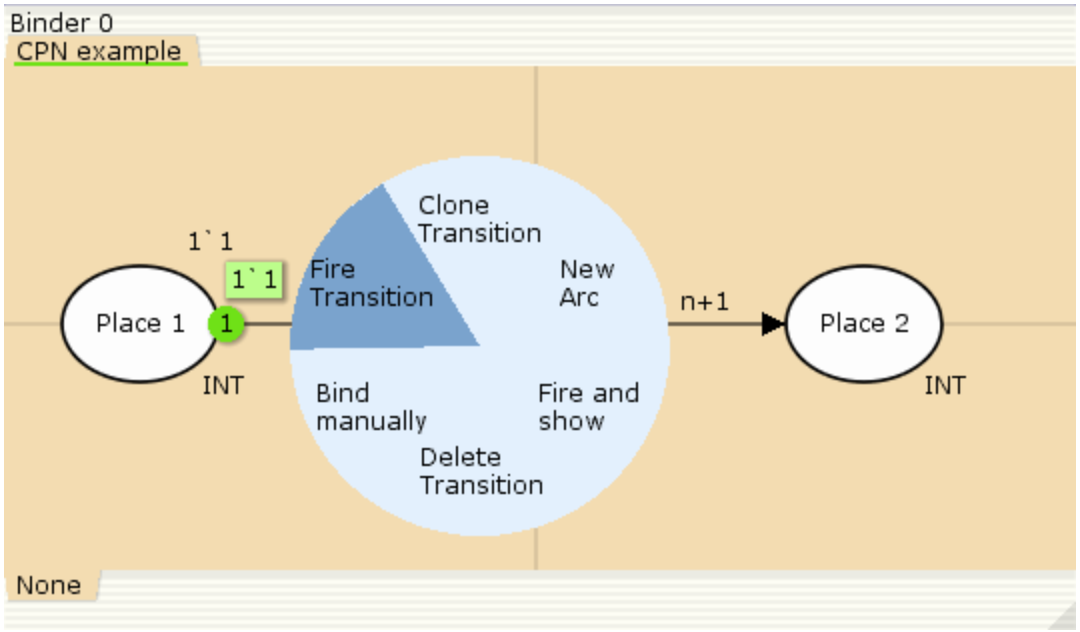
Petri Net + programming language

Suited for modelling of concurrent systems

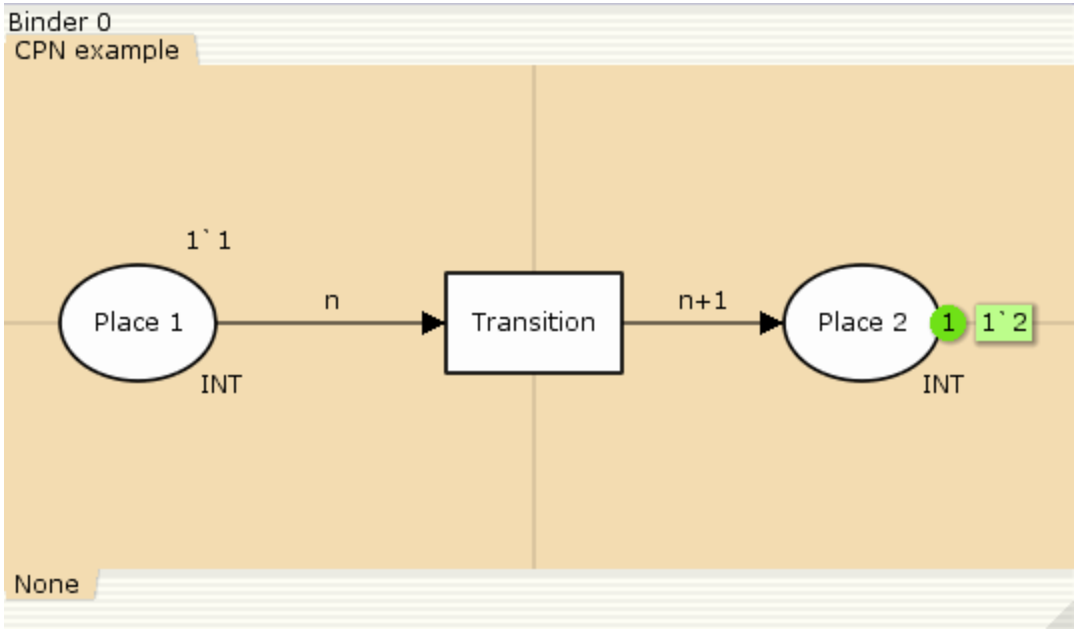
CPN Example


Binder 0
CPN example
1`1
1`1
Place 1
1
INT
n
Transition
n+1
Place 2
INT
None

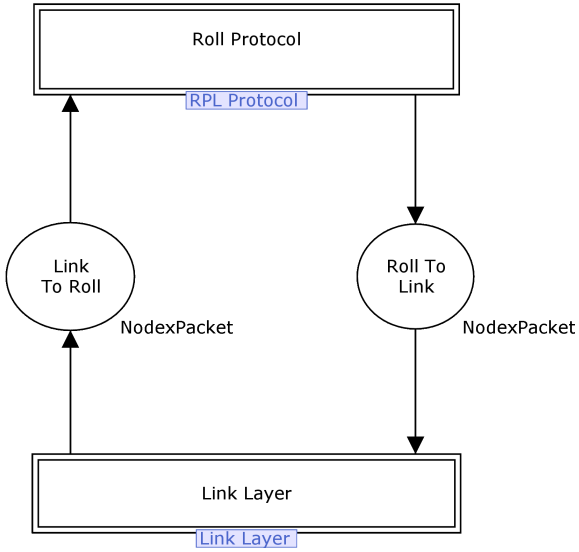
CPN Example


Binder 0
CPN example
1`1
1`1
Place 1
1
INT
Fire
Transition
Clone
Transition
New
Arc
n+1
Place 2
INT
Bind
manually
Fire and
show
Delete
Transition
None

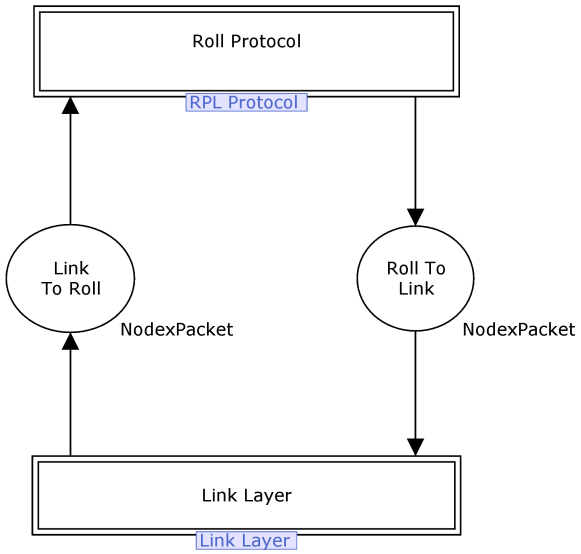
CPN Example

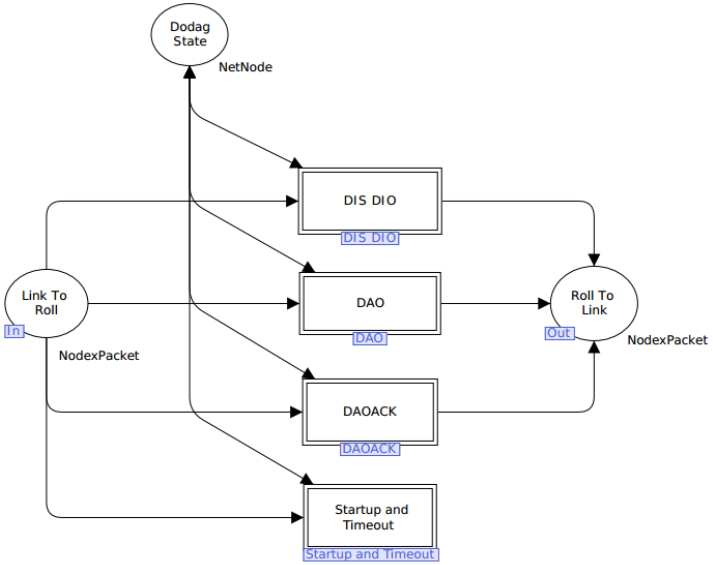

Binder 0
CPN example
1`1
Place 1
INT
n
Transition
n+1
Place 2
INT
1
1`2
None

CPN Roll Protocol Model

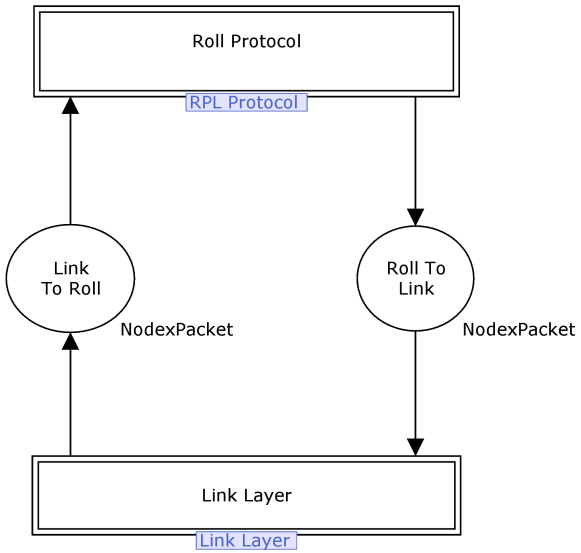


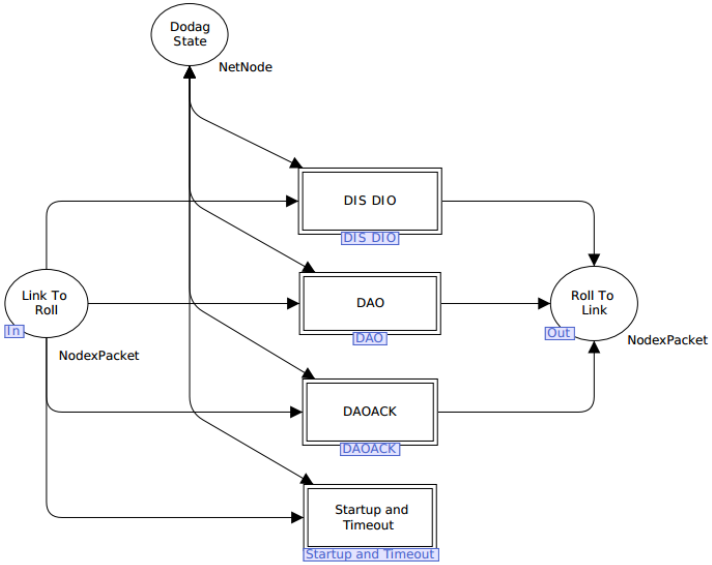
CPN Roll Protocol Model

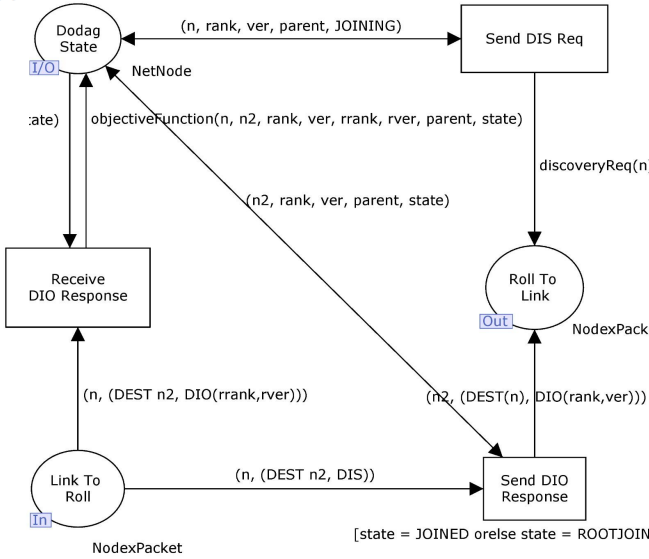




CPN Roll Protocol Model







Research Questions

- How to transform CPN models to code for the TinyOS platform?
- How to refine CPN models for code generation?

- Are pragmatics suitable for the model refinement process?
- What general steps are required in the refinement process?

Related work:
Code Generation from Process-Partitioned CPN Models
Pragmatics on a simple Messaging Protocol

TinyOS and nesC

Targeting WSN: Constrained CPU, RAM and Battery usage

Component based and split-phase programming model

- Events & Commands
- Components & Interfaces
- Configuration

Uses C-like programming language (nesC)

Approach

Using pragmatics

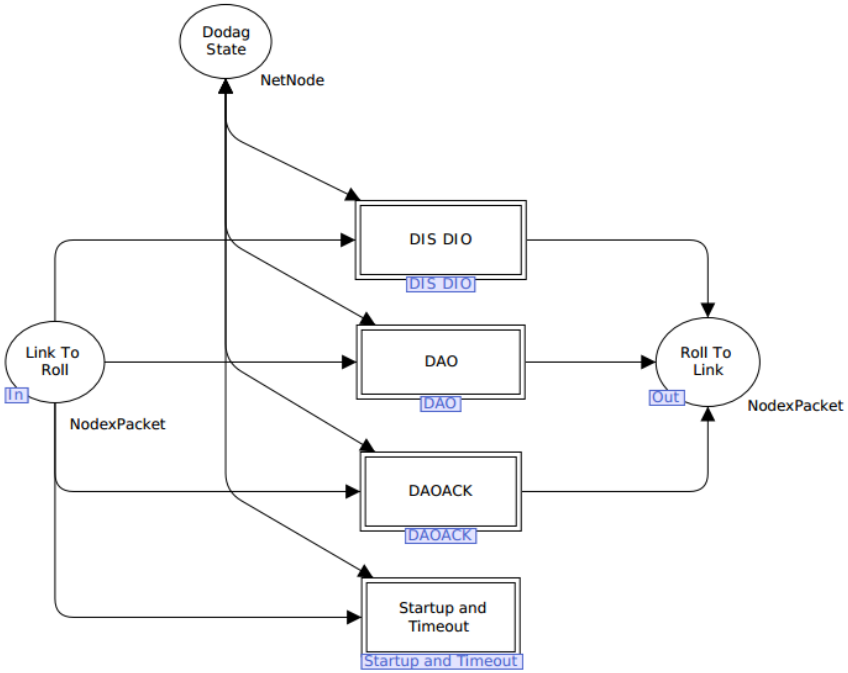
Restructuring model

Refinement Process

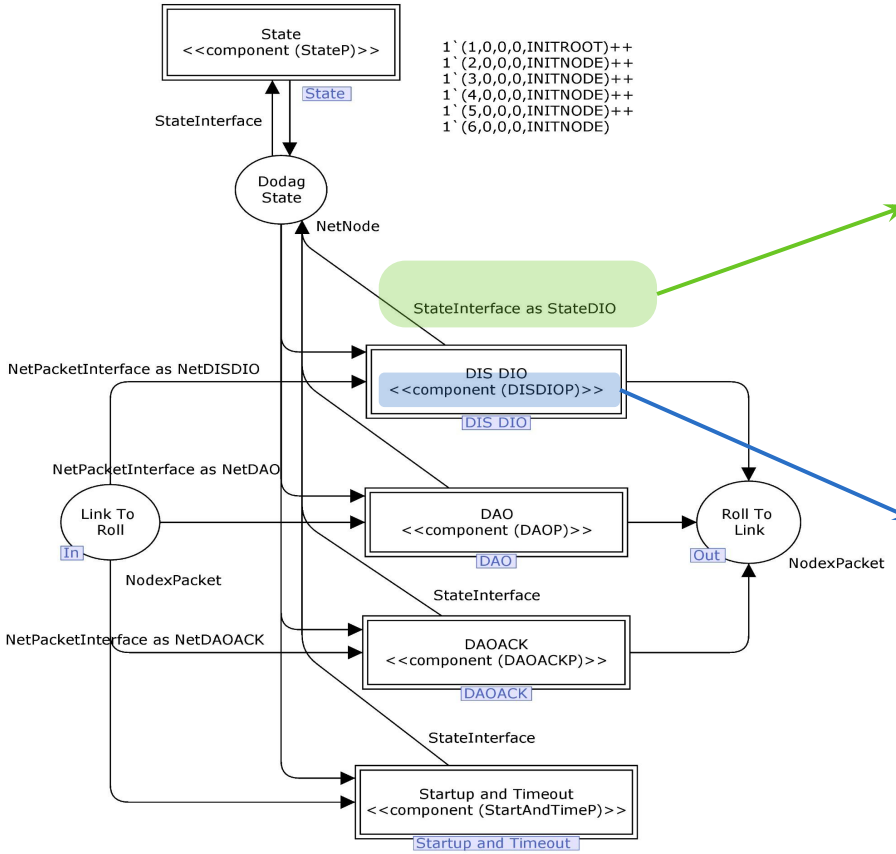
1. Component Architecture
2. Resolving Interface Conflicts
3. Component and Interface Signature
4. Component Classification
5. Internal behaviour

Case study: Original Model

Roll Protocol



1: Component Architecture



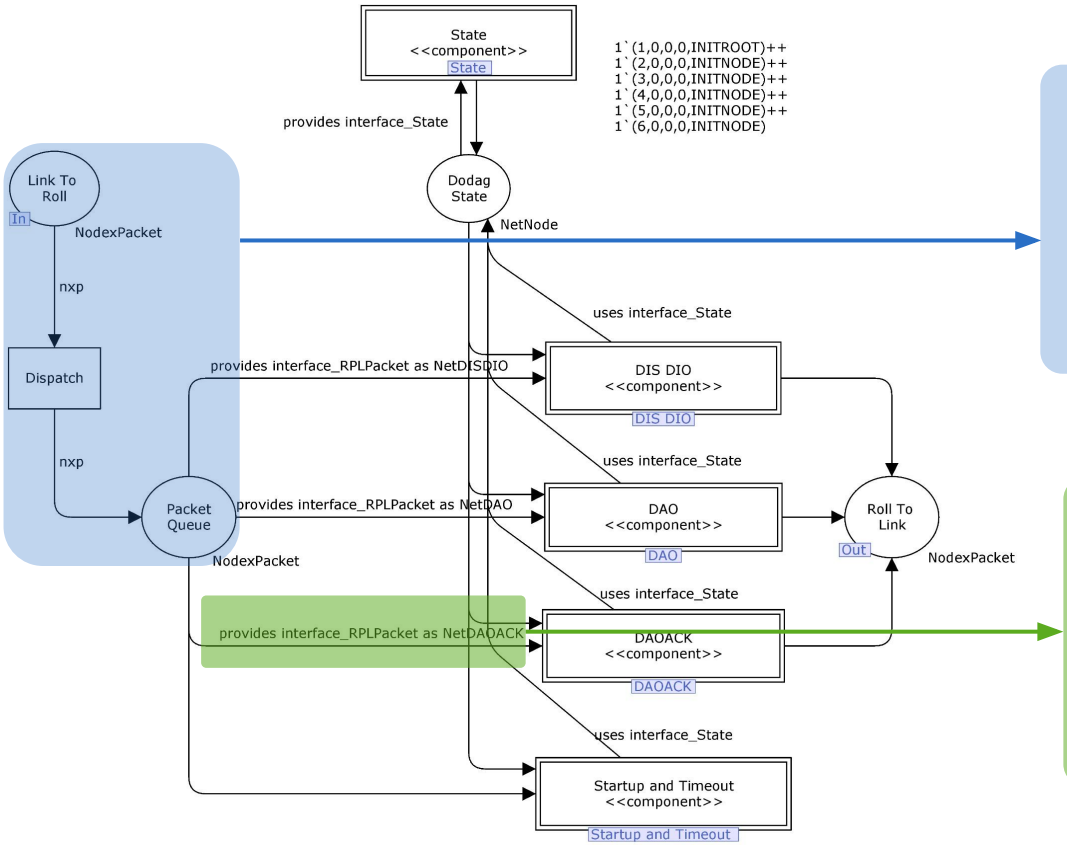
Interface pragmatic

TinyOS interface

Component pragmatic

TinyOS component

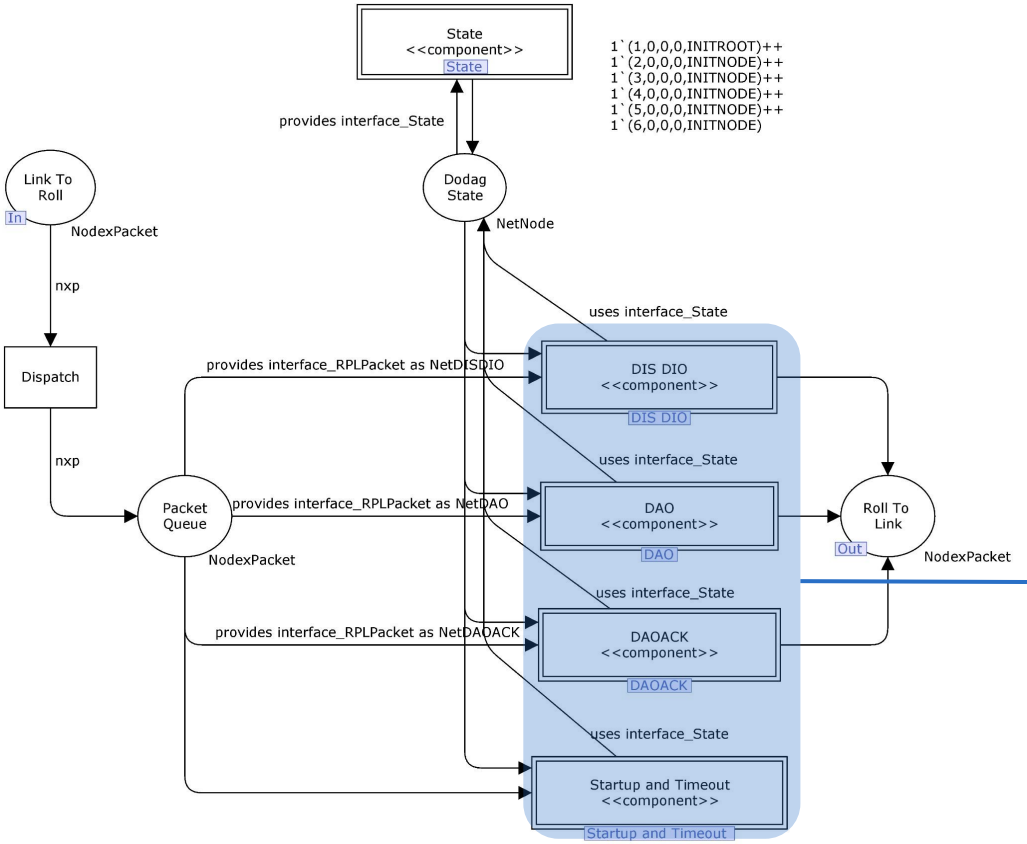
2: Interface Conflicts



Resolving Hierarchy ambiguity

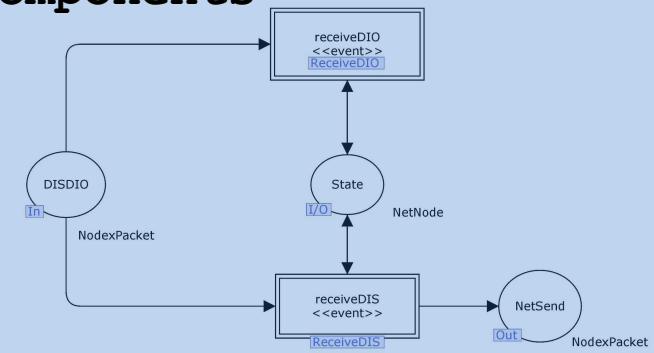
Giving interfaces unique local names

3: Signatures

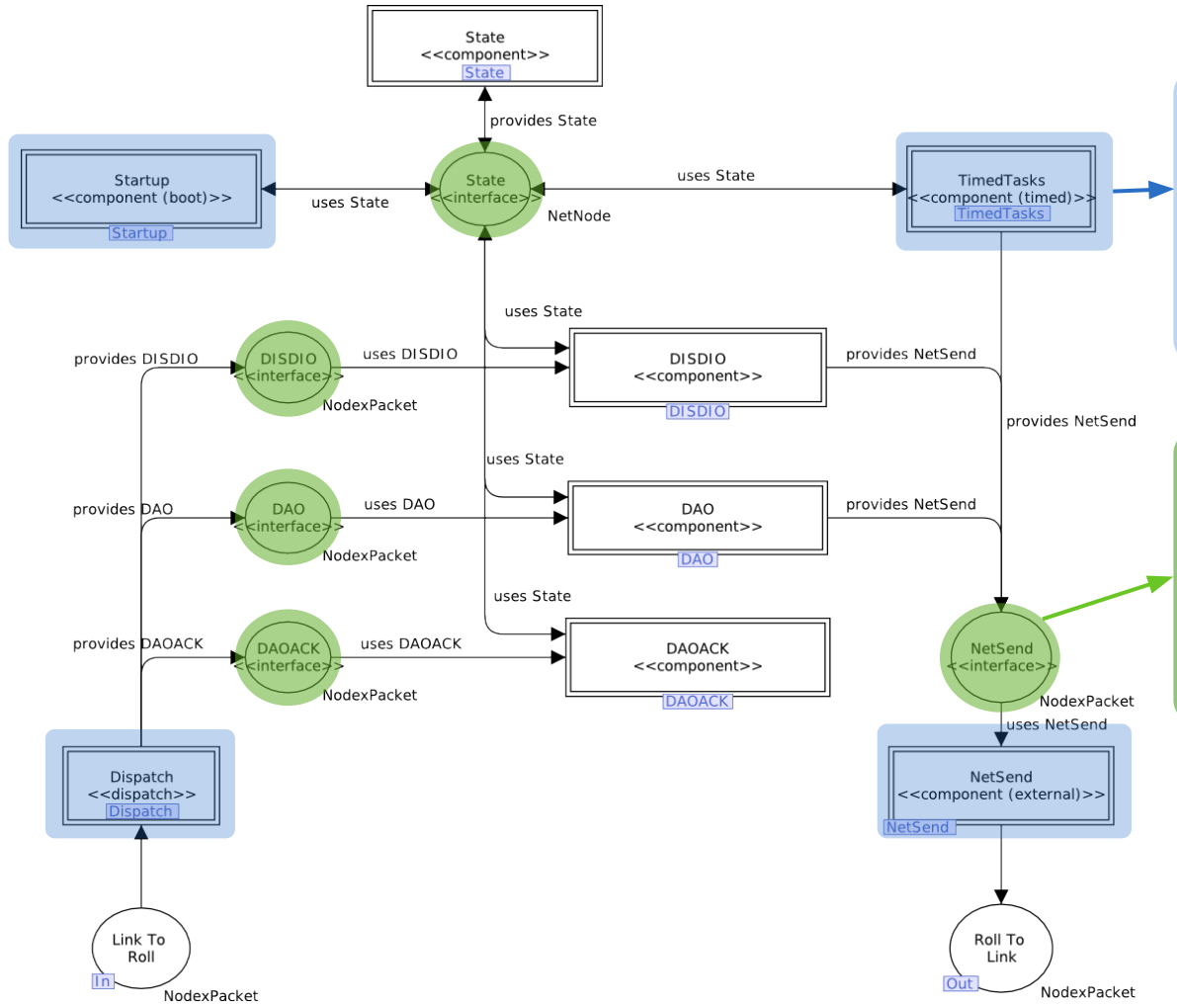


Adding signatures to components and interfaces

Clearer defined components



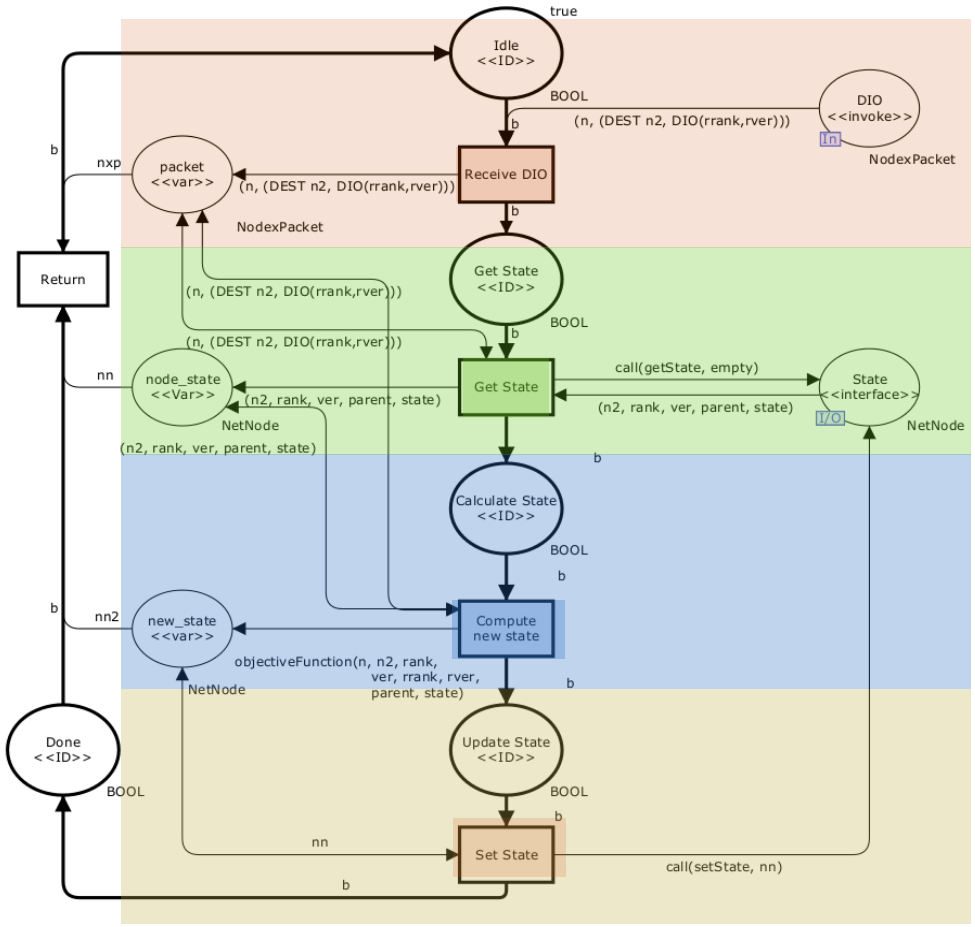
4: Component Classification



Split components into different types

Make interfaces explicit

5: Internal Behaviour



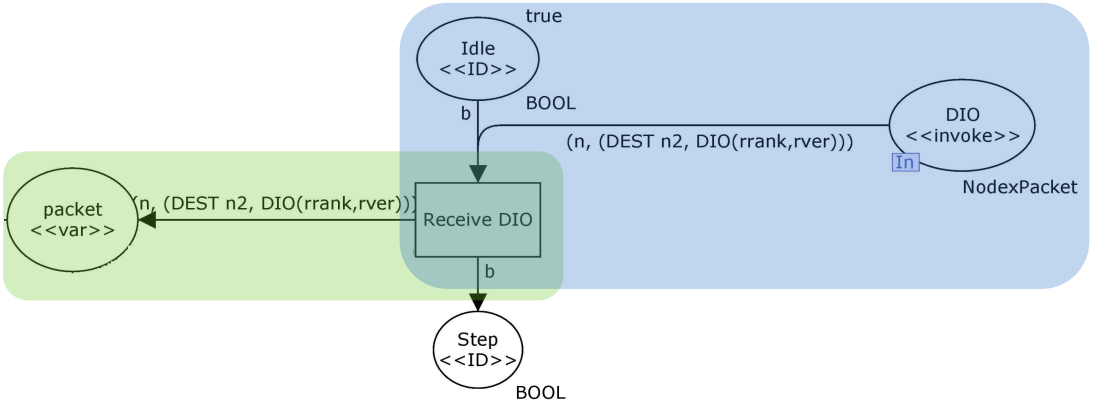

```
Idle\n<<ID>>  [1]
 Receive DIO:
   [Invoke method, Variable assignment]

Step\n<<ID>>  [1]
 Get State:
   [Variable usage, Variable assignment,
    Interface call, Interface returns]

Step2\n<<ID>> [1]
 Compute new state:
   [Variable usage, Variable assignment]

Step4\n<<ID>> [1]
 Set State:
    [Variable usage, Interface call]


Done\n<<ID>>  [0]  [End node: true]
```

Code Generation Patterns

Identified patterns

```
Idle<<ID>> [1]
 Node : If statement false
 Node : End node     false
[Receive DIO]
 Edge : Invoke method        true
 Edge : Variable usage       false
 Edge : Variable assignment  true
 Edge : Interface call       false
 Edge : Interface returns    false
```

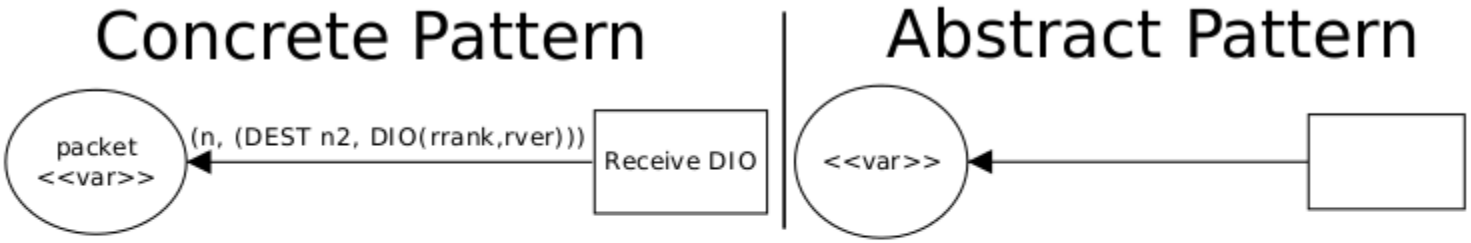
CPN Model



Generated source code

```
event void receiveDIO(NodexPacket var_nodexpacket) {
    packet = var_nodexpacket;
    ...
}
```

Variable Assignment




```
// var = value;
packet = var_nodexpacket;
```

Generated Code

```
event void DISDIO.receiveDIO(NodexPacket var_nodexpacket) {
    NetNode new_state;
    NetNode node_state;
    NodexPacket packet;

    packet = var_nodexpacket;
    node_state = call State.getState();
    new_state = objectiveFunction(...);
    call State.setState(new_state);
}
```

Generated Application

Description	Filename
A configuration file	ConfigurationApp.nc
A nesC makefile	Makefile
A header file mapping colour sets	global.h
A set of components	e.g, DISDIOC, DAOC
A set of corresponding interfaces	e.g, DISDIO, DAO

Conclusions

- A five step refinement process
- Pragmatics for mapping the relationship between CPN and TinyOS
- Generate structure and behaviour
- Generated code is readable

Future Work

- Automated Testing and Analysis
- Improving the Code Generator

Thank you. Questions?