Chapter 1

State Space Analysis of the CPN Web Socket Protocol

One of the advantages of Coloured Petri Nets is the ability to conduct state space analysis, which can be used to obtain information about a model's behavioural properties, and can be used to locate errors and increase confidence in the correctness of the CPN model.

1.1 State Spaces

A state space is a directed graph where each node represents a reachable marking (a state) and each arc represents an occurring binding element (a transition firing with specific values bound to the variables of the transition). CPN Tools by default generates the state space in breadth-first order.

Once generated, the state space can be visualised directly in CPN Tools. Starting with the node for the initial state, one can pick a node and show all nodes that are reachable from it, and in this way explore the state space visually. This can be very tedious and unmanageable for complex state spaces, though, and instead it is usually better to use queries to automate the analysis based on state spaces.

1.1.1 Strongly Connected Component graph

In graph theory, a strongly connected component (SCC) of a graph is a maximal subgraph where all nodes are reachable from each other. An SCC graph has a node for each SCC of the graph, connected by arcs determined by the arcs in the underlying graph between nodes that belong to different

SCCs. An SCC graph is acyclic, and an SCC is said to be trivial if it contains only one node.

By calculating the SCC graph of the state space, some of the further analysis becomes simpler and faster, such as determining reachability and cyclic behaviour.

1.1.2 Application of State Spaces

The biggest drawback of state space analysis is the size a state spaces may become very large. The number of nodes and arcs often grows exponentially with the number of starting system parameters.

Size of the midel configuration considered?

This can be remedied by picking smaller configurations, that encapsulate different parts of the system. This was necessary with the WebSocket Protocol model, as the state space took too long to generate initially.

A variant of this is situations where tokens can be generated an unlimited amount of times on a place, making the state space infinite. This can be remedied by modifying the model to limit the number of simultaneous tokens in the offending place.

Diskutere mer hva som måtte gjøres med WebSocket, illustrasjon

A model that incorporates random values is not fit for computing a state space, as the number of possible enabled bindings are arbitrary for a given state depending on the possible random values. Some values might not get used. Other times there is an infinite number of possible values (like a floating point number).

utpeke forskjell på non-deterministic og random

This can sometimes be alleviated by changing the behaviour to be deterministic, for example by replacing the random function with a small color set, such as an index or an integer with bounds. This lets the state space generator create bindings for every possible value.

For the WebSocket Protocol model, this was a problem for the masking key in WebSocket frames, which is supposed to be a random 4-byte string. The randomisation function was simply changed to always return four zeros.

1.2 State space report

Once the state space has been generated, CPN Tools lets the user save a state space report as a text document. The report is split into parts that each describe different aspects about the state space.

To explain each section of the state space report, a simple report for the WebSocket protocol has been generated, in a configuration where no messages are set to be sent. Thus, the only thing that will happen is that a

connection will be established. Later in the chapter we will consider more elaborate configurations of the WebSocket protocol.

1.2.1 Statistics

The first section describes general statistics about the state space.

```
State Space
   Nodes:  17
   Arcs:   16
   Secs:   0
   Status: Full

Scc Graph
   Nodes:  17
   Arcs:   16
   Secs:   0
```

This state space has 17 possible markings, with 16 enabled transition occurrences connecting them. There is one more node than there are arcs, which means this graph is a tree.

The `Secs` field shows that it took less than one second to calculate this state space, while the `Status` field tells whether the report is generated from a partial or full state space.

We also see that the Scc Graph has the same number of nodes and arcs, meaning that there are no cycles in the state space (although this was already known from the fact that it is a tree).

1.2.2 Boundedness Properties

This section describes the minimum and maximum number of tokens for each place in the model, as well as the actual tokens these places can have. The text has been reformatted and truncated for readability.

```
Best Integer Bounds
                          Upper       Lower
   ClientApplication
        Active_Connection 1
                          1           0
        Conn_Result 1
                          1           0
        Connection_failed 1
                          0           0
        Messages_received 1
                          1           1
        Messages_to_be_sent 1
```

```
                          0          0
    .....

Best Upper Multi-set Bounds
    ClientApplication
         Active_Connection 1
                        1`()
         Conn_Result 1
                        1`success
         Connection_failed 1
                        empty
         Messages_received 1
                        1`[]
         Messages_to_be_sent 1
                        empty
    .....
    ClientWebSocket
         Connection_status 1
                        1`CONN_OPEN
    .....
    ServerWebSocket
         Connection_Status 1
                        1`CONN_OPEN
    .....

Best Lower Multi-set Bounds
    ClientApplication
         Active_Connection 1
                        empty
         Conn_Result 1
                        empty
         Connection_failed 1
                        empty
         Messages_received 1
                        1`[]
         Messages_to_be_sent 1
                        empty
    .....
    ClientWebSocket
         Connection_status 1
                        empty
    .....
    ServerWebSocket
         Connection_Status 1
                        empty
    .....
```

Many places show a lower and upper bound of 1. This shows a weakness in the approach of using lists to facilitate ordered processing of tokens:

We can't see the actual number of tokens that are in the place, because technically there is just a list there.

Apart from that, we see that both the client and the server has an open connection at some point, as the `Connection_status` place on the `ClientWebSocket` and `ServerWebSocket` pages have both had a `CONN_OPEN` token.

1.2.3 Home Properties

This section shows all home markings. A home marking is a marking that can always be reached no matter where we are in the state space.

Vise marking

```
Home Markings
   [17]
```

We see that there is one such marking at node 17. From earlier we know that the state space is a tree, and if this node is always reachable it must be a leaf and all the other nodes must be in a chain. This tells us that there is only one possible sequence of transitions to establish a connection. We can then confidently say that the model works correctly with this configuration.

1.2.4 Liveness Properties

This section describes liveliness of the state space. Some of the transitions have been omitted for readability.

```
Dead Markings
   [17]


Dead Transition Instances
   ClientApplication'Fail 1
   ClientApplication'Receive_data 1
   ClientApplication'Send_data 1
   ClientWebSocket'Filter_messages 1
   .....

Live Transition Instances
   None
```

A dead marking is a marking from where no other markings can be reached. In other words, there are no transitions for which there are enabled bindings, and the system is effectivley stopped. For our example, we have a single dead marking, and it is the same as our home marking, confirming that this is a leaf node in the tree.

from where? for which? formuler

We also get a listing of dead transition instances, which are transitions that never have any enabled bindings and are thus never fired. This can be useful to detect problems with a model, but in this example it is expected for many of the transitions, since we are not sending any kind of messages in this configuration.

Last, there are live transition instances. A transition is live if we from any reachable marking can find an occurrence sequence containing the transition. Our example has no such transition, which follows trivially from the fact that there is a dead marking.

The State Space report also contains fairness properties, but this does not apply to our model since it contains no loops. We will not explain more about this here, and instead refer you to [JK09] chapter 7 for more information.

1.3 TODO:

Skrive om resten av analysene.

Flette inn dette:

1.3.1 Error discovery

An error in the model was discovered this way, in the Unwrap and Receive module, where the pong reply was adding a new list instead of appending to the old one in outgoing messages.

Flere detaljer - hvordan viste fielen seg? Figur, før og etter

Chapter 2

Technology and foundations

One of the first decisions that had to be made for this thesis was whether to base the work on an existing platform or to create a new one from scratch. In this chapter we will describe the reasoning behind our choices, and give an overview of the technologies that have been used.

2.1 (decision)

Finne på tittel

A simple but easy way of manipulating a CPN model is by representing it as a tree. Simple tree editors are a feature of most GUI software platforms. Even so, we realised early that writing everything from scratch would probably take much longer than adapting an existing platform.

There are of course many complete implementations of Petri Net tools in different languages and toolkits, but few of them are open source, or written with extensibility in mind. If we were to base our work on an existing platform, it would have to be open and extendable.

To narrow our search, we limited our options to solutions in languages we had experience with: Java, c++/Qt and Ruby. Java is a popular language, and we already have Access/CPN, a part of the CPN Tools project, which can parse .cpn files into Java objects.

By searching the web, we discovered the ePNK framework, an extendable framework for working with Petri Nets in a graphical manner, and lets you specify your own Petri Net type. It is built on the Eclipse Modeling Framework (EMF) (which Access/CPN is also built on).

We also needed a way to represent pragmatics. It was suggested to try an ontology-based approach, and we decided on SADL, another Eclipse plugin that lets us easily define and work with ontologies.

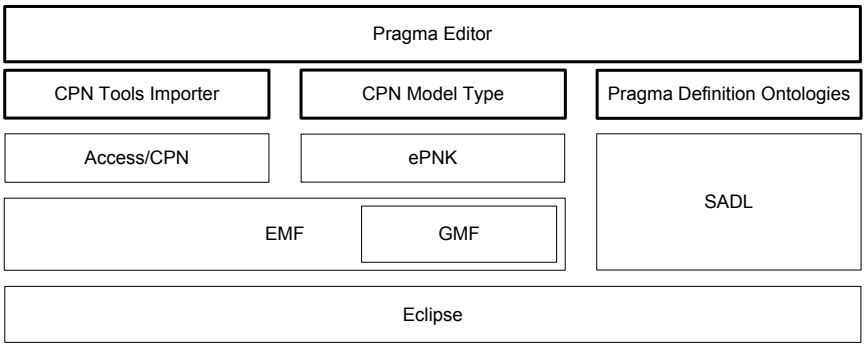

Figure 2.1: Application Overview Diagram

Fig. 2.1 shows the different elements that make up the application. The elements with bold frames are the ones newly created for this thesis, while the rest below are the existing solutions used and built upon. These will be described in the following sections, from the bottom up.

2.2 Eclipse IDE

Eclipse IDE is an open source, cross-platform, polyglot development environment. Its plugin framework makes it greatly extendable and customisable, and especially makes it easy for developers to quickly create anything from small custom macros, to advanced editors, to whole applications. The Eclipse IDE is open source, and part of the Eclipse Project, a community for incubating and developing open source projects.

The Eclipse IDE is built on the Eclipse Rich Client Platform (RCP), Fig. 2.2. At the bottom of this we have the Platform Runtime, based on the OSGi framework, which provides the plugin architecture. The other plugins shown in the diagram together form a basic generic IDE.

The principal Eclipse distribution is the Eclipse Java IDE, which is one of the most popular tools for developing all kinds of Java applications, from small desktop applications, to mobile apps for Android, to web applications, to enterprise-scale solutions.

Plugins are the building blocks of Eclipse, and there exists a wide range of plugins that add tools, functionality and services. For example, this thesis was written in LaTeX using the Texlipse plugin, and managed with the Git

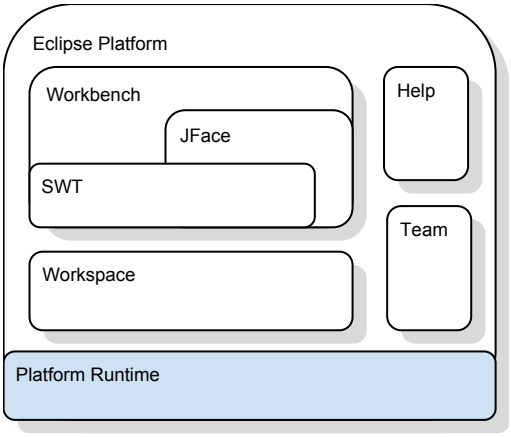

Figure 2.2: The Eclipse RCP

version control system through the EGit plugin.

Publishing a custom plugin is simple. By packaging it and serving it on a regular web server, anyone can add the web server url to the update manager in Eclipse, and it will let you download and install it directly, as well as enabling update notifications.

It is possible to package Eclipse with sets of plugins to form custom editions of Eclipse that are tailored for specific environments and programming languages. Aptana Studio is one example, aimed at Ruby on Rails and PHP development.

2.3 Eclipse Modeling Framework

EMF is a framework for Model Driven Development (MDD) in Java. It is an Eclipse plugin that is part of the Eclipse Platform, and open source. By providing modeling and code generation tools, it lets developers create model specifications that can be converted to Java classes, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

prøve å utvide, men vanskelig å finne noe mer fornuftig å skrive

2.3.1 Graphical Modeling Framework

GMF builds on EMF to provide graphical viewing and editing of models. It uses metamodels created with EMF to generate implementations of views and editors that can create and edit the respective models.

2.4 ePNK: Petri Net framework

ePNK is an Eclipse plugin both for working with standard Petri Net models, and a platform for creating new tools for specialised Petri Net types, which is exactly what we need for our annotated CPN. It uses EMF and GMF to work with the Petri Net models and provide generic editors for custom Petri Net variants.

There are several reasons why ePNK is a good choice:

referanse?

- It saves models using the ISO/IEC 15909 standard file format PNML,
- It is currently actively developed,
- It is designed to be generic and easily extendable by creating new model types, and
- It includes both a tree editor and a graphical editor, provided through GMF.

ePNK includes definitions for the core PNML model type, as well as two subtypes of Petri Nets. The first is P/T-Nets, or Place/Transition Nets, which expand on the core model with a few key items: initial markings for places, inscriptions on arcs, and constraining arcs to only go between a place and a transition (this is not enforced in PNML, as there are Petri Net variants that allow this).

The other is High level Petri nets (HLPNG). This type adds several more labels, all of which are Structured Labels. These are parsed and validated using a syntax that is inspired from (but not the same as) CPNML from CPN Tools. It is possible to write invalid data in these labels and still save the document, as they will only be marked as invalid

Neither of these two conform exactly to the Coloured Petri Nets created by CPN Tools. HLPNG comes close, but is missing a few things like ports and sockets (RefPlaces can emulate this), and substitutin transitions. Also, the structured labels are not compatible with CPNML syntax from CPN Tools, and for our prototype, these structured labels are not necessary with regard to annotations. It is possible that this might be useful in a future version, where for example pragmatics are available depending on things like the colorset of a place or the variables on an arc, but this would take too much time to implement.

Bedre formulering av siste setning?

Our decision was to develop our own Petri net type, which matches CPN Tools as close as possible.

2.5 Access/CPN: Java interface for CPN Tools

CPN Tools has a sister project called Access/CPN. This is an EMF-based tool to parse .cpn files and represent them as an EMF-model. The .cpn files saved by CPN Tools are XML-based, which makes them easy to parse, but having an existing solution for this is preferrable.

The model definition used by Access/CPN is very similar to that of ePNK.

Diskutere mer i implementation?

TODO: Vurdere å utvide med egen algoritme som konverterer direkte til ePNK?

2.6 SADL: Ontologies

Ontologies are a way to present information and meta-information so that it can be understood by computers. Essentially, this is done by defining classes that have properties, relations and constraints, and then association information with these classes.

There is a lot of ongoing research on this subject, especially to create a semantic web, that is extending web pages to provide meta-information about the content they contain and enabling software to understand it and reason about it.

SADL is an Eclipse plugin that defines an english-like syntax for defining ontologies, and comes with a text editor that features syntax highlighting, parsing and validation. This is useful as we can possibly reuse the editor in our plugin for defining model-specific pragmatics.

It also has tools to parse and reason with these ontologies, which we will use to filter and validate which pragmatics are available for different model entities.

2.7 Summary

After picking these technologies, since all the componets have Eclipse in common, it was an easy decision to develop our project as an Eclipse plugin. This also let us centralise all our development in Eclipse.

Bibliography

[JK09] Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.