Software 9

Group deis902e17 - P9 Project

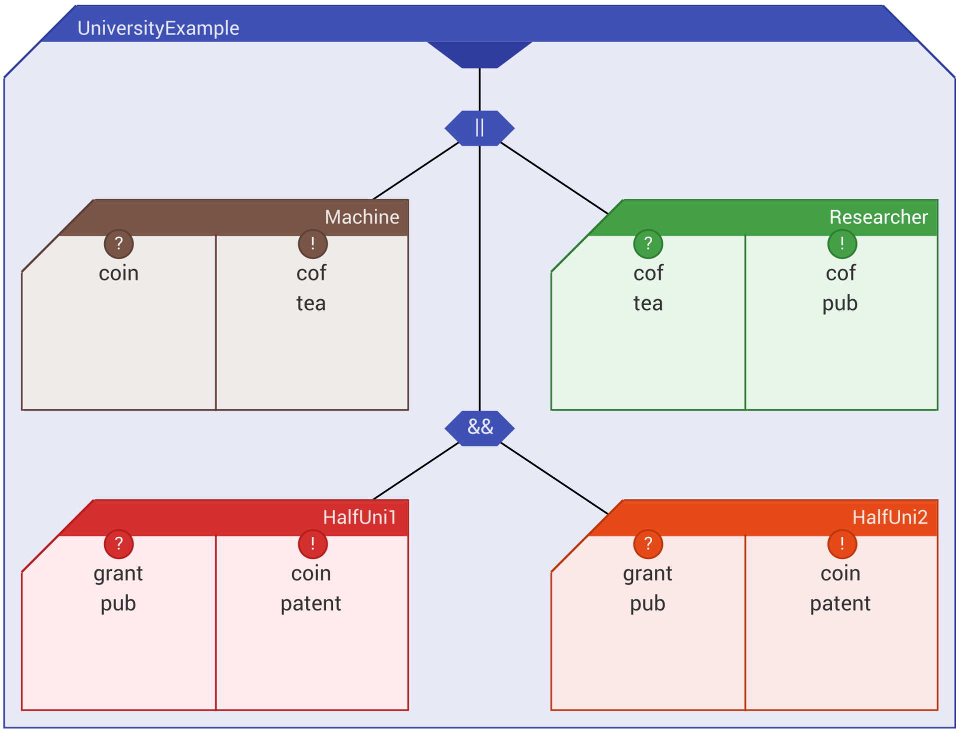
Ecdar 2.0

A New Integrated Modelling and Verification Environment for Compositional Real-Time Systems

Group Members:
Bartholomæussen, Casper Møller
Gundersen, Tobias Rosenkrantz
Lauritsen, Rene Mejer
Ovesen, Christian

Supervisor:
Nyman, Mathias Ulrik

Co-Supervisor:
Raptis, Dimitrios



12th of January 2018

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Ø
http://www.cs.aau.dk

Title:
A New Integrated Modelling and Verification Environment for Compositional Real-Time Systems: Ecdar 2.0

Subject:
Semantics and Verification

Project Period:
Fall Semester 2017

Project Group:
deis902e17

Participants:
Bartholomæussen, Casper Møller
Gundersen, Tobias Rosenkrantz
Lauritsen, Rene Mejer
Ovesen, Christian

Supervisor:
Ulrik Mathias Nyman

Co-Supervisor:
Dimitrios Raptis

Pages: 61

Date of Completion:
12th of January 2018

Abstract:
This report concerns the development of ECDAR 2.0, a model checking environment for compositional real-time systems based on the theory of timed input/output automata. We develop a new integrated modelling and verification environment for ECDAR in JavaFX to improve upon issues with the UI of ECDAR. We base ECDAR 2.0 on the codebase of the model checker H-UPPAAL.
We propose system views as a way of defining compositional systems using component instances and the operators conjunction, composition, and quotient. System views are designed to give users an overview of systems, generate system declarations, and be used in verification queries.
We develop features to assist in constructing valid models, let users export images, and increase productivity through shortcuts and better overview of models and systems. ECDAR 2.0 also features system views, which is currently useful for visualising systems.

Preface

This report is the product of a third semester project by four Software Engineering Master students at Aalborg University. The project this semester is a pre-specialisation project, which means that work on this project will continue on the fourth semester.

We want to give special thanks to our supervisors Ulrik Mathias Nyman and Dimitrios Raptis for helping us during the project period.

The reader should possess a basic knowledge of computer science and model checking to benefit the most from reading this report.

Reading Guide

All figures in the report are made by the authors unless stated otherwise in the figure caption. Lines or parts of source code that are unnecessary, are omitted and replaced with "`...`".

Citation Style

We use Harvard style for citations by listing authors and publication year. The bibliography can be found on page 50.

Source Code

A guide on how to access the source code can be found in Appendix B.

Contents

1 Introduction

The field of model checking has evolved throughout the years, from when Owicki and Gries (1976) made the (according to Clarke (2008)) best known formal system for conditional critical regions, at the time where proofs had to be made by hand, till today where Larsen et al. (2017) use model checking to generate test-cases in the area of Timed I/O Automata (TIOA). Model checking tools have evolved from when `Spin`[1] first came out in 1995. The same goes for the UPPAAL[2] tool, which had its initial release in 1995, and its latest update was July 1st, 2014.

Especially UPPAAL has resulted in the expansion of model checking tools, as UPPAAL has been branched to a wide range of tools namely: UPPAAL CORA, UPPAAL TIGA, ECDAR, UPPAAL TRON, UPPAAL STRATEGO, UPPAAL PORT, UPPAAL TIMES, and UPPAAL SMC. The tools in the UPPAAL family all have their different niche domain of application, but are all using similar Graphical User Interfaces (GUIs) with different underlying functionality.

We have especially found the ECDAR[3] tool to be interesting. ECDAR is used to model TIOA, and supports a wide range of operators such as refinement checking (compares the behaviour of two automata), conjunction (combines automata), composition (parallels automata), and quotient (removes behaviour from an automaton) (David et al. 2010). ECDAR also features compositional verification, which enables a divide-and-conquer approach to verification of a system. After trying the ECDAR tool, it seemed that TIOA does not fit in the UPPAAL GUI. The GUI does for instance not afford stepwise design, a visualisation of compositional verification, and also require the user to change the system declaration, if they have many different queries in their system. The tool also lacks features found in more modern Integrated Development Environments (IDEs) such as JetBrains IDEs[4]. In this project we develop a new front end for ECDAR in order to improve its GUI.

The latest version of ECDAR is 0.10. Therefore, we refer to the already existing ECDAR tool as ECDAR 0.10. With the new front end, we refer to the new application as ECDAR 2.0.

1.1 H-Uppaal

The preconditions for this project, is to build a new version of the ECDAR tool based on the codebase of H-UPPAAL[5]. H-UPPAAL is a different take on the UPPAAL GUI. It improves on some issues related to creating large models in UPPAAL, for example by dividing the automata model into hierarchical components (Mourtizsen and Jensen 2016; Mourtizsen

[1] http://spinroot.com/spin/whatispin.html

[2] http://uppaal.org

[3] http://ecdar.cs.aau.dk/

[4] https://www.jetbrains.com/

[5] We base our project on this node: https://github.com/ulriknyman/H-Uppaal/tr...

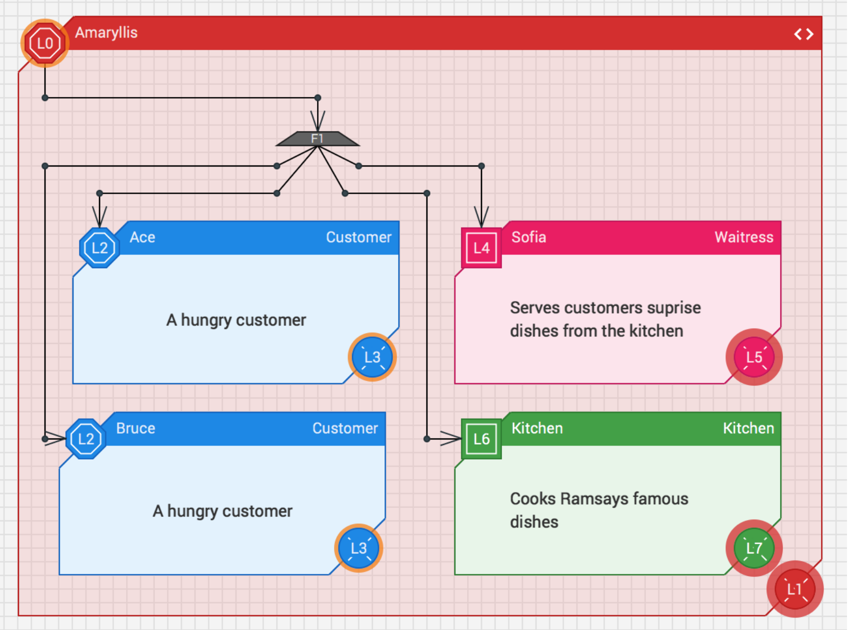


Figure 1.1: Screenshot of a hierarchical component in H-Uppaal (Mourtizsen and Jensen 2016).

and Jensen 2017). Our work is inspired by H-UPPAAL, and how they created an IDE for model checking.

The UPPAAL issues listed in Mourtizsen and Jensen (2016) are mostly related to the complexity and readability of large models. The following are examples: properties (e.g. invariants and clock resets) can be graphically placed away from their location or edge, locations do not have to be named (they are anonymous), and information can be hidden by overlapping objects. To combat these issues, they created H-UPPAAL, an IDE with continuous syntax checking, static code analysis, and background queries.

The representation of hierarchical timed automata in H-UPPAAL is presented in Figures 1.1 and 1.2. Figure 1.1 shows a hierarchical component, which contains subcomponents for `Customer`, `Waitress`, and `Kitchen`. Figure 1.2 shows the `Customer` component.

H-UPPAAL fixes many issues related to the GUI of UPPAAL, which also makes it an appropriate codebase to build ECDAR 2.0 upon, as ECDAR 0.10 also uses the GUI of UPPAAL.

1.2 PyEcdar

In this section we present a related tool, PYECDAR[6]. It is a Command-Line Interface (CLI) for analysis of TIOA. Contrary to ECDAR 0.10, it features robustness analysis and explicit

[6] https://project.inria.fr/pyecdar/

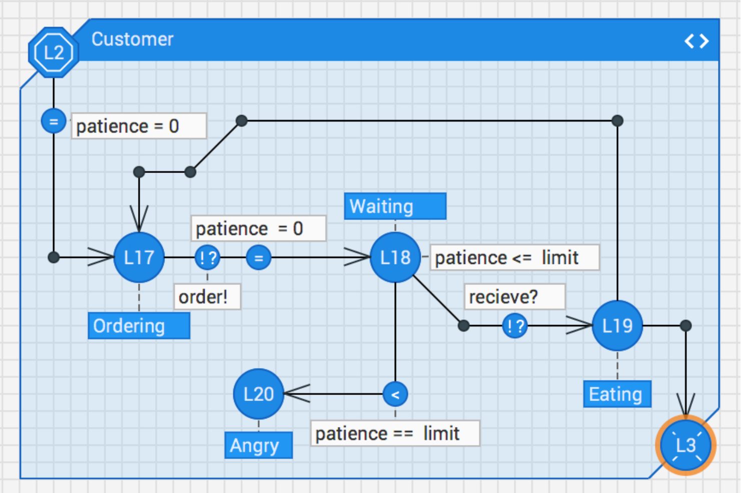


Figure 1.2: Screenshot of a component in H-Uppaal (Mourtizsen and Jensen 2016).

computation of conjunction, composition, and quotient. We present definitions for the applied specification theory in Chapter 2.

PYECDAR has no GUI or graphical simulator, and users cannot manually create or edit components. Instead, users must load the models from XML files (Legay et al. 2013), which for instance can be made in ECDAR 0.10.

Robustness describes to what degree an implementation can tolerate error in time. An implementation is robust with respect to a specification up to some bound δ, if it keeps refining the specification when its upper-bounded time constraints $x \leq a$ are replaced by $x \leq a + \delta$ and lower-bounded time constraints $x \geq b$ by $x \geq b - \delta$. PYECDAR can check for robustness for a specific δ and synthesise the maximum δ (Legay and Traonouez 2013).

PYECDAR can explicitly compute conjunction, composition, and quotient and save these as ECDAR 0.10 XML files. Thus, users can see a graphical representation of the higher-order components by using ECDAR 0.10 in order to get a better understanding of them.

2 Specification Theory

ECDAR 2.0 is based on ECDAR 0.10 which uses the specification theory presented by David et al. (2010). In this section we present the formal definitions for the specification theory. We present the definitions of TIOA, Timed I/O Transition Systems (TIOTSs), specifications, implementations, refinement, logical conjunction, structural composition, and quotient.

David et al. (2010) use a syntactical and a semantic representation of timed systems. They refer to the semantic representation as TIOTS. TIOTSs are infinite, and they use a syntactical and finite representation, to symbolically represent TIOTSs. They refer to this finite syntactical representation as TIOA. Throughout the section we will utilise the university example, which has been acquired from the ECDAR 0.10 university demo, to explain the concepts defined by David et al. (2010).

2.1 Timed I/O Automata

TIOA are defined by David et al. (2010) as the following:

A TIOA is a tuple $A = (Loc, q_0, Clk, E, Act, Inv)$ where

- Loc is a finite set of locations.
- $q_0 \in Loc$ is the initial location.
- Clk is a finite set of clocks.
- $E \subseteq Loc \times Act \times \mathcal{B}(Clk) \times \mathcal{P}(Clk) \times Loc$ is a set of edges.
- $Act = Act_i \oplus Act_o$ is a finite set of actions partitioned into inputs (Act_i) and outputs (Act_o).
- $Inv : Loc \mapsto \mathcal{B}(C)$ is a set of location invariants.

A TIOA has a set of clocks Clk which are used to represent time. Figure 2.1 shows an example of a TIOA, representing a university. E describes a set of edges. An edge has a start location, an action, a constraint $\mathcal{B}(Clk)$, a power set $\mathcal{P}(Clk)$, and an end location. $\mathcal{B}(Clk)$ from the definition represents a constraint over clocks. For edges, this constraint must be satisfied when executing the edge. Such constraints are called guards. On Figure 2.1 the edge from `Start` to `Grant` has the guard $u <= 2$.

$\mathcal{P}(Clk)$ is the power set of Clk and the set of clocks to reset. The previously mentioned edge on Figure 2.1 has the update `u = 0` that resets the clock u.

The previously mentioned edge is an input edge (solid arrow) over the *grant* channel. The edge from `Grant` to `Start` is an output edge (dashed arrow) over the *patent* channel.

Invariants define a constraint over clocks, $\mathcal{B}(C)$, on locations. Unlike guards, these constraints are on a location instead of an edge and must be satisfied when entering and while in the location. The location `Grant` of Figure 2.1 has the invariant $u <= 20$.

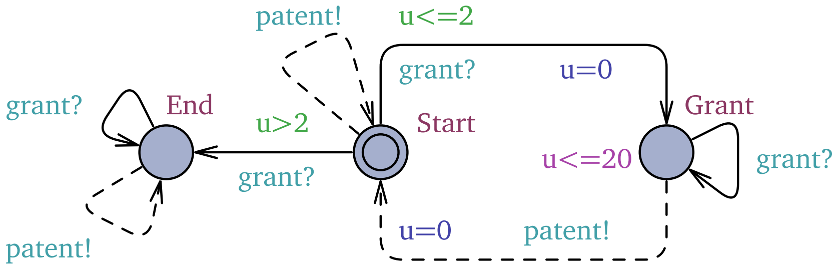


Figure 2.1: `Spec` component of the university example (David et al. 2017) presented throughout the report.

2.2 Timed I/O Transition Systems

TIOTSs are the semantic representations induced by TIOA. TIOTSs are defined by David et al. (2010) as:

A TIOTS is a tuple $S = (St^S, s_0, \Sigma^S, \rightarrow^S)$ where

- St^S is an infinite set of states.
- $s_0 \in St^S$ is the initial state.
- $\Sigma^S = \Sigma_i^S \oplus \Sigma_o^S$ is a finite set of actions partitioned into inputs (Σ_i^S) and outputs (Σ_o^S).
- $\rightarrow^S : St^S \times (\Sigma^S \cup \mathbb{R}_{\geq 0}) \times St^S$ is a transition relation.

We write $s \xrightarrow{a}^S s'$ instead of $(s, a, s') \in \rightarrow^S$ and use $i?$, $o!$, and d to range over inputs, outputs, and delays ($\mathbb{R}_{\geq 0}$), respectively. Any TIOTS satisfies the following:

- Time *determinism*:

  Whenever $s \xrightarrow{d}^S s'$ and $s \xrightarrow{d}^S s''$ then $s' = s''$.

- Time *reflexitivity*:

  $s \xrightarrow{0}^S s$ for all $s \in St^S$.

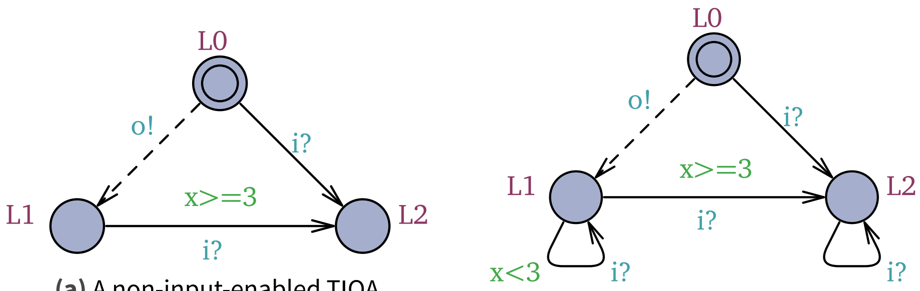
- Time *additivity*:

  For all $s, s'' \in St^S$ and all $d_1, d_2 \in \mathbb{R}_{\geq 0}$ we have $s \xrightarrow{d_1+d_2}^S s''$ iff $s \xrightarrow{d_1}^S s'$ and $s' \xrightarrow{d_2}^S s''$ for an $s' \in St^S$.

Time determinism means that you can transition to exactly one state through a given delay. Time reflexitivity means that you are always able to delay 0 time units, and you will end up in the same state. Time additivity means that if and only if you perform a delay d_1 and then perform another delay d_2, you end up in the same state as if you performed a delay $d_1 + d_2$.

2.3 Specifications and Input-Enableness

A TIOTS S is *input-enabled*, iff $\forall s \in St^S . \forall i? \in \Sigma_i^S . s \xrightarrow{i?}^S$. That is, it can always accept any of its defined inputs. A TIOTS is a specification if it is input-enabled. A *specification*



(a) A non-input-enabled TIOA.

(b) An angelic completion of (a).

Figure 2.2: A non-input-enabled TIOA and its angelic completion.

automaton is a TIOA which induces a specification. An example of a specification automaton can be seen in Figure 2.1. This specification is always able to take the input *grant?*, its only input, and is therefore input-enabled.

You can transform an automaton to an input-enabled one, with *angelic completion*. It creates self-loops for undefined inputs in each location. It essential corresponds to ignoring those inputs. Figure 2.2 illustrates an example of angelic completion.

2.4 Implementations

A subset of TIOTSs are implementations. An implementation must satisfy output urgency and independent progress as defined below.

An *implementation* $P = (St^P, p_0, \Sigma^P, \rightarrow^P)$ is a specification such that for each state $p \in St^P$:

- *output urgency*:

$$\forall p', p'' \in St^P \text{ if } p \xrightarrow{o!}^P p' \text{ and } p \xrightarrow{d}^P p'' \text{ then } d = 0$$

- *independent progress*:

$$\text{either } (\forall d \geq 0 . p \xrightarrow{d}^P)$$
$$\text{or } \exists d \in \mathbb{R}_{\geq 0} . \exists o! \in \Sigma_o^P p \xrightarrow{d} p' \text{ and } p' \xrightarrow{o!}^P \text{ for a } p' \in St^P.$$

Output urgency means that if a TIOTS is able to do an output, it must do so before time is allowed to pass. The independent progress rules state, that a TIOTS must always be able to delay continuously or delay until it can output. This ensures a transition system can never block the progress of time.

2.5 Features

In this section we define the features refinement (compares the behaviour of two automata), conjunction (combines automata), composition (parallels automata), and quotient (removes behaviour from an automata).

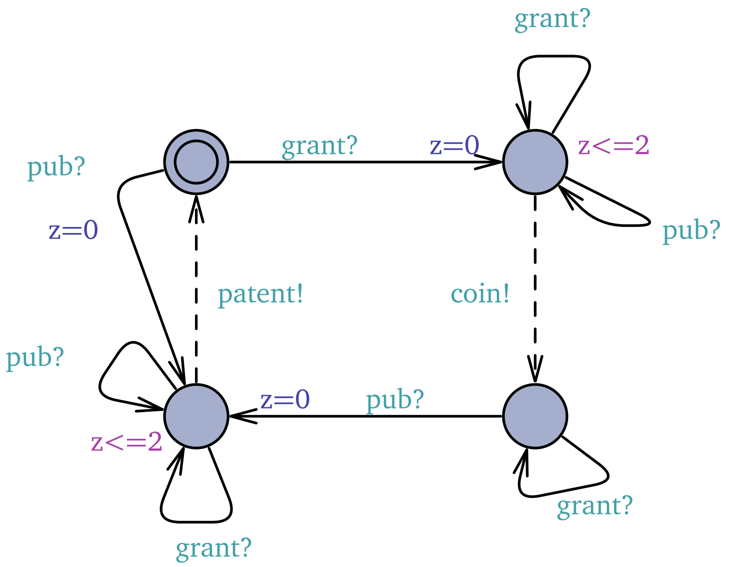


Figure 2.3: The `University` specification from the university example (David et al. 2017).

Refinement and Consistency: Refinement, as defined by David et al. (2010), allows comparison between two specifications, by showing which specification allows more or same *behaviour*.

A specification $S = (St^S, s_0, \Sigma, \rightarrow^S)$ *refines* another specification $T = (St^T, t_0, \Sigma, \rightarrow^T)$, written $S \leq T$, iff there exists a binary relation $R \subseteq St^S \times St^T$ containing (s_0, t_0) such that for each pair of states $(s, t) \in R$ we have:

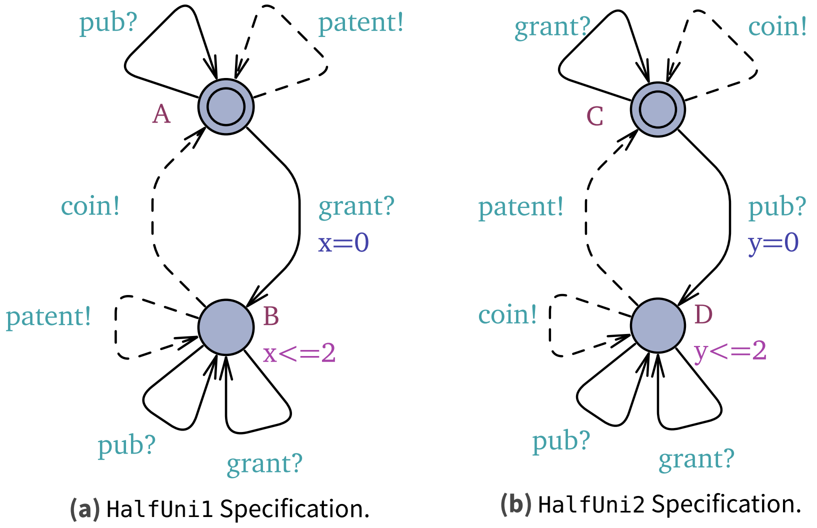
- whenever $t \xrightarrow{i?}^T t'$ for some $t' \in St^T$ then $s \xrightarrow{i?}^S s'$ and $(s', t') \in R$ for some $s' \in St^S$.
- whenever $s \xrightarrow{o!}^S s'$ for some $s' \in St^S$ then $t \xrightarrow{o!}^T t'$ and $(s', t') \in R$ for some $t' \in St^T$.
- whenever $s \xrightarrow{d}^S s'$ for some $d \in \mathbb{R}_{\geq 0}$ then $t \xrightarrow{d}^T t'$ and $(s', t') \in R$ for some $t' \in St^T$.

$S \leq T$ means that S has less behaviour than or equal behaviour to T. A specification automaton A_1 refines another specification automaton A_2, written $A_1 \leq A_2$, iff $[\![A_1]\!]_{sem} \leq [\![A_2]\!]_{sem}$. That is, there is a corresponding refinement between their transition systems.

By the definition of David et al. (2010) a specification is *consistent*, if an implementation refines it. A consistent specification can have states that if entered would disallow progress of time and output transitions. These states are referred to as *inconsistent states*. An implementation cannot have inconsistent states.

As opposed to inconsistent states, a specification can also include a *universal state*. In a universal state every possible behaviour defined by the transition system is enabled. That is, it can continuously both delay indefinitely and perform every action Σ^S in its transition system S.

Logical Conjunction: Conjunction between a specification S and specification T is written $S \wedge T$. The definition of conjunction uses S^Δ to symbolise the transformation, where S is pruned of all inconsistent states.



(a) `HalfUni1` Specification. **(b)** `HalfUni2` Specification.

Figure 2.4: Shows two different specifications that are modified versions taken from the university example (David et al. 2017). If they are conjoined together they should refine the `University` component shown in Figure 2.3

.

Conjunction is defined by David et al. (2010) as $S \wedge T = (S \times T)^{\Delta}$ for specifications S and T over the same actions *Act* where $S \times T$ is consistent.

We use Figures 2.3 and 2.4 to give an example of conjunction. We attempt to divide the `University` component into two components `HalfUni1` and `HalfUni2` (see Figure 2.4). We then check if the conjunction of the two is a specification with the query `specification: HalfUni1 && HalfUni2`.

We define components not made by conjunction, composition, or quotient as *first-order* components, and *high-order* for the components that are a result of those operations.

Structural Composition: Composition between two specifications makes them run in parallel. According to David et al. (2010), composition can only be used on two compatible specifications. Compatible specifications are specifications where if they are composed, it is possible to avoid inconsistent states. We write the composition between specifications S and T as $S|T$.

As an example we use the university example. We have introduced the `University` component in Figure 2.3 and the `Spec` component for the whole university in Figure 2.1. `University` does not refine `Spec` as it needs the behaviour of the `Researcher` and `Machine` component. The `Machine` and `Researcher` components can be seen in Figures 2.5 and 2.6, respectively. Now that we have defined all components needed to refine the specification `Spec`, we can compose these and then do a refinement check. `refinement: (Machine || University || Researcher) <= Spec`. This query is satisfied. We can also try and substitute the `University` component with the conjunction of `HalfUni1` and `HalfUni2` mentioned earlier. `refinement: (Machine || (HalfUni1 && HalfUni2) ||`

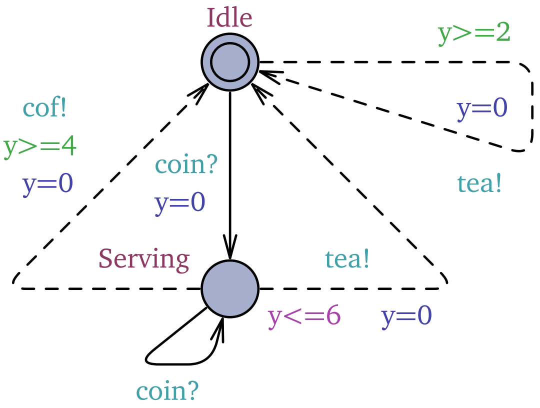


Figure 2.5: Shows the `Machine` specification

`Researcher) <= Spec`. This query is also satisfied.

Quotient: David et al. (2010) defines the quotient operator. It has two operands, a large specification T and a small specification S. Where T includes the behaviour of S. It determines what behaviour the smaller specification needs, in order to refine the larger specification. Quotienting is denoted as $T \backslash\backslash S$. The resulting specification is the specification x with most behaviour that satisfies the following query: $S||x \leq T$.

The quotient operator can be used in cases where you have a specification that partially refines another specification but it is missing some behaviour. If you need this missing behaviour for other queries, it can then be obtained with the quotient operator. For example, we have the `University` component from earlier and need it to refine the `Spec` component. The quotient $Spec\backslash\backslash University$ is the specification with most behaviour which composed with `University` refines `Spec`. The `Machine` and `Researcher` components refine `Spec` if composed together with `University`. This can be checked with the following query in ECDAR 0.10 using the quotient operator (the operator in ECDAR 0.10 is a single \): `refinement: (Machine || Researcher) <= (Spec\University)`. This query holds.

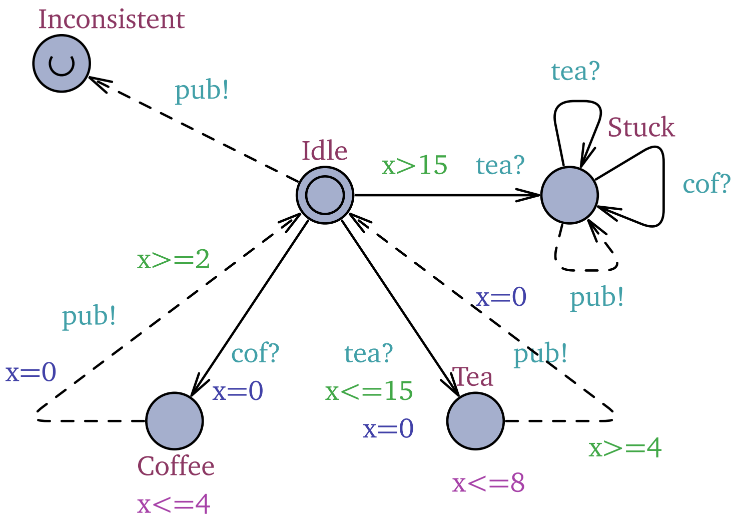


Figure 2.6: Shows the `Researcher` Specification from the university example (David et al. 2017).

3 Analysis

To get a more in-depth picture of what ECDAR 2.0 originate from, we will first present ECDAR 0.10 and how it can be used as a tool, for then to discuss the issues that we have found. To have a better understanding of the usage of ECDAR 0.10, we have conducted an interview. We present it and discuss the results of it.

3.1 Ecdar 0.10

ECDAR 0.10 uses the specification theory for compositional verification of real-time systems using TIOA, presented in Chapter 2. The tool provides an environment, where the user can design real-time systems using TIOA and verify properties of the system using the constructs of the theory: refinement, consistency, conjunction, composition, and quotient. ECDAR 0.10 uses the timed game engine of UPPAAL TIGA (Behrmann et al. 2007) to solve the various games involved in checking for refinement and computing composition. The automata presented in Chapter 2 are constructed in ECDAR 0.10.

3.1.1 Usage

As previously mentioned, ECDAR 0.10 uses the UPPAAL TIGA engine. In fact, ECDAR 0.10 is included in the current version (0.17) of UPPAAL TIGA. The standalone version (currently 0.10) of ECDAR is also available for download.

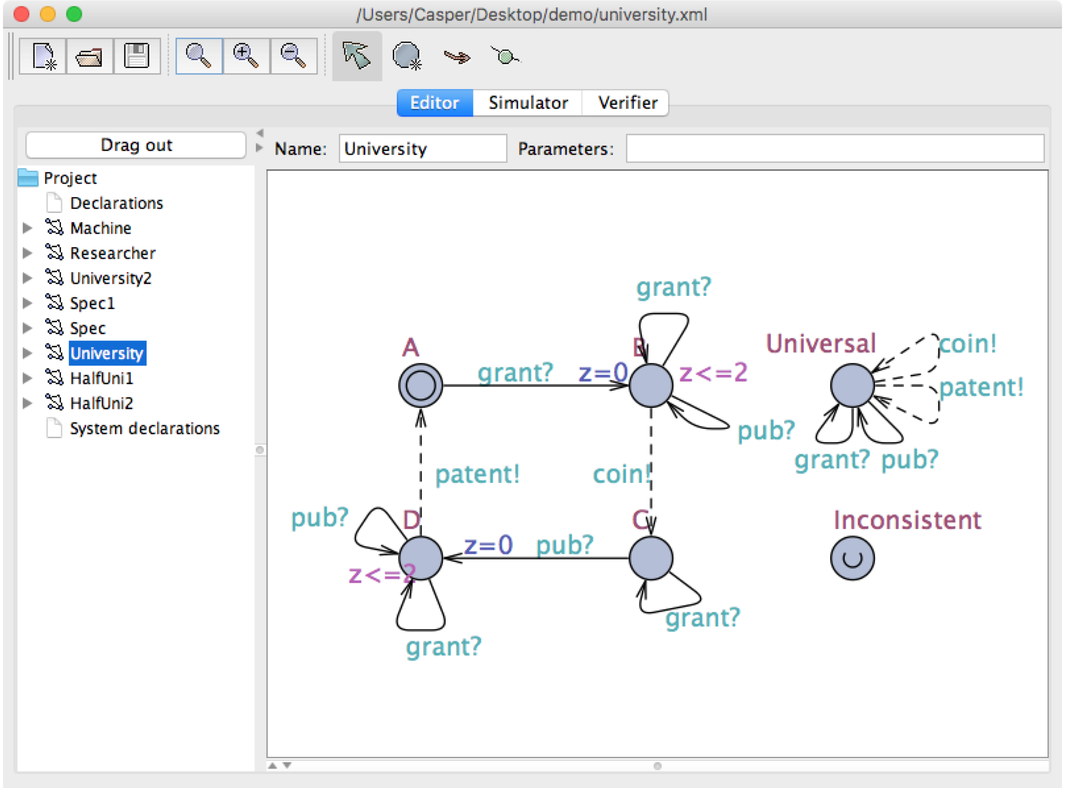
In this section, we give a general description of ECDAR 0.10 and how it can be used in designing compositional systems and verifying properties for them.

Figure 3.1 shows the editor of ECDAR 0.10. The editor is used for defining templates, both visually as a TIOA, but also textually for the accompanying declarations, like clocks. Apart from the TIOA and their local declarations, it is also possible to define global and system declarations. In the global declarations, users can define the communication channels. Only broadcast channels are valid. The system declarations declare which templates to instantiate and which inputs and outputs each of them use.

The verifier tab (seen in Figure 3.2) is essential to ECDAR 0.10. Here, a user can query the system declared in the system declarations. A query can, for instance, check for consistency, whether a component is a specification, or if a component refines another.

A simple example can also be seen in Figure 3.2: `specification: (University || Machine || Researcher)`. The query asks whether a composition of the `University`, `Machine`, and `Researcher` components is a specification.

The last main tab of ECDAR 0.10 is the simulator tab (seen in Figure 3.3). The simulator provides a view of the system components interacting with each other. A user can choose which transitions to take, or just let ECDAR 0.10 make random choices. It can also be used to visualise the strategy that was found for some query in the verifier. The visualisation could be helpful, when a property does not hold, as the user can see the strategy.



Name: University

Parameters:

Figure 3.1: The editor of Ecdar 0.10.

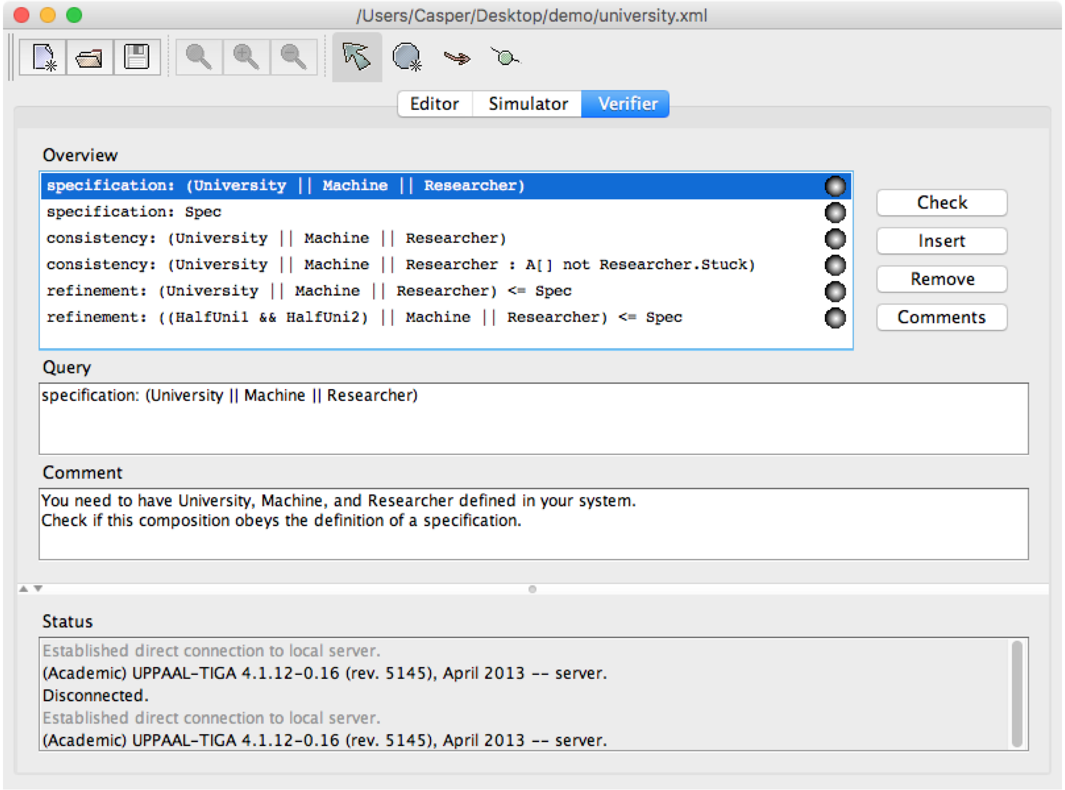
3.2 Ecdar Issues

As mentioned in Section 3.1, ECDAR 0.10 is a part of UPPAAL TIGA, which itself reuses most of the UPPAAL user interface. For UPPAAL TIGA it might be fitting to inherit the User Interface (UI) of UPPAAL. However, we will find that this process can lead to invalid and redundant design when done with ECDAR 0.10. As described in Section 1.1, Mourtizsen and Jensen (2016) account for some of the issues that they have located in UPPAAL, most of which are related to the complexity and readability of large models. They have designed H-UPPAAL to counter some of these issues. By reusing and modifying their codebase to fit the ECDAR domain, we can benefit from their solutions.

On the download page of the ECDAR 0.10 webpage[1], there are a number of known issues. Some of these issues we have encountered ourselves and are further described in this section. An example of these known issues is concerned with input-enabledness: users must explicitly make every location input-enabled to perform a refinement check.

This section only outlines some of the issues we have encountered in ECDAR 0.10.

[1] http://people.cs.aau.dk/~adavid/ecdar/download.html



Overview

```
specification: (University || Machine || Researcher)
specification: Spec
consistency: (University || Machine || Researcher)
consistency: (University || Machine || Researcher : A[] not Researcher.Stuck)
refinement: (University || Machine || Researcher) <= Spec
refinement: ((HalfUni1 && HalfUni2) || Machine || Researcher) <= Spec
```

Query

specification: (University || Machine || Researcher)

Comment

You need to have University, Machine, and Researcher defined in your system.
Check if this composition obeys the definition of a specification.

Status

```
Established direct connection to local server.
(Academic) UPPAAL-TIGA 4.1.12-0.16 (rev. 5145), April 2013 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL-TIGA 4.1.12-0.16 (rev. 5145), April 2013 -- server.
```

Figure 3.2: The verifier of Ecdar 0.10.

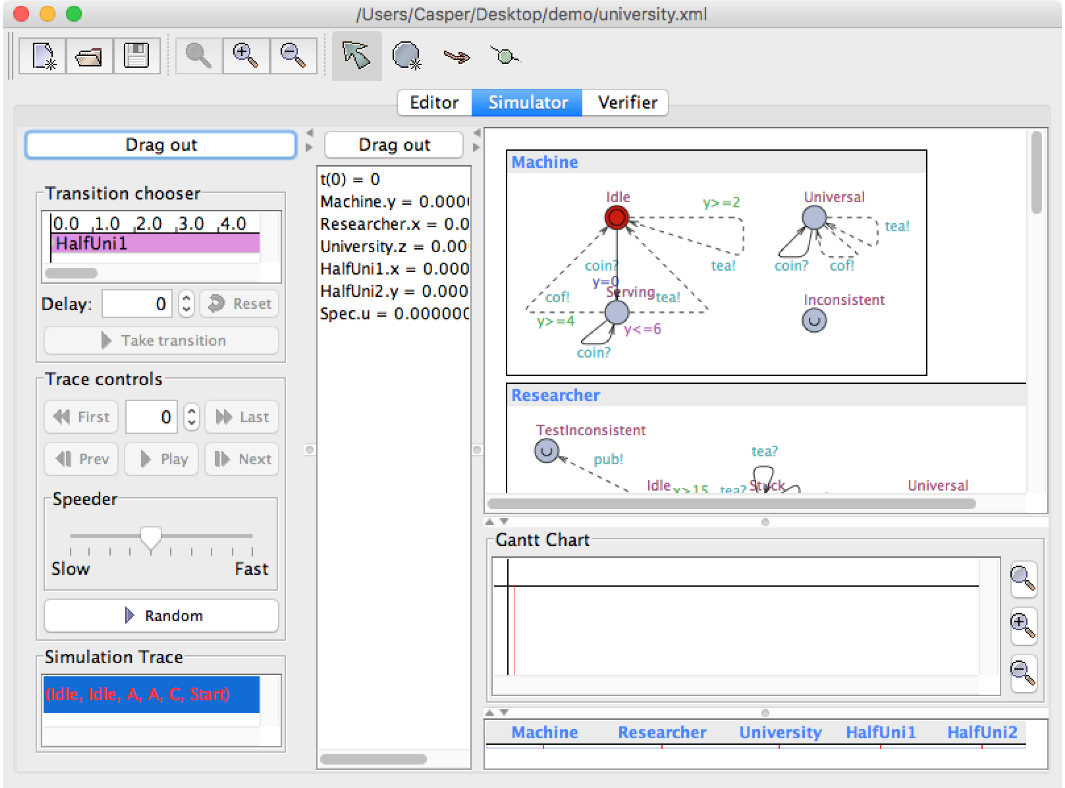
System Declarations In the system declarations file, the user instantiates the components to be composed into a system and declares which inputs and outputs the components use. Figure 3.4 shows an excerpt of the system declarations file in the *Milner-8Nodes* example project that is bundled with ECDAR 0.10.

Each component instance and their possible inputs and outputs must be declared in the system declarations. This can result in a significant amount of boilerplate code. Writing such code is repetitive, redundant and error-prone. The inputs and outputs that a user must declare in this file, could as well have been automatically detected in the models.

In a stepwise design, a user might want to use different systems (or variants of systems) in different queries. However, in ECDAR 0.10 each project has exactly one system declarations files that can only declare one system. We see two workarounds:

- A user can use a larger system containing all component instances used in all queries. This way the engine would only explore the instances needed for a specific query. Thus, the state space in unaffected. However, the engine keeps all declared instances in memory, increasing memory usage.

- A user can declare multiple systems in the system declarations, but comment out the declarations of the system not used by a specific query. This way, the



```
t(0) = 0
Machine.y = 0.000
Researcher.x = 0.0
University.z = 0.00
HalfUni1.x = 0.000
HalfUni2.y = 0.000
Spec.u = 0.000000
```

Simulation Trace: (Idle, Idle, A, A, C, Start)

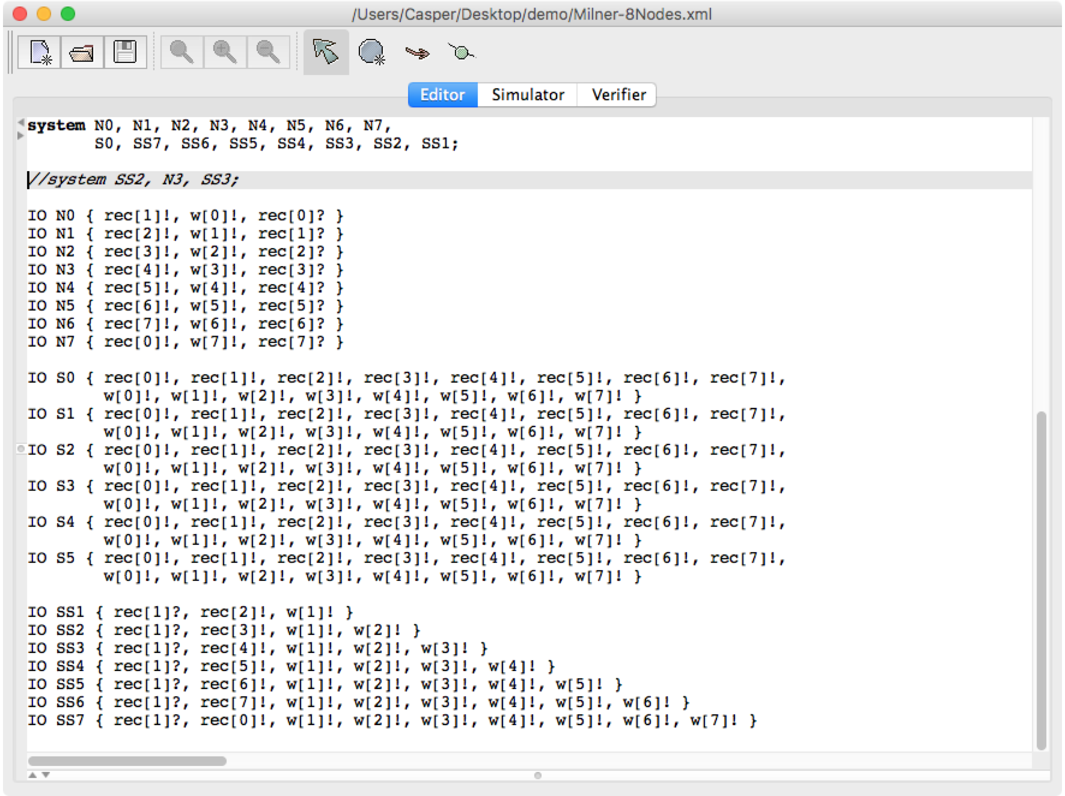
Figure 3.3: The simulator of Ecdar 0.10.

engine only allocates memory for the necessary instances. However, the user has to comment/uncomment system declarations depending on which queries they want to run.

Universal and Inconsistent Locations To use the quotient operator in queries the user must add the `Universal` and `Inconsistent` locations to each of the components used in the queried system. The former must accept all inputs and generate all outputs of that process. The latter is an urgent location without any transitions. An example of these locations can be seen in Figure 3.5. This is a repetitive task, where users must construct almost identical locations multiple times. Users must also maintain the universal location whenever new inputs or outputs are added to the component.

Invalid TIOA Models The editor in Ecdar 0.10 does not warn or hinder the user in creating invalid TIOA models. There are multiple ways of creating an invalid model:

- In Ecdar 0.10 it is possible to construct committed locations. However, the theory of David et al. (2010) does not define the behaviour for this concept. In Ecdar 0.10 a committed location can block the progress of the whole system. Such behaviour



```
system N0, N1, N2, N3, N4, N5, N6, N7,
       S0, SS7, SS6, SS5, SS4, SS3, SS2, SS1;

//system SS2, N3, SS3;

IO N0 { rec[1]!, w[0]!, rec[0]? }
IO N1 { rec[2]!, w[1]!, rec[1]? }
IO N2 { rec[3]!, w[2]!, rec[2]? }
IO N3 { rec[4]!, w[3]!, rec[3]? }
IO N4 { rec[5]!, w[4]!, rec[4]? }
IO N5 { rec[6]!, w[5]!, rec[5]? }
IO N6 { rec[7]!, w[6]!, rec[6]? }
IO N7 { rec[0]!, w[7]!, rec[7]? }

IO S0 { rec[0]!, rec[1]!, rec[2]!, rec[3]!, rec[4]!, rec[5]!, rec[6]!, rec[7]!,
        w[0]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]!, w[7]! }
IO S1 { rec[0]!, rec[1]!, rec[2]!, rec[3]!, rec[4]!, rec[5]!, rec[6]!, rec[7]!,
        w[0]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]!, w[7]! }
IO S2 { rec[0]!, rec[1]!, rec[2]!, rec[3]!, rec[4]!, rec[5]!, rec[6]!, rec[7]!,
        w[0]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]!, w[7]! }
IO S3 { rec[0]!, rec[1]!, rec[2]!, rec[3]!, rec[4]!, rec[5]!, rec[6]!, rec[7]!,
        w[0]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]!, w[7]! }
IO S4 { rec[0]!, rec[1]!, rec[2]!, rec[3]!, rec[4]!, rec[5]!, rec[6]!, rec[7]!,
        w[0]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]!, w[7]! }
IO S5 { rec[0]!, rec[1]!, rec[2]!, rec[3]!, rec[4]!, rec[5]!, rec[6]!, rec[7]!,
        w[0]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]!, w[7]! }

IO SS1 { rec[1]?, rec[2]!, w[1]! }
IO SS2 { rec[1]?, rec[3]!, w[1]!, w[2]! }
IO SS3 { rec[1]?, rec[4]!, w[1]!, w[2]!, w[3]! }
IO SS4 { rec[1]?, rec[5]!, w[1]!, w[2]!, w[3]!, w[4]! }
IO SS5 { rec[1]?, rec[6]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]! }
IO SS6 { rec[1]?, rec[7]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]! }
IO SS7 { rec[1]?, rec[0]!, w[1]!, w[2]!, w[3]!, w[4]!, w[5]!, w[6]!, w[7]! }
```

Figure 3.4: Extract of a system declarations file in Ecdar 0.10.

should not be possible when using TIOA, as each component should be able to progress independent of the other components.

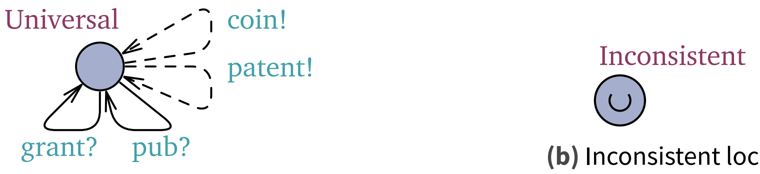
- Users denote whether an edge is an input or output edge (we define this as the *I/O status* of the edge) with a checkbox. They denote whether an edge should synchronise with inputs or outputs with the `?` (for inputs) or `!` (for outputs) suffixes as in UPPAAL. Users can, through combinations of these, create inputs on output edges and vice versa.

- Global variables can be declared and written to by a component.

- Tau transitions are invalid, but can be added to a component.

- Non-broadcast channels can be declared and used in components.

While the editor and syntax checker does not catch these issues, the verifier gives a warning when one of the issues is detected.



(a) Universal location.

(b) Inconsistent location.

Figure 3.5: Universal and inconsistent locations in Ecdar 0.10.

3.3 Interviews

In order to gain more insight into the users of Ecdar 0.10, we decided to interview some users. The purpose is to get new ideas for features we could make in Ecdar 2.0 based on user needs and how they are working in Ecdar 0.10.

We identified a group of Ecdar 0.10 users, who are mostly based in our department at Aalborg University. Of the users, present at Aalborg University, were a postdoc, and our supervisor. The interviewees are closely related to each other and the development of Ecdar 0.10, meaning that they might be biased. Our supervisor might especially be biased, as he is one of the creators of the theory behind Ecdar (David et al. 2010). However, since the number of users of Ecdar 0.10 is very limited, we still want to hear their inputs and how they are working with Ecdar 0.10, keeping the bias in mind.

The interviewees are both familiar with Uppaal. Our supervisor is one of the creators of the theory behind Ecdar and therefore has in-depth knowledge of the Ecdar 0.10 tool. The postdoc has used Ecdar 0.10 for about four months for one project.

The interviews were conducted in a semi-structured (Benyon 2013) manner. This made the interview into more of a dialogue, where we could ask follow-up questions when needed. The planned questions can be seen in Appendix A. Two group members conducted the interviews, one to ask questions and the other to take notes. To support the documentation, the interview was also audio recorded. Appendix B describes how to access the audio files of the interviews.

In case that one of the interviewees wanted to show us something in Ecdar 0.10, we had a computer with an example project, which they could use to point at and explain. During the interviews, we recorded the screen of the computer, however, the interviewees did barely use it.

3.3.1 Results

To outline the results of the interviews, we have structured this section in the same manner as the main topics of the interview found in Appendix A. Furthermore, we have added paragraphs for system declarations and collaboration, as the interviews also covered these. We have also added a paragraph about wishes and suggestions, to describe the interviewees' ideas.

Design Process: When the interviewees start a new ECDAR 0.10 project, they do so in different ways: our supervisor starts out by making a model using pen and paper, and hereafter uses ECDAR 0.10 to make an overall specification, which he then refines. The postdoc starts right away in ECDAR 0.10 by making the intended models and the system declarations.

They both agree, if the number of locations in a component exceeds 15, the model should be refactored into multiple components. An ECDAR 0.10 project normally consists of one to five components, but in an example shown by the postdoc, he had 20 components.

Besides that, their design processes also include refactoring the models so they can be used in scientific papers.

System Declarations: The postdoc only needed to declare one system in the system declarations of his project. For one of the examples found in ECDAR 0.10, which our supervisor helped model, the system declarations is so complex that our he decided to make a small Bash script that wrote it for him, as writing it by hand would be too error-prone and would take a long time.

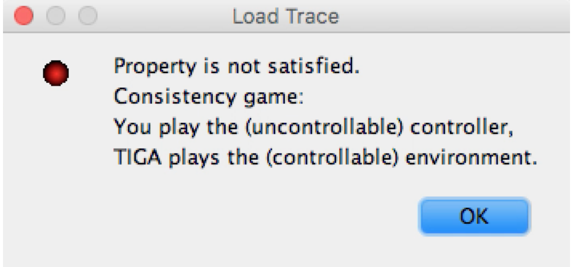
Verifier: According to the interviewees, refinement checking is the most important query used in the verifier, hereafter comes consistency checking and composition. When our supervisor is making models, he makes intermediate queries i.e. queries not included in the final model, but the postdoc did not used intermediate queries. Both of the interviewees agree that it is ideal to use intermediate queries, since they can be used to discover problems during the modelling process.

Simulator: Both interviewees agree that the simulator of ECDAR 0.10 contains significant flaws. One of the flaws is the lack of overview, e.g. where the transition chooser is trying to guide the simulation.

Debugging a model is tedious in ECDAR 0.10 according to our supervisor. He states: "... to understand what is wrong in your model, (you) have to sit and think about it for five minutes, then go and fix it, as it (ed. the simulator) does not bring you intuitively fast to where you really want." For instance, you have to transition to the end of a strategy in order to see why a refinement does not hold.

The postdoc uses the simulator to trigger the syntax checker, which makes the simulator important for his workflow. It also shows that the intended way to trigger the syntax checker is not good enough for him.

Collaboration: The interviewees both collaborate with others using the following process: (1) They attend a meeting where they propose and accept ideas for models. (2) One of the attendees constructs the models. (3) They hold another meeting, where they raise and approve suggestions for changes to the models. (4) They continue with 1 and 2 until they agree on the models.



Load Trace

Property is not satisfied.
Consistency game:
You play the (uncontrollable) controller,
TIGA plays the (controllable) environment.

Figure 3.6: Error message in Ecdar 0.10.

Pros and Cons: Both interviewees believe compositional verification and refinement checking are strengths of the theory of TIOA and work well together.

The main drawbacks of ECDAR 0.10 are the problems related to system declarations and the simulator; the simulator has quite some flaws, which make it unfit for use, especially for larger models. Other drawbacks include that you manually have to make components input-enabled, and poor error messages. One error message that was brought up doing the interview, is shown in Figure 3.6. In this example, ECDAR 0.10 runs a refinement check. The error message does not make sense, as there are not any controller or environment for this check. Another issue with it is that the game for refinement check is different from the UPPAAL TIGA games, but the simulator still shows the game as a UPPAAL TIGA game.

Wishes and Suggestions: In this paragraph we outline ideas that the interviewees suggested during the interviews.

- To assist researchers with collaboration on models, our supervisor suggested that the tool could integrate support for *Version Control Systems (VCSs)* and feature a special *diff tool* for models. For instance, ECDAR 2.0 could query older versions of a model and compare the results with current version.
- Both interviewees suggested a new way to *animate a transition in the simulator*, in order to make it aligned with modern UI design. When ECDAR 0.10 displays a transition, it highlights the chosen edge by making it red. Instead, ECDAR 2.0 could displays an animation that shows the transition.
- For now, it is strenuous when you need to comment and out-comment the systems in the system declarations, according to which query you want to run in the verifier. That is why our supervisor suggested making it possible to make *systems in a GUI*. This could be a big improvement to user experience, as it can remove complexity and make it possible for users to maintain an overview.
- As the area of mutation-based testing is promising, it could be interesting to make a *mutation tool* for ECDAR 2.0. The postdoc suggested the mutation tool, as he had used ECDAR 0.10 for that purpose. Such a mutation tool could also include a test-case generator, as described by Larsen et al. (2017).

3.3.2 Summary

By interviewing users of ECDAR 0.10 we have now gained knowledge about how they are working with the tool, and their suggestions for improvements. Especially the suggestion about systems in a GUI had caught our attention. In Section 8.1 we discuss source of error by only having the interviewees that we have.

4 Scope

The inputs and outputs of TIOA introduce another way to conceptualise systems compared to regular timed automata. Yet, ECDAR 0.10 uses a GUI designed for UPPAAL, which causes usability problems in ECDAR 0.10 as mentioned in Sections 3.2 and 3.3.

We intend to improve the GUI for ECDAR in order to improve productivity and usability of compositional analysis and design of real-time systems. To do this, we propose a new tool, ECDAR 2.0, that uses the ECDAR engine. The new tool should use concepts from modern IDEs. We also propose system views to help give an overview of systems and ease the declaration and use of systems in queries.

4.1 Integrated Modelling and Verification Environment

ECDAR 0.10 users have to do some things manually according to Section 3.2. IDEs provide facilities to assist users in developing software. To use the same concepts in modelling and verification, we introduce the term *Integrated Modelling and Verification Environment (IMVE)*. It is a model checking tool with facilities to assist in modelling and verification. To improve productivity, we intend to develop ECDAR 2.0 as an IMVE. We will do this with respect to the following areas.

Project Management: Users of an IMVE should be able to manage projects by creating a new empty project, opening a project, saving an existing project, and saving an existing project as a new one (this feature is often called **save as**). ECDAR 0.10 features these. However, H-UPPAAL does not feature options to create a new project and to save a project as a new one.

Export Models: To present systems and components in, for instance, documents, an IMVE could feature the possibility to export the parts as an image. This feature is a part of ECDAR 0.10, but is not found in H-UPPAAL.

Enforcement of Good Practices: A IMVE should enforce good practices. For instance, every component must have an *initial location* in order to be valid. The UI of ECDAR 0.10 does not enforce this; its users can delete the initial location in a component, and newly created components do not have any locations. ECDAR 2.0 should, however, enforce this.

Tau transitions: are prohibited in a TIOA, but can be added to components in ECDAR 0.10 (see Section 3.2). ECDAR 2.0 should prevent this from happening.

ECDAR 0.10 users can construct invalid models by defining *inputs on output edges or vice versa* according to Section 3.2. ECDAR 2.0 should prevent this from happening.

Inconsistent and Universal Locations: According to Section 3.2, certain queries require an `Inconsistent` and a `Universal` location to be present in ECDAR 0.10. This requirement leads to the repetitive action of adding these locations to certain components. This is even the case, when the locations have no incoming edges from other locations. In ECDAR 2.0 we want to handle this problem by adding these locations automatically, when translating models to the ECDAR XML for the back end. Furthermore, ECDAR 2.0 should feature a quicker way to add these locations to components visually.

Interaction Between Components: From our experience with H-UPPAAL, we sometimes struggle with getting an overview of components, and understanding which components might interact. Thus, ECDAR 2.0 should give information about interaction of components.

Note that this feature shares some similarities with the *Component Communication* suggestion described in the *Future Work* of Mourtizsen and Jensen (2017). Their suggestion is to show information on the sides of a component, as a way of describing how the component might interact with other components.

Keyboard Shortcuts: Shortcuts can reduce the time to execute commands. Furthermore, when designing shortcuts with conventions in mind, they can reduce the time used looking for a specific command. An example of this is the possibility to open a project with `Ctrl + O` on Windows and Linux and `Cmd + O` on macOS. To improve productivity ECDAR 2.0 should have keyboard shortcuts.

4.2 System Views

Systems can consist of components related with conjunctions, compositions, and quotients. With stepwise design, users work with several variations of a system. Multiple variations can use some of the same components.

To graphically represent systems and to give users an overview of them, we introduce the concept of *system views*. A system view could include instances of first-order components, their relations, and a unique system name. It could also include other systems to enable construction of systems in a hierarchical manner.

When querying systems with several first-order components, the ECDAR 0.10 queries become longer, as they each need to include names of all the components. To create an overview of the queries, improve productivity, and provide information hiding when writing queries, users can use system names. In this way, a system name work as an *alias* for the components of the system and relations of the components.

In the system declarations of ECDAR 0.10, users must, according to Section 3.2, instantiate components and declare which inputs and outputs they use. Furthermore, users must comment and uncomment system declarations whenever they want to query other systems. With system views, ECDAR 2.0 could *generate system declarations* for each query in the following way. Based on the component instances in the system queried, ECDAR 2.0 could instantiate components. By statically analysing the first-order components in the query, ECDAR 2.0 could find which inputs and outputs they use. ECDAR 2.0 could then declare that in the system declarations.

5 Design

In this chapter we discuss design choices of ECDAR 2.0. First, we discuss design principles that we follow throughout the development of ECDAR 2.0. Then we discuss changes to H-UPPAAL required for the application to conform to ECDAR 2.0. We then discuss the architecture of ECDAR 2.0. Lastly, we discuss features concerning IMVE and system views.

5.1 Design Principles

To guide our choices of the design of ECDAR 2.0, we present five design principles in this section. Some of them are inspired by the heuristics evaluation for user interface design proposed by Nielsen (1993). The principles should not dictate the design, as there may be cases where it is hard to satisfy the principles completely, e.g. with conflicting principles. However, we strive to follow the principles throughout the design of ECDAR 2.0. The areas mentioned in Section 4.1 are closely related to these principles.

> **Principle 1** Feedback - ECDAR 2.0 should be responsive in the sense that when a user performs an action, this action has an immediate effect in the UI.

The user should know the current status of the system. Thus, the UI should update whenever an action causes a change in the system. The principle is related to the heuristic of *Feedback*, which says that the user should be informed about the system status within a reasonable time (Nielsen 1993).

To support this principle, we make use of the Model-View-Presenter (MVP) pattern and bindings between the model and view, which among other things helps in updating the UI when the model changes. Read more about the pattern in Section 5.3.1.

> **Principle 2** Error Prevention - ECDAR 2.0 should prevent the user from making invalid models.

Like the heuristic of *Prevent errors*, it is often better to prevent a problem from occurring, than just showing a good error message (Nielsen 1993). This principle coincides with the enforcement of good practices described in Section 4.1.

> **Principle 3** Consistency - New additions to ECDAR 2.0 should be consistent and conform to the design of the H-UPPAAL codebase.

The H-UPPAAL codebase already contains some useful features like keyboard shortcuts. ECDAR 2.0 should also feature shortcuts for new additions to ECDAR 2.0, since users would expect the features to be systemwide. Also the general design of the system views (mentioned in Section 4.2) should look and feel similar to how the modelling of components works.

This principle is related to the *Consistency* heuristic (Nielsen 1993). The heuristic also recommends following platform conventions. That will make it easier for users learn how to use the application. Since ECDAR 2.0 runs on multiple Operating Systems (OSs), it could benefit from adhering to platform standards. The H-UPPAAL codebase already adheres to some conventions. For instance, the keyboard shortcut for saving projects on Windows and Linux is `Ctrl + S`, and the shortcut on macOS is `Cmd + S`.

> **Principle 4** Errors and Invalid Actions - ECDAR 2.0 should provide feedback to the user when errors occur and when user try to perform a prohibited action.

As mentioned in Principle 2 about *Error Prevention*, we want to restrict users from making invalid models. To help them understand the restrictions, we need to provide feedback about why they cannot perform certain actions. The feedback can for example be presented as an understandable error message. The heuristic *Good Error Messages* (Nielsen 1993), describes how a system should provide indicators for errors, so that users are able to recover from them.

> **Principle 5** Productivity - ECDAR 2.0 should provide features so that the user can quickly understand and construct models in ECDAR 2.0.

As mentioned in Section 4.1, ECDAR 2.0 should provide facilities like those of modern IDEs. This may involve keyboard shortcuts, to quickly perform an action, or reduce the steps involved in common actions like refactoring and syntax checking. This is related to the *Shortcuts* heuristic (Nielsen 1993), which mentions that shortcuts are often used by expert users.

Another aspect of this principle is to help users understand models. This may be useful when receiving a model from another user, but also help locate errors. The concept of system views is an example of how users can get an overview of the components involved in a system.

5.2 H-Uppaal

As mentioned in Section 1.1 we build ECDAR 2.0 upon the codebase of H-UPPAAL. In this section we discuss the design changes required in order for the application to conform to ECDAR. Some of these changes are made as a result of the principle about error prevention (Principle 2), as we want the user to only create valid ECDAR models, and not all H-UPPAAL concepts fit in this domain.

H-UPPAAL utilises hierarchies to divide large models into small components. However, hierarchies are not a part of the TIOA modelling language, which ECDAR is based upon. Furthermore, we do not know of any theory that applies hierarchies to the TIOA modelling language. Thus, we choose to remove this feature from ECDAR 2.0.

H-UPPAAL uses the concept of a main component and subcomponents. They are central parts of H-UPPAAL and are tightly coupled with hierarchies. Since we do not continue to work with hierarchies, ECDAR 2.0 does not retain the concept of main component or subcomponents.

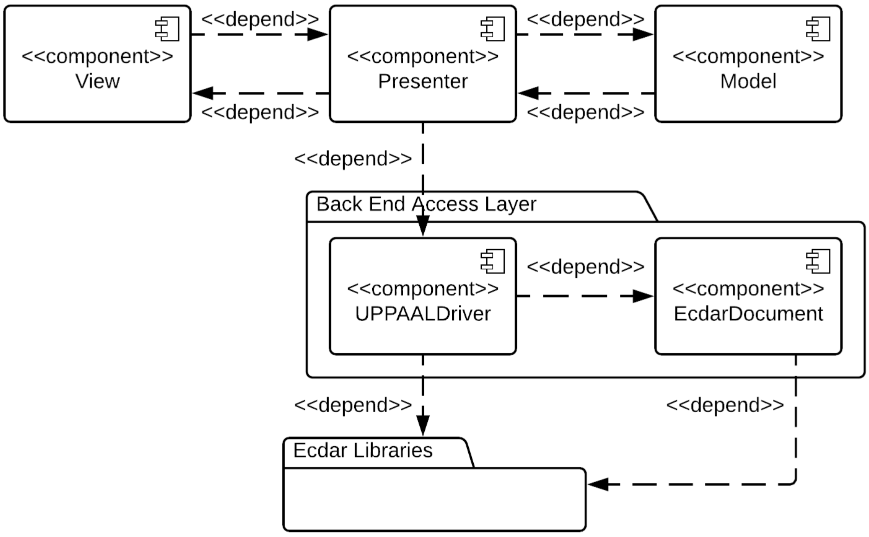


Figure 5.1: A UML 2.5 (Object Management Group 2015) component diagram of the architecture of Ecdar 2.0.

H-UPPAAL also has final locations, which are locations used to transition out of a subcomponent. It is mainly used in hierarchies, to allow the model to exit specific components. As ECDAR neither has final locations in the semantics nor hierarchies, this feature is not needed and is therefore removed.

H-UPPAAL lets users add committed locations. These are locations where, when entered, time is paused and the next transition must include a committed location. Committed locations are not part of the TIOA semantics and therefore needs to be removed.

5.3 Architecture

As we are using H-UPPAAL as our codebase, we inherit the architecture of H-UPPAAL, meaning that ECDAR 2.0 follows the same architecture. The architecture of ECDAR 2.0 is given in Figure 5.1.

The architecture is based on the MVP pattern. The **Presenter** communicates with the **UPPAALDriver**, which handles the communication with the back end and other ECDAR libraries. The **Ecdar Libraries** include `uppaal.jar`, `model.jar`, and the OS specific server file.

5.3.1 The MVP Pattern and Data Binding

The model-view-presenter pattern was first described by Potel (1996) and has throughout the years been used in many different ways (Fowler 2006). The MVP pattern in ECDAR 2.0

is inherited from the H-UPPAAL project. By using MVP, we keep a clear separation of the views and the models, ensuring that any change to the models, triggered by the views, e.g. adding a location to a component, have to be made through the presenters. In this way, the data bindings are initialised in the respective presenters. In this way we also comply with the Principle 1 about feedback as the change on the view happens instantly.

We give an an example of use of data binding and MVP in Section 6.2

5.3.2 Back End Access Layer

The **Back End Access Layer** handles generation of ECDAR back end XML, as well as the communication with the back end. It is inherited from the H-UPPAAL project. The layer consists of two components, **UPPAALDriver** and **EcdarDocument**. **UPPAALDriver** selects what ECDAR 0.10 back end to use based on the OS of the device, and handles communication with that back end. The **UPPAALDriver** also generate back end XML by using the **EcdarDocument** component. The back end is used to run queries.

By separating the functionality and communication from the presenters, we decouple the ECDAR 0.10 libraries and the back end from the front end of ECDAR 2.0. This makes it easier to, for instance, introduce new back ends.

The ECDAR 0.10 libraries and back end are not a part of the front end of ECDAR 2.0.

5.4 Integrated Modelling and Verification Environment

In this section we discuss the design of ECDAR 2.0 related to IMVEs. This involves the handling of the initial location, tau transitions, edge I/O statuses, and the `Inconsistent` and `Universal` locations.

5.4.1 Initial Locations

ECDAR 2.0 should enforce every component to have an initial location according to Section 4.1. In order to make it impossible to construct components without an initial location, ECDAR 2.0 enforces the use of initial locations in the following way:

- When creating a component, ECDAR 2.0 creates an initial location for that component.

- If users try to delete an initial location, that location shakes, and ECDAR 2.0 displays an error message, telling the users that an initial location is required (Principle 4).

By enforcing an initial location, we may limit user control. However, in this case we believe it is more important to avoid invalid models (Principle 2), as we do not expect it to annoy users.

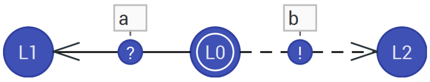


Figure 5.2: Component with an input and an output edge.

5.4.2 Tau Transitions

ECDAR 2.0 should make it impossible to construct tau transitions according to Section 4.1. To do this ECDAR 2.0 enforces the use of synchronisation on all edges.

In H-UPPAAL, an edge has nails that are intermediary points on the path from the source of the edge to the target of the edge. While ECDAR 0.10 places synchronisation on the edge itself, H-UPPAAL places it on a nail of the edge. ECDAR 2.0 enforces the use of synchronisation using such synchronisation nails:

- When creating an edge without nails, ECDAR 2.0 creates a synchronisation nail on it. To preserve the shape of the edge, ECDAR 2.0 positions the nail between the source and target locations of the edge.
- In order to make modelling more efficient, we want experienced users to be able to place synchronisation nails at custom positions while constructing edges. Thus, if a user creates the first nail of an edge while constructing an edge, ECDAR 2.0 makes that nail a synchronisation nail.
- If users try to delete a synchronisation nail, that nail shakes, and ECDAR 2.0 displays an error message (Principle 4).

As with initial locations (see Section 5.4.1), the choice of forcing synchronisation nails prevents errors (Principle 2). It might also improve productivity (Principle 5), as users do not have to add synchronisation nails themselves.

5.4.3 Edge I/O Statuses

ECDAR 2.0 should not allow defining inputs on output edges or vice versa (Principle 2) according to Section 4.1. Instead of letting users add a `?` or `!` as suffix to the synchronisation name (a textfield), ECDAR 2.0 displays `?` or `!` on the nail to denote the I/O status. Furthermore, it uses solid (for input) and dashed (for output) lines for the edge like in ECDAR 0.10. Figure 5.2 illustrates an example with an input and an output edge.

When translating models to XML for the ECDAR back end, we add the appropriate suffix based on the I/O status of the edge.

5.4.4 Inconsistent and Universal Locations

ECDAR 2.0 should feature a way to add inconsistent and universal locations to components visually according to Section 4.1. Therefore, we add two menu elements to the context menu of a component: **Add Inconsistent Location**, which adds an inconsistent location,

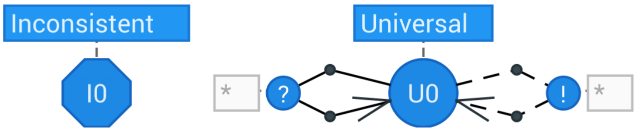


Figure 5.3: Inconsistent and universal locations.

and **Add Universal Location**, which adds a universal location. This improves productivity (Principle 5) as it automatically adds the locations and makes them easily available in a context menu.

If we implement these locations as is, users may accidentally edit the **Inconsistent** or **Universal** location such that it no longer holds the properties of such locations. To avoid users changing the **Inconsistent** and **Universal** locations, ECDAR 2.0 locks the locations such that they cannot be edited, except for letting custom edges end in them, as this does not change the behaviour of the locations. However no custom edges can start from a locked location. Furthermore, users can add nicknames to locked locations, as this does not change the semantics. The locations should be consistent (Principle 3) with the design of other locations, but as mentioned we make some limitations to prevent users from creating invalid models (Principle 2).

ECDAR 2.0 users may want to make their models presentable (i.e. in papers). Thus, we want the **Inconsistent** and **Universal** locations to be movable.

Multiple inconsistent (respectively universal) locations in the same component have the same semantics, so more than one of each type of location does not enhance modelling. However having many custom edges end in the same inconsistent or universal location may clutter a model with crisscrossing lines and thus lessen the readability of a model. Therefore, ECDAR 2.0 allows users to define multiple **Inconsistent** or **Universal** locations in the same component. When translating models to XML, for each component we merge all **Inconsistent** and **Universal** locations such that there is only one `Inconsistent` and one `Universal` location in the XML. When merging the inconsistent and universal locations, all edges that these locations, now end in the same respective location.

The UI of the **Inconsistent** and **Universal** locations can be seen in Figure 5.3. An inconsistent location is just an urgent location without outgoing edges. Universal locations have, by definition, a self looping edge for every input and output actions in the component. To avoid clutter, we instead use the `*` symbol to represent all input or output actions. Lastly, the inconsistent and universal locations uses different identifiers from custom locations. For example `I0`, `I1`, `I2` for inconsistent locations, and `U0`, `U1`, `U2` for universal locations.

5.4.5 I/O Signatures

ECDAR 2.0 should give information about interaction of components according to Section 4.1. Interaction is based on the inputs and outputs. Thus, we propose an *I/O signature* of a component. It contains the input and output channels to use by the component. ECDAR 2.0 visualises this signature as shown in Figure 5.4. Solid arrows going

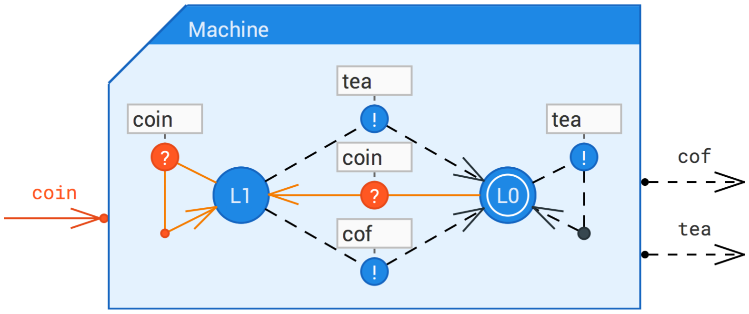


Figure 5.4: Component with visualised I/O signature. The mouse cursor (not visible) is hovering the *coin* input of the signature.

into the component from the left represents input channels. Dashed arrows going out of the component from the right side represents output channels.

To further expand on the overview benefit, moving the mouse cursor to one of the inputs or outputs of the signature, makes ECDAR 2.0 highlight all the edges that use the corresponding channel. In Figure 5.4 the mouse cursor is hovering the *coin* input, and all edges interacting on this channel are highlighted. Note that a channel is only represented as an arrow once, even if there are multiple uses of it. Both the overview and highlighting feature help in understanding models and locating errors (Principle 5).

The visualisation of the signature is responsive (Principle 1), so that users see an immediate change in the signature, when typing the name of a channel. As an example a user creates a new output edge, with a channel named *cof*, as soon as the user enters "c", a new arrow on the right side of the component appears with the label *c*. As the user completes the name, the label is updated to show *co* followed by *cof*. The responsiveness combined with only representing a channel once, has the side benefit of being able to quickly catch typos. In the very simple example of *cof*, if the user were to create a new output edge on the same channel, but instead typed *cod*, a completely new arrow would appear. By moving the mouse cursor to a wrongly typed channel, the user is also able to quickly locate the edge containing the error.

5.5 System Views

In Section 4.2, we introduce the concept of system views. In this section we design the GUI of system views for ECDAR 2.0. We start by defining the overall structure of a system view. Then we design how to present operators and component instances.

Structure: A system in ECDAR consists of instances of first-order components combined with operators. Operators act on other operators (i.e. higher-order components) or on first-order components. Thus, we can represent systems in a tree-like structure with operators as branches and first-order components as leafs. This can be seen in Figure 5.5.

To represent system views, we could allow operators to reference the same component

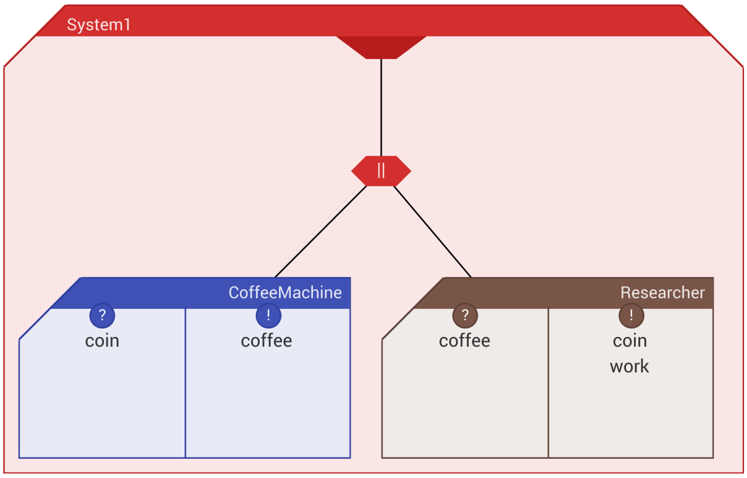


Figure 5.5: A realistic system view. It uses composition (||). Coin, coffee and work represent the I/O signatures.

instances. However we would have to construct semantics for this behaviour and furthermore, ensure that the semantics do not deviate from the TIOA modelling language. A tree structure would not allow this behaviour, and is therefore a better fit than any representation that would allow this behaviour.

As a tree structure fits well with the semantics of systems and does not require us to create new semantics, which is why we choose to represent system views graphically as tree structures.

Since system views are represented as tree structures, they need a root as a starting point. We could either let the user choose a component as a root component, or have the root as the topmost component in the system view. To understand a system view the user must first locate the root, this requires the root to be easily distinguishable from other components. If a component could be set as the root then it might be hard to distinguishing a component that is also a root component from other components of the same type. We therefore choose to have the root as its own UI element. As tree structures are typically represented with the root as the topmost object, we choose to place and locked vertically to the toolbar of the system view. This allows users to drag it horizontally. This can be seen in Figure 5.5.

Operators: In ECDAR you can combine components with the three operators: conjunction, composition, and quotient. These are described more in Section 2.5. Conjunction and composition are both allowed to have two or more operands, which is why we allow these operators to have an arbitrary number of edges going to child nodes. An example of these two operators can be seen in Figure 5.5 and Figure 5.6. The quotient operator has two operands and is non-commutative. It is restricted to only have two operands in system views. An example of the quotient operator can be seen in Figure 5.7.

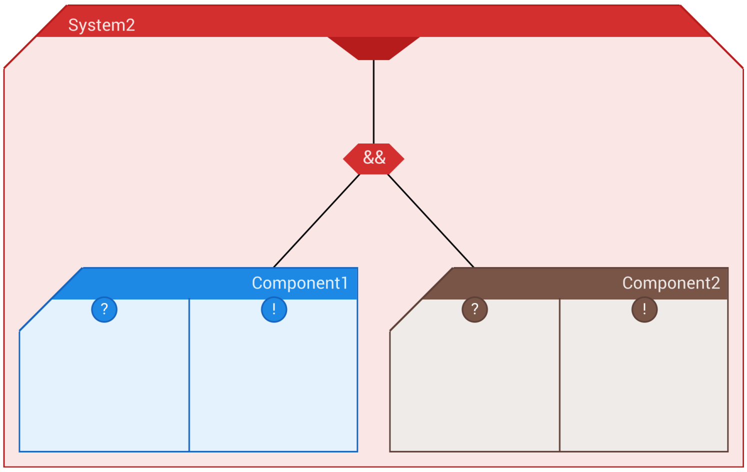


Figure 5.6: A system view that uses conjunction (`&&`).

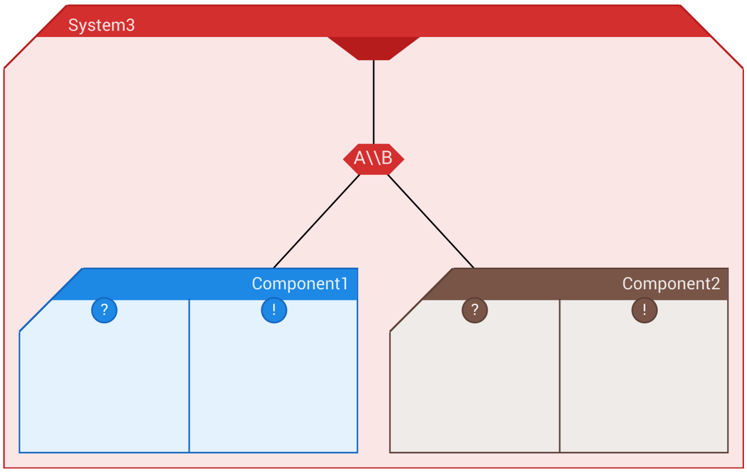


Figure 5.7: A system view using the quotient operator (`A\\B`).

Component Instances: Component instances in system views are instances of specific first-order components developed in ECDAR 2.0. These instances have the same shape as components, furthermore, they use the same colour scheme as the corresponding component. As it is possible to create multiple instances of the same component, we allow for specifying an instance identifier on instances to differentiate between them. This is seen in Figure 5.5.

Component instances need to be compatible with each other in order to perform the conjunction and quotient operator upon these. Compatibility between components differs from the operator that is being used and is determined through their I/O signatures. As we want users to quickly determine whether some components are compatible for certain operators, we show I/O signatures on all component instances (Principle 5). This can be seen in Figure 5.5 for an input *coin* and two outputs *coffee* and *report*.

6 Implementation

In this chapter, we discuss some of the implementation of Ecdar 2.0. A guide on how to access our source code and how to run Ecdar 2.0 is given in Appendix B.

We have developed Ecdar 2.0 through an agile approach and licensed it under MIT[1]. To aid our development, we have enforced the use of code reviews and pull requests through GitHub, and we run Continuous Integration (CI) trough Travis CI[2]. In order to build Ecdar 2.0, we need to use Ecdar 0.10 libraries, which are licensed under other licenses. Thus, our codebase with CI is private.

To make the source code of Ecdar 2.0 public, we have a public GitHub repository[3] that is cloned from our private codebase and then stripped of the aforementioned libraries.

We start this chapter by discussing code documentation. We then discuss the use of bindings and the MVP pattern. Lastly, we outline various improvements of Ecdar done through the project.

6.1 Documentation

Documentation of code is an exercise in software engineering, which we have put emphasis on. The reasons why is that Ecdar 2.0 is an open-source project and also a project which we are going work on in the 10th semester. That is why we have enforced that new and changed code should be documented, especially: classes, interfaces, non-getter and non-setter methods, in addition to constructs that the developers see the need for doing so.

An example of this is the `push()` method (see Listing 6.1) in the `UndoRedoStack` class. Here the name of the method indicates that it only pushes to the stack, however, the method also executes the `perform` parameter. To avoid confusion, we rename the method to `pushAndPerform()` and add the documentation as given in Listing 6.2.

In total, the front end of Ecdar 2.0 has a coverage of Javadoc documentation of 14.96 %[4]. In comparion, the front end of H-Uppaal does not use Javadoc. A detailed report of the documentation coverage for both Ecdar 2.0 and H-Uppaal can be found in Appendix B.

[1] https://opensource.org/licenses/MIT

[2] https://travis-ci.com

[3] https://github.com/tgunde13/SW9ecdarRelease

[4] This includes: parameters, methods, classes, interfaces, enums, fields, and constructors

```java
public static Command push(final Runnable perform, final Runnable undo,
                           final String description, final String icon) {
```

Listing 6.1: Excerpt of the `push()` method in the H-Uppaal `UndoRedoStack` class.

```java
/**
 * Pushes to the stack and performs the redo action once.
 * @param perform the redo action
 * @param undo the undo action
 * @param description a description of the actions
 * @param icon icon of the redo-undo command
 * @return the command created
 */
public static Command pushAndPerform(final Runnable perform, final Runnable undo,
                                     final String description, final String icon) {
```

Listing 6.2: Excerpt of the `pushAndPerform()` method in the Ecdar 2.0 `UndoRedoStack` class.

```java
 1  public void initializeDropDownMenu() {
 2      ...
 3      dropDownMenu.addClickableAndDisableableListElement(
 4              "Draw Edge",
 5              getLocation().getIsLocked(),
 6              (event) -> {
 7                  final Edge newEdge = new Edge(getLocation(),
 8                                                EcdarController.getGlobalEdgeStatus());
 9                  ...
10                  getComponent().addEdge(newEdge);
11                  dropDownMenu.hide();
12              }
13      );
14      ...
15  }
```

Listing 6.3: Excerpt of the `initializeDropDownMenu()` method from the `LocationController` class.

6.2 Data Binding

In this section we present example of code in ECDAR 2.0, that benefit from the use of data binding (through the MVP pattern) between models and views. Both bindings and MVP are presented in Section 5.3.

We explain how a model is updated when a user adds a new edge to a component, and how the corresponding view is updated (the I/O signature on components) to reflect the change.

When modelling a component in ECDAR 2.0, one way for a user to create a new edge, is to right click on a location and select *"Draw Edge"* from the context menu. The right click triggers a `DropDownMenu` object (a context menu) from the corresponding `LocationController` object. Clicking on the *"Draw Edge"* option runs the lambda expression at Line 6 in Listing 6.3. This creates a new `Edge` and adds it to the associated component (Line 10). The associated component is a model of type `Component`. This *"Draw Edge"* action follows the MVP pattern, by having the user make a change in the view (`LocationController`)

```
 1 private void initializeIOListeners() {
 2     final ChangeListener<Object> listener =
 3             (observable, oldValue, newValue) -> updateIOList();
 4
 5     edges.addListener((ListChangeListener<Edge>) c -> {
 6         updateIOList();
 7
 8         while (c.next()) {
 9             for (final Edge e : c.getAddedSubList()) {
10                 addSyncListener(listener, e);
11             }
12
13             for (final Edge e : c.getRemoved()) {
14                 e.syncProperty().removeListener(listener);
15                 e.ioStatus.removeListener(listener);
16             }
17         }
18     });
19     edges.forEach(edge -> addSyncListener(listener, edge));
20 }
```

Listing 6.4: The `initializeIOListeners()` method from the `Component` class.

that updates the model (`Component`).

In addition to edges, a `Component` object also contains collections for which input (`inputStrings`) and output (`outputStrings`) channels are it the I/O signature. These collections are updated when a change occurs in the edge collection. Listing 6.4 shows the code that initialises the listeners for these changes.

The while loop (Line 8) iterates through changes, and adds listeners to new edges or removes the listeners for the edges that have been removed. This means that both `inputStrings` and `outputStrings` are updated when any synchronisation property of any edge in the component changes. The changes can be as small as the user changing the name of a property or as large as opening an existing ECDAR 2.0 project.

The changes to the I/O signature (`inputStrings` and `outputStrings`) need to be reflected in the view. The `ComponentController` and `ComponentInstanceController` classes both use the I/O signature and need to listen for changes. The design for each usage can respectively be found in Section 5.4.5 and Section 5.5. Listing 6.5 shows the method that adds listeners to `inputStrings` (not shown in the listing) and `outputStrings` for the `ComponentController` class. The listener added at Line 2 updates the view by removing existing output signature arrows followed by adding arrows for all channels returned from `c.getAddedSubList` to the view (Line 6). The returned channels represent the newest changes.

Using these listeners, ECDAR 2.0 updates the I/O signature arrows as the user types a synchronisation property. As mentioned in Section 5.4.5 the quick updates of the view makes the UI responsive (Principle 1).

In a similar manner, the `ComponentInstanceController` class also benefits from bindings to `inputStrings` and `outputStrings`.

```
private void initializeSignatureListeners(final Component newComponent) {
    newComponent.getOutputStrings().addListener((ListChangeListener<String>) c -> {
        outputSignatureContainer.getChildren().clear();
        while (c.next()){
            c.getAddedSubList().forEach((channel) ->
                    insertSignatureArrow(channel, EdgeStatus.OUTPUT));
        }
    });
    ...
}
```

Listing 6.5: Excerpt of the `initializeSignatureListeners()` method of the `ComponentController` class.

6.3 Various Improvements

We have made improvements to the H-Uppaal codebase used in Ecdar 2.0, as we experienced both user- and developer-oriented problems. In this section, we outline some of the areas that we have improved upon.

Appendix C contains a complete list of development issues and bug fixes resolved during this project.

Refactoring: We experienced that it was difficult to read and understand code with large methods (e.g. more than 100 lines) with several levels of nesting (lambdas and control structures). To improve the internal structure of the code, we refactored it. The `UPPAALDriver` class is an example of a class, where we believe the readability has been improved by refactoring, such that some control structures span fewer lines and has fewer levels of nesting (issue #99).

In addition to readability issues, we experienced near duplicate code. For instance, most of the presentation classes load their corresponding FXML file, which is the same process for each of them (issue #218). Listing 6.6 presents an example of such loading. We created the `EcdarFXMLLoader` class in order to refactor. The boilerplate code in each presentation classes, can be replaced by a call of the `loadAndGetController()` method in the `EcdarFXMLLoader` class. An example is presented in Listing 6.7.

Bugs: We also spent time on fixing bugs in the codebase. Most were minor bugs encountered while using the early version of Ecdar 2.0. One of the bugs is concerned with loading a project. A Ecdar 2.0 or H-Uppaal project is stored as JSON files in a specific directory. If a file with different filename extension is stored in the directory, the corresponding program would crash (issue #63). We fixed the bug since it is very likely, that a user has another file type in these directories (e.g. text files for describing a project or screenshots of the models). The solution was to ignore non-JSON files.

```java
public NailPresentation(final Nail nail, final Edge edge, final Component component,
                        final EdgeController edgeController) {
    final URL url = this.getClass().getResource("NailPresentation.fxml");

    final FXMLLoader fxmlLoader = new FXMLLoader();
    fxmlLoader.setLocation(url);
    fxmlLoader.setBuilderFactory(new JavaFXBuilderFactory());

    try {
        fxmlLoader.setRoot(this);
        fxmlLoader.load(url.openStream());

        controller = fxmlLoader.getController();
        ...

    } catch (final IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}
```

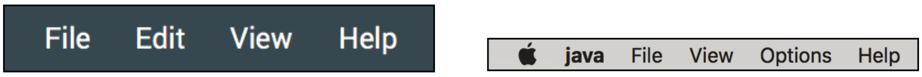
Listing 6.6: Constructor for `NailPresentation` class before refactoring.

```java
public NailPresentation(final Nail nail, final Edge edge, final Component component,
                        final EdgeController edgeController) {
    controller = new EcdarFXMLLoader().loadAndGetController("NailPresentation.fxml",
                                                            this);
    ...
}
```

Listing 6.7: Constructor for `NailPresentation` class after refactoring.

Platform Standards: As mentioned in Section 5.1, we want to adhere to platform standards (Principle 3). We experienced that the menu bar did not follow the macOS standard. Here, applications should share a systemwide menu bar; it changes menu elements when user changes applications. However, the menu bar was inside the application.

To follow the standard, we made ECDAR 2.0 use the macOS menu bar when on that OS. Figure 6.1a shows how the menu bar looked on macOS, and how it still looks on Windows and Linux. Figure 6.1b shows how the menu bar in ECDAR 2.0 currently looks on macOS.



(a) Ecdar 2.0 menu bar on Windows and Linux.

(b) Ecdar 2.0 menu bar on macOS.

Figure 6.1: Two types of menu bars that follow their respective platform standards.

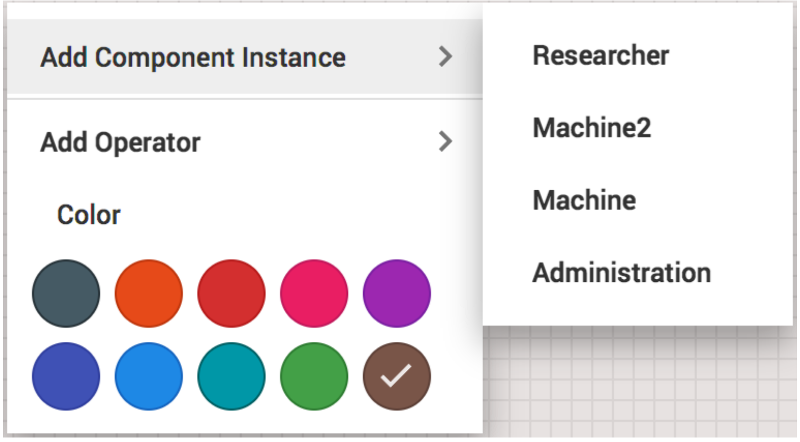


Figure 6.2: Context menu in system view.

Context Menu: The context menu in ECDAR 2.0 gives access to quick actions in several places. The menu is presented when the user right clicks a UI element or presses certain buttons. It is very crucial to the system, since many actions are available and discoverable from it, e.g. "Add Component Instance" (seen in Figure 6.2) and "Delete". The context menu inherited from the codebase had several issues accompanying it: the menu was placed behind other UI elements (issue #28), submenus could cover each other, buttons stayed marked after the menu has been closed (issue #156), and other minor issues.

We fixed some issues by updating the library used for the *Material Design*, JFoenix[5], to the latest version (issue #204), but it also introduced some other minor problems, that we did not deem necessary to fix.

[5] JFoenix GitHub repository: https://github.com/jfoenixadmin/JFoenix

7 Performance Optimisation

Throughout the development of Ecdar 2.0, we experienced lag when using the tool on some devices. In this chapter, we present tests of this performance issue, discuss where in the code the issue lies and present an approach to optimise the performance. Lastly, we test the approach and compare it with the non-optimised version.

Using system monitors, we found that the bottleneck is the Central Processing Unit (CPU) utilisation rather than the memory usage or the disk utilisation. The CPU utilisation is high, even when Ecdar 2.0 is idle and only displaying an empty component. One device has about 100 % utilisation on one of its logical cores (see Figure 7.1) while running Ecdar 2.0.

7.1 Testing

This lead to testing the CPU utilisation of Ecdar 2.0. We tested on three Windows, one macOS, and one Linux configurations. Their hardware specification is listed in Appendix D.

Ecdar 2.0 starts with a window size of 80 % of the screen width and 80 % of the screen height, which is not changed doing testing. In the tests presented in this chapter, we start Ecdar 2.0 and wait for the CPU to become stable before we start measuring. The measurement is done using system monitors native to the corresponding OS (the Windows Resource Monitor, the Linux System Monitor, and the macOS Activity Monitor).

Note that the different configurations used in tests use different variations of Ecdar 2.0 and have different programs running in background. Thus, we do not compare the

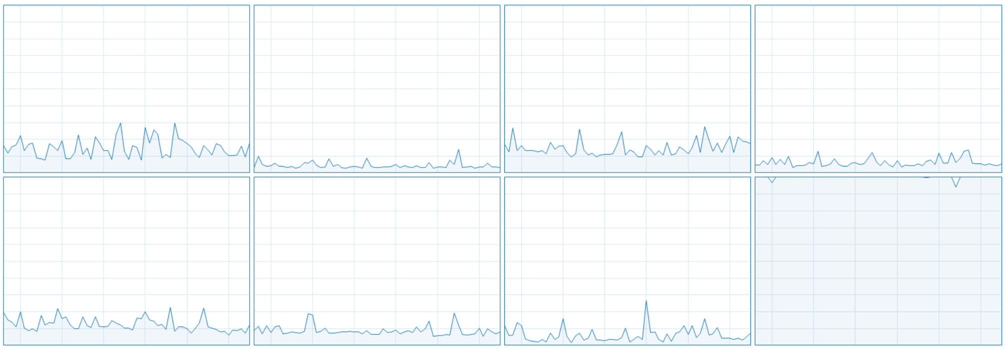


Figure 7.1: Overall CPU utilisation over 60 seconds while running Ecdar 2.0 without optimisation. The figure shows the utilisation of the eight logical cores of Configuration 1. Each block ranges from 0 % (bottom) to 100 % (top) and from 0 seconds (left) to 60 seconds (right). The utilisation is captured with the Windows Task Manager.

Configuration	Non-optimised	Caching	Comparison
1	13.17 %	5.75 %	- 56.34 %
2	24.67 %	7.29 %	- 70.45 %
3	12.53 %	1.63 %	- 86.99 %
4	19.5 %	25.75 %	+ 32.05 %
5	53.9 %	21.05 %	- 60.95 %

Table 7.1: CPU utilisation of Ecdar 2.0 without optimisation and with caching. The last column is the reduction (-) or increase (+) of utilisation with caching enabled compared with no optimisation.

configurations with each other.

The utilisation on Windows is measured as the average percentage of CPU consumption by the ECDAR 2.0 process over 60 seconds. The utilisation on the other OSs is measured by observing the lowest and highest utilisation by the ECDAR 2.0 process two times, each over 15 seconds, and then averaging the observations. Note that we this way assume the utilisation to be evenly distributed between lowest and highest. Appendix E presents the observations for the Linux and macOS configurations. The utilisation for the different configurations are presented in Table 7.1.

7.2 Analysis

In this section we search for a cause for this amount of CPU utilisation and try to reduce it.

By recursively removing parts of the code and observe the difference in utilisation we narrowed it down to the `CanvasPresentation`, `ComponentPresentation`, and `Ecdarpresentation` views. Our observation shows that the background queries did not seem to have a significant impact, as one might think.

We tried different approaches to reduce CPU utilisation, for instance caching views as a bitmap. Caching speeds up subsequent renderings in many cases, but also increase memory usage according to Oracle (2015). Caching of JavaFX views is disabled by default (Oracle 2015).

Enabling caching for `EcdarPresentation` seemed to reduce CPU utilisation a significant amount on some devices. Changing the settings for the caching and what to cache did not seem to improve it any further. To decide whether to use cache, we measure the CPU utilisation with caching enabled on `EcdarPresentation`. We uses the same test setup as with the previous tests.

Figure 7.2 shows the overall CPU utilisation. This shows that the utilisation no longer reaches 100 % utilisation on any of the logical cores.

The utilisation for the ECDAR 2.0 process is given in Table 7.1. The table also illustrates the reduction or increase in utilisation when caching compared to no optimisation. Most of the configurations experience a significant reduction (56 % to 87 %) in CPU utilisation.

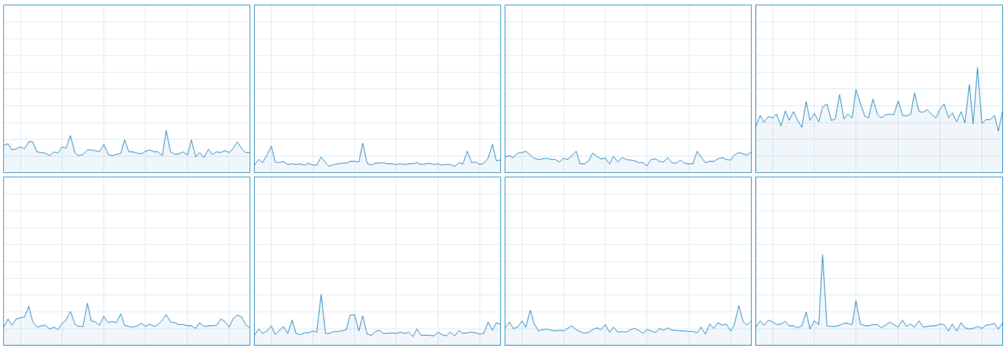


Figure 7.2: Overall CPU utilisation over 60 seconds while running Ecdar 2.0 with caching enabled. The same programs are running in backgrounds as with Figure 7.1. We also use the same system monitor.

Only one configuration (the one running Linux) experienced an increase in utilisation.

7.3 Solution

Since caching in general reduces CPU utilisation, ECDAR 2.0 now enables it as default. However, there are still reasons for a user to disable caching, for instance, as we saw with Configuration 4, on some configurations caching increases CPU utilisation. For this reason, ECDAR 2.0 users can now toggle caching on and off through the top menu.

8 Discussion

In this chapter we discuss the conducted interviews, caching of views, and how we could test ECDAR 2.0.

8.1 Interviews

The interviews we conducted (described in Section 3.3), only questioned a very small and likely biased user group. We need to be critical in the evaluation of the results from the interviews, as they are not sufficient to represent all future users of ECDAR. Despite this concern, we can still use the results as inspirations and to discuss internally in the project group. The idea of system views was suggested during an interview, and was discussed internally before we decided to design and implement it.

It is challenging to perform a proper evaluation using the small user base of ECDAR 0.10. If we were to proceed with an evaluation, we could send out a survey to people who have used tools based on UPPAAL (like ECDAR 0.10 is). This survey should be about how they use UPPAAL, and cannot be too specific about the TIOA theory or queries related to ECDAR.

Another suggestion would be to make a usability evaluation of ECDAR 2.0. This suggestion is further described in Section 8.3.

8.2 Caching

ECDAR 2.0 caches `EcdarPresentation` (our root view) as default in order to reduce CPU utilisation according to Chapter 7. However, such caching might not be optimal for all views in all situations. Rather, we could configure caching of individual views.

If a view will be animated or scaled over time, the cache might not be usable at the time of the next frame. To optimise, we could set a `CacheHint`[1] on such views. These could be set either at compile-time or run-time. We might also want to disable caching on some views completely.

8.3 Testing

For now we have only conducted a performance of ECDAR 2.0 but other types of test could also be conducted. In this section we discuss the potential tests that could be conducted on ECDAR 2.0.

We could use *unit testing* to conduct a functional test of the front end of ECDAR 2.0 . We could also use it to conduct an integration test of the verification engine.

[1] https://docs.oracle.com/javase/8/javafx/api/javafx/scene/CacheHint.html

As ECDAR 2.0 load projects from a directory structure of JSON files, a user might accidentally try and load a wrong folder. Furthermore users can edit project files, before loading them with ECDAR 2.0. E.g a user might want to duplicate a part of a component, and does so by duplicating the text in the JSON file. If done incorrectly, the project files might become invalid. We could test how robust ECDAR 2.0 is at handling such corrupt files.

We should conduct UI tests. For such tests we could use the test framework TestFX[2], which work by clicking on UI elements and measure the state of views. In this way, we test the flow of the UI and whether the UI elements are working as intended. Such tests could be included in our CI environment

We designed ECDAR 2.0 according to the principles in Section 5.1. We could perform a *usability evaluation* to assess how well we followed these principles. Furthermore, we could gain knowledge about users' expectations from the tool and how they approach a modelling task. As there are few people using ECDAR 0.10, we could perform the test on users of UPPAAL.

[2] https://github.com/TestFX/TestFX

9 Conclusion

We have developed a front end as an Integrated Modelling and Verification Environment (IMVE) for compositional real-time systems based on the theory of timed input/output automata called ECDAR 2.0. We have built it to improve the UI of ECDAR 0.10. Together with the back end of ECDAR 0.10, it can be built as a Java 8 program.

ECDAR 2.0 is a modelling and verification tool inspired by features of modern IDEs, or as we call them, IMVE features. ECDAR 2.0 is built upon the codebase of H-UPPAAL, which has already fixed some UI issues with UPPAAL (see Section 1.1). With some basic functionality in place, we choose to limit our scope (in Chapter 4) to two areas of development: IMVE features and system views. We also establish five design principles (see Section 5.1) to guide the design of ECDAR 2.0. The development issues we have finished during this project are listed in Appendix C.

9.1 Integrated Modelling and Verification Environment

ECDAR 2.0 is built around the MVP architectural pattern and data binding in order to provide quick feedback (Principle 1). For instance, the I/O signature on a component is immediately updated as a user changes the synchronisation property of an edge.

To assist users in constructing valid models, we enforce good practices (Principle 2). For instance, we enforce the presence of the initial location, and remove the possibility to construct tau transitions.

The design of ECDAR 2.0 should be consistent (Principle 3). ECDAR 2.0 should consistently use features like keyboard shortcuts and context menus. Component and system views should be consistent with each other, meaning that the same shortcuts and actions should be available from both views, whenever it makes sense. As we did not finish all features for system views, it is also missing some of the features that makes it consistent with component views (e.g. nails). We have also implemented features to adhere to platform standards, like the menu bar on macOS.

ECDAR 2.0 has also been designed to notify the user of errors and invalid actions (Principle 4). As an example, users are notified of the invalid action, if they try to delete the synchronisation nail on an edge or the initial location. In system views users are also notified, if an edge between two elements is invalid. An example of an invalid edge is between two component instances.

A good IDE (and IMVE) should have facilities to increase developers' (respectively modellers') productivity (Principle 5). ECDAR 2.0 includes project management features, such as, creating a new project and saving a project as a new one. These features are not present in H-UPPAAL. Models made in ECDAR 2.0 may be used in documents and academic papers, so we have developed features to export models as images. ECDAR 0.10 requires users to manually construct `Inconsistent` and `Universal` locations for specific queries, and it can especially be a hassle for users to maintain `Universal` locations. To make it less of a hassle for users, ECDAR 2.0 automatically generates such locations, including the edges needed for `Universal` locations.

9.2 System Views

We have also introduced system views to ECDAR 2.0. A system view represents a system of component instances combined with the operators conjunction, composition, and quotient. In ECDAR 0.10 users run queries on a declared system, which the user has to define in a system declarations file. System views are in the same way designed to represent a system declarations file. As with components, a system can also be exported as an image.

We have mainly implemented the graphical part of system views in the current version of ECDAR 2.0. The complete benefit from using system views requires implementation of features such as aliasing and generation of system declarations.

9.3 Performance Optimising

We have tested the performance of ECDAR 2.0, as it has been a noticeable issue. We tested the performance when ECDAR 2.0 was idle and found that the CPU utilisation of the process was unnecessary high. To combat this, we added the option to cache the root view of ECDAR 2.0. This gives us a *reduced utilisation of 56 % to 87 %* on four out of our five test configurations.

10 Future Work

In this chapter we consider the future development of features and improvements for ECDAR 2.0. It is important to consider as we will be continuing the work on ECDAR 2.0 in the next semester. Furthermore, we expect others will continue working on ECDAR 2.0.

10.1 Multiple Engines

The front end of ECDAR 2.0 uses the back end of ECDAR 0.10 to perform verification. Contrary to ECDAR 0.10, PYECDAR features robustness analysis and explicit computation of conjunction, composition, and quotient. We could add PYECDAR as an additional *choice of engine* for ECDAR 2.0. For instance, when writing a query, users could choose what engine to use for that query. Furthermore, if other compatible engines exist, we could add those too.

10.2 Simulator

The simulator in ECDAR 0.10 can be a useful tool for debugging and understanding models. During the interview (see Section 3.3.1), we questioned the interviewees about their use of the simulator in ECDAR 0.10. They confirmed the usefulness of a simulator, but also brought attention to flaws in the ECDAR 0.10 simulator.

Mourtizsen and Jensen (2017) also considered to add a simulator to H-UPPAAL, and they state the challenges like this: "The challenge with this (ed. the simulator) is not retrieving the trace and states from the verification engine, which is already possible through its interface. Instead the challenge is presenting these states and traces to the user." — (Mourtizsen and Jensen 2017). We face the same challenges as Mourtizsen and Jensen (2017), with the main challenge being to design a simulator, and how transitions and interaction between components should be displayed. A simulator extension for ECDAR 2.0 needs to adhere to the design principles in Section 5.1. An example feature, suggested in the interviews, that follows the feedback principle (Principle 1) is to *animate a transition in the simulator*.

10.3 Version Control Systems

Another extension suggested by the interviewees is to include support for VCSs such as Git and SVN. An extension like this could change the way that they are collaborating on modelling projects today. The process of working with models, described by the interviewees, reminded us a bit about a plan-driven process. Furthermore, it seems to us that the process to get a model accepted is tedious and lengthy. As TIOA already affords stepwise design e.g. through refinement, a VCS extension could move the process of making these models towards a more agile modelling process.

The interviewees also proposed a diff tool for ECDAR 2.0. The diff tool could be able to display changes from one commit to another, and also run the same queries on the different versions of the same model.

10.4 Mutation Testing

The interviewees also suggested to make a mutation tool for ECDAR 2.0. Larsen et al. (2017) presents an approach to conducting model-based mutation testing. They construct a TIOA in ECDAR and uses it as a test model. They then use a mutation tool to generate mutations of the test model. Then they use the refinement operator of ECDAR to check which mutants conform to the test model.

The engine of ECDAR provides a strategy for a non-refinement. These are either winning or cooperative strategies that show how the corresponding mutant does not conform to the test model. Thus, if a system under test contains the same fault as a non-conforming mutant, the corresponding strategy will also show how the system does not conform to the test model.

Because of this fact, running (in parallel) the test model, a mutant, and the system under test, we can test the system for the faults represented by the non-conforming mutants.

Instead of using an external mutation tool to mutate ECDAR back end XML, we could integrate mutation in ECDAR 2.0. ECDAR 2.0 could for instance use a system view as the test model. We could also automate the process of conformance checking all generated mutants and the process of generating the test-cases. Likewise, ECDAR 2.0 could include a feature to conduct the mutation testing on some types of systems under test.

10.5 SMC

UPPAAL SMC[1] is an extension of UPPAAL that provides *Statistical Model Checking (SMC)*, which ”refers to a series of techniques that monitor several runs of the system with respect to some property, and then use results from the statistics to get an overall estimate of the correctness of the design” (David et al. 2011). We could look into the possibility of applying SMC in ECDAR 2.0.

10.6 Integrated Modelling and Verification Environment

We introduced the term IMVE in section Section 4.1. In this section we present features to further assist in modelling and verification.

[1] http://people.cs.aau.dk/~adavid/smc

Continuous Syntax Checking: ECDAR 0.10 only performs syntax checking when building or when users manually issue checks. To help users find faults earlier in the process, we could continuously perform syntax checks. H-UPPAAL already supports this for UPPAAL model checking.

Model Refactoring: Refactoring is changing internal structures while preserving the external behaviour (Fowler and Beck 1999). You can refactor manually, however, this becomes more time consuming and complex for larger systems, possibly leading to more faults. IDEs have features to assist code refactoring, such as changing names and structures, and to extract logic. We want similar features for models in ECDAR 2.0.

Background Analysis: IDEs use static code analysis to detect common mistakes and bad practices. We could do static analysis of TIOA, which could include checking for unused clocks, variables, and functions.

Component Labels: If a component contains a fault, it might not have the expected behaviour. To help the user notice some faults immediately, we could introduce *component labels*. One way to do so would be to label inconsistent components and implementations. An example is that a user intends to construct an consistent component, but the component contains a fault making it inconsistent. With labels, the user could immediately see that it does not have the intended behaviour.

I/O Signature Features: The I/O signature mentioned in Section 5.4.5 could be improved upon. It could let users rename a signature such that all edges using this signature gets renamed as well. Furthermore, it could indicate whether a channel is used in other components, and show the related components. This would further improve productivity (Principle 5).

System Declarations: In ECDAR 2.0 the system declarations must manually be defined. The System declaration are also query dependent. Thus, ECDAR 2.0 users must change the declarations when they run different queries. To combat this, we could work on generating system declarations instead.

Cloning: ECDAR 2.0 users cannot clone components or systems. Cloning can be a productive assistance, if users want multiple similar components or systems. Also, in stepwise design, we expect users to make new iterations of components and systems. We believe it would be beneficial to be able to do this while preserving old versions of components and systems in stepwise design. Thus, ECDAR 2.0 users should be able to clone components.

10.7 System Views

We discusses, in Section 5.5, the design of the UI of system views. However multiple features were not implemented. In this section we will discuss the missing features of system views as potential future work.

UI Boundaries: We represent system views as tree structures. In a tree structure representation, the parents are typically above their children. In the current implementation of system views, there is nothing that prevents users from placing child nodes above their parents. We would like to set boundaries on dragging of nodes such that children are always below their parents. This would preserve the tree structure visually. For the quotient operator, we could use dragging boundaries to restrict the operands in such a way, that the left operand (in the TIOA theory) is always to the left of the operator and the right operand (in the TIOA theory) is on the right side. The boundaries limit the user's freedom of control, but has the benefit of preserving the tree structure.

Aliases and Generation of System Declarations: According to Section 4.2 we want users to be able to use system views in queries. This requires us to be able to generate system declarations from system views. Furthermore we need to implement aliases of system views, so that they can be used in queries.

10.8 Backlog

While working on ECDAR 2.0 we accumulated a number of issues in our backlog. These issues deal with fixing bugs, and adding missing features. The backlog issues are not necessarily inside the scope of this project, but suggestions for future work on ECDAR 2.0. In this section, we will go through some issues from the backlog. An overview of all backlog issues can be found in Appendix F.

Background Queries: This concerns issue #105. As we have inherited functionality for background queries from the H-UPPAAL codebase. It can easily be extended to queries needed for ECDAR. An example of background queries for ECDAR could be: check for output urgency, independent progress, and consistency. Output urgency and independent progress are interesting as they are required for a specification to be an implementation, and consistency check since it is needed for refinement checking. By using background queries, ECDAR 2.0 can also label specifications accordingly (as suggested in Section 10.6).

Error Handling when Opening Files This concerns issue #58. During development we have followed the principle Principle 2. However, in ECDAR 2.0 it is possible to open invalid files. This is not properly handled and can lead to crashes and errors. If a user opens an invalid file or a file that has been edited by the user in such a way that it breaks certain rules, ECDAR 2.0 should provide proper errors to prevent crashes.

Verification Options: This concerns issue #132. This issues deals with implementing the different verification options from ECDAR 0.10 into ECDAR 2.0. ECDAR 0.10 has different options for verification, e.g forward search order. These allow users to customise the verification process, sometimes leading to a shorter verification time. ECDAR 2.0 does currently not support these options.

Bibliography

Behrmann, Gerd, Cougnard, Agnes, David, Alexandre, Fleury, Emmanuel, Larsen, Kim Guldstrand, and Lime, Didier (2007). “Uppaal-tiga: Time for playing games!” In: *CAV*. Vol. 4590. Springer, pp. 121–125.

Benyon, David (2013). *Designing interactive systems: a comprehensive guide to HCI, UX and interaction design*. 3rd ed. Pearson Education Limited. ISBN: 9781292013848.

Clarke, Edmund M. (2008). “The Birth of Model Checking”. In: *25 Years of Model Checking: History, Achievements, Perspectives*. Ed. by Grumberg, Orna and Veith, Helmut. Springer Berlin Heidelberg, pp. 1–26. ISBN: 978-3-540-69850-0. DOI: 10.1007/978-3-540-69850-0_1.

David, Alexandre, Larsen, Kim G., Mikucionis, Marius, Bulychev, Peter, Zheng, Wang, and Legay, Axel (2011). *Statistical Model-Checker*. http://people.cs.aau.dk/~adavid/smc. Accessed 8th of January 2018.

David, Alexandre, Larsen, Kim G., Nyman, Ulrik, Legay, Alex, and Wąsowski, Andrzej (2017). *University Example*. http://people.cs.aau.dk/~adavid/ecdar/examples.html.

David, Alexandre, Larsen, Kim Guldstrand, Legay, Axel, Nyman, Ulrik, and Wasowski, Andrzej (2010). “Timed I/O Automata: A Complete Specification Theory for Real-time Systems”. In: *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*. HSCC ’10. Stockholm, Sweden: ACM, pp. 91–100. ISBN: 978-1-60558-955-8. DOI: 10.1145/1755952.1755967.

Fowler, Martin (2006). *GUI Architectures. Model-View-Presenter (MVP)*. Accessed 6th of January 2018. URL: https://martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp.

Fowler, Martin and Beck, Kent (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Larsen, Kim Guldstrand, Lorber, Florian, Nielsen, Brian, and Nyman, Ulrik Mathias (2017). “Mutation-Based Test-Case Generation with Ecdar”. In: *Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2017*. IEEE, pp. 319–328. DOI: 10.1109/ICSTW.2017.60.

Legay, Axel and Traonouez, Louis-Marie (2013). “PyEcdar: Towards Open Source Implementation for Timed Systems”. In: *Automated Technology for Verification and Analysis: 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*. Ed. by Van Hung, Dang and Ogawa, Mizuhito. Springer International Publishing, pp. 460–463. ISBN: 978-3-319-02444-8. DOI: 10.1007/978-3-319-02444-8_35.

Legay, Axel, Traonouez, Louis-Marie, and Fahrenberg, Uli (2013). *Differences with ECDAR*. https://project.inria.fr/pyecdar/differences-with-ecdar/. Accessed 8th of November 2017.

Mourtizsen, Niklas Kirk and Jensen, Rasmus Holm (2016). *Introducing Hierarchies to Networks of Timed Automata - HUPPAAL - a New Integrated Development Environment for Model Checking*. Aalborg University.

– (2017). “Improving the Model Checking Activity Using H-UPPAAL a New Integrated Development Environment for Model Checking”. Master’s thesis. Aalborg University.

Nielsen, Jakob (1993). *Usability Engineering*. 1st ed. Academic Press. ISBN: 0125184069.

Object Management Group (2015). *OMG Unified Modeling Language*. http://www.omg.org/spec/UML/2.5/PDF.

Oracle (2015). *Node - Cache*. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html#cacheProperty. Accessed 6th of January 2018.

Owicki, Susan and Gries, David (1976). “Verifying Properties of Parallel Programs: An Axiomatic Approach”. In: *Commun. ACM* 19.5, pp. 279–285. ISSN: 0001-0782. DOI: 10.1145/360051.360224.

Potel, Mike (1996). “MVP: Model-View-Presenter the Taligent programming model for C++ and Java”. In: *Taligent Inc*, p. 20.

A Interview Questions

General
What is your experience with ecdar or Uppaal, in years or months?
How would you typically approach an Ecdar project? • What is the first thing you do when creating a new project?
Tell me three things you like about the Ecdar tool?
Tell me three things you do not like about the Ecdar tool?
Do you work on models by yourself or do you collaborate with other people?
Design Process
Tell me about the process of designing templates
Incremental process or are the templates sufficient after the initial creation?
Organisation and layout of models • What is the typical size of your Ecdar model? • How many templates are typical and how many locations do they contain?
How would you know if your code behind your model is correct?
Verification
What are some things you usually want to verify in a compositional system?
Which types of queries are frequently used? • Are these queries used in most of your Ecdar models?
Are there any queries that you use while designing a model, and not necessarily needed for the final verification? - and which? (e.g. check for consistency of components)
Do you declare multiple systems (in system declarations) and change them on-the-go?
How often do you perform these queries? Do you switch between the model editor and query tab often?

Table A.1: The question used for the user interview.

Simulator
How do you use the simulator?
How often do you use the simulator? And in which way?
What do you use the simulator for?
Is the simulator important to your workflow? - why?
Last remarks
Is there something you would like to add?
Do you have other comments about ECDAR?
Did we forget to ask you about something?
Ask for a single wish in a new version of ECDAR.

Table A.2: The question used for the user interview, cont.

B Attachments

There is a `deis902e17.zip` file attached to the submitted project. The file can also be downloaded from http://projekter.aau.dk/. The file contains the following:

- A `README.md` file that describes how to access the rest of content of the zip file and how to build and run ECDAR 2.0 with it.
- Audio files of the conducted interviews. Please note that the interview with our supervisor were conducted in Danish.
- Source code of the front end of ECDAR 2.0 as a Java 8 project.
- Three ECDAR 2.0 sample projects.
- Detailed Javadoc coverage reports of the front ends of ECDAR 2.0 and H-UPPAAL.

C Finished Issues

This appendix presents all development and bug issues that were finished during this project.

Number	Name	Type
#1	Copy code for H-UPPAAL	development
#12	Save model / template as image	development
#14	Removed committed locations	bug
#15	Stop the warning box from popping up when adding a new location	development
#16	Keyboard shortcuts for creating a new component	development
#17	Make the bottom bar remember its height when it has been changed	bug
#20	Move top menu to correct positions for MacOS	development
#21	Make In- and Output edges	development
#23	Rename file panel to project panel	development
#28	Context menus should be fully visible	bug
#33	Help menu in Ecdar is wrong about final location	bug
#34	Create build server for the project	development
#36	Pressing an item in the help menu adds something to the UndoRedoStack	bug
#38	Add Input / Output field to the json objects	development
#41	Translate I/O edges	development
#52	Option to remove grid	development
#54	Fix the bottom item in the component and query lists is cut-off	bug
#57	Add a save as in the file menu	development
#63	Application crashes if any files other than .json files are located in the project folder	bug
#64	Auto crop exported png	development
#77	renamed push to pushAndPerform	bug

Table C.1: First part of the list of issues that we have finished in this project.

Number	Name	Type
#78	Create New Project button	development
#79	UndoRedoStack not cleaned after open/create project	bug
#82	Rename Huppaal components to Ecdar	bug
#84	Remove Main Component	development
#90	Cannot export PNG when the ECDAR project has not been saved	bug
#91	Refactor (de)serialize of project into Project class	bug
#96	Simplify gradle	bug
#97	Remove subcomponent from project	development
#99	Refactoring Uppaal driver	bug
#100	Fix exception handling in various of files	development
#107	Fix the nasty memory usage	development
#113	New / open project does not clear declarations	bug
#117	Removed main component check from query	development
#135	The channel name should not contain "!" or "?"	development
#140	Add Universal/Inconsistent locations	development
#141	Add I/O Signatures	development
#151	A new location is added twice to the undo-redo stack	bug
#152	Refactor, e.g. unused imports/classes	development
#156	Dropdown items stay marked	bug
#162	Force sync on edge by automatically adding it	development
#163	Refactor context menu	development
#174	Add listener on synchronisations	development
#175	Scaled export, removed add synchronization menu item	development
#179	System views - Implementation af Designet / UI	
#182	System view - UI for operations + programming	
#190	Export without border, toggle dcl button when export	development
#204	Update jfoenix to latest version	development
#208	Implement a better solution to id's than atomic integer	development
#210	Highlight edges when mouseover on an arrow in signature	development
#217	System view snap to grid + Refactor	development
#218	FXML loader refactor	development
#220	Added AGExample to ecdar examples	development

Table C.2: Second part of the list of issues that we have finished in this project.

Number	Name	Type
#222	Change the I/O button	development
#223	Component instance frame and background	development
#225	Component instance shape, name, toolbar, draggable	development
#226	New icons for system and components and added button for add system and add component in project pane	development
#227	Component instance selectable	development
#229	System Views - Add signature to components and systems	development
#240	Add system root	development
#243	FJXToggleButton for edge status	development
#244	System views operators	development
#254	Shift click to create edges in system	development
#257	System view - Save Component, operators and edges as JSON	development
#258	Export a system as png (with / without border)	
#261	Snap to grid and delete	development
#263	Drag bounds and system edge selectable	development
#265	Made it possible to connect edges to operators, minimal restrictions	development
#271	Adds an initial version of the university example to Ecdar samples	bug
#281	System views - UI for components + programming	development
#290	jar build update and license added	bug

Table C.3: Third part of the list of issues that we have finished in this project.

D Test Configurations

Configuration	CPU	Screen Resolution	OS
1	Intel Core i7-7700HQ @ 2.80GHz	1920 x 1080	Windows 10, v. 1079
2	Intel Core i5-6200U @ 2.30GHz	1366 x 768	Windows 10, v. 1079
3	Intel Core i7-3537U @ 2.00GHz	1366 x 768	Windows 10, v. 1079
4	Intel Core i7-3537U @ 2.00GHz	1366 x 768	Arch Linux Gnome v. 3.26.2, Linux kernel 4.14.11-1
5	Intel Core i7-3615QM @ 2.30GHz	2880 x 1800	macOS v. 10.13.1

Table D.1: The device configurations used in the tests presented in Chapter 7. Configuration 3 and 4 uses the same device. All devices used are laptops running on AC power rather than on battery.

E CPU Utilisation Observations

In this chapter we present the observations for configurations 4 and 5. The observations without optimisation is given in Table E.1. Those with caching is given in Table E.2.

Configuration	Low	High	Average
4	19 %	20 %	19.5 %
4	19 %	20 %	19.5 %
5	49.7 %	54.2 %	51.95 %
5	51.7 %	60.0 %	55.85 %

Table E.1: CPU utilisation of Ecdar 2.0 over 15 seconds on Configuration 4 and 5 without optimisation.

Configuration	Low	High	Average
4	24 %	26 %	25 %
4	25 %	28 %	26.5 %
5	14.4 %	26.6 %	20.5 %
5	17.6 %	25.6 %	21.6 %

Table E.2: CPU utilisation of Ecdar 2.0 over 15 seconds on Configuration 4 and 5 with caching enabled on `EcdarPresentation`.

F Backlog Issues

This appendix contains all issues that we were unable to finish in this project.

Number	Name	Type
#11	Add zoom to the tool	Development
#13	Save model / template as pdf	Development
#40	Let users open Ecdar from a project folder	Development
#43	Make a menu for keybindings	Development
#47	Add label for urgent keybinding	development
#49	Naming a component "queries" will make the component be read as the query json file when loading	Bug
#55	Add a parser for queries	Development
#58	Better error handling, when trying to open a wrong folder	Development
#60	Add button for showing the QueryPane	Development
#81	Make image export to make it possible to export multiple components	Development
#83	Shows number of warnings when there are none	Bug
#85	Wrong syntax highlighting, does not use spaces	Bug
#86	Functionality to duplicate a component	Development
#103	Open from Ecdar 0.10 XML file	Development
#105	Background queries for consistency, output urgency, and independent progress	Development
#106	Consider redesign/update of Query panel	Development
#111	Easy way to change source/target location of an edge	Development
#115	Deadlock check might not be correct	Bug
#127	Dismiss the error tabpane using the ESC key	Development
#132	Verification Options	Development
#142	Change Colour Theme of Ecdar	Development
#146	Copy/pasting a group of locations/edges	Development

Table F.1: First part of the list of issues in our backlog after finishing this project.

Number	Name	Type
#147	Version control support. Integration with git, version and branches	Development
#160	Find components which can synchronise with a synchronisation	Development
#161	Automatically add channel when used in a synchronisation to Global declarations	Development
#168	Implement parameters for templates	Development
#180	System views - Generate system declarations from system views or from queries	Development
#183	System views - Generate queries	Development
#184	System views - Alias	Development
#186	Better positioning of Inconsistent and Universal state in xml document	Development
#203	Nickname textfield is in focus after edge/location deletion	Bug
#221	Allow adding extra sync arrows to I/O signatures	Development
#224	Deleting the last component should change the canvas to the Declarations file	Bug
#231	I/O signature does not show æøå or ÆØÅ	Bug
#232	Query takes focus when changing input and output	Bug
#233	Old icons in status/error tabPaneContainer	Bug
#238	System view - Collapse component and systems so they are small	Development
#239	"New Project" shows a native JFX dialog	Bug
#241	Color selector in Ecdar Presentation is enabled when selecting a component instance	Bug
#242	Delete button in Ecdar Presentation does not work on component instances	Bug
#245	We should only make Inc and Uni if user uses it or uses quotient	Development
#252	System View - Add nails to edges	Development
#267	System Views - Bounds for edges on operators	Development
#270	Two submenues can be open at the same time	Bug
#295	Automatically do angelic completion	Development
#296	Update the help menu	Development
#298	Declaration not selected on FilePane	Bug

Table F.2: Second part of the list of issues in our backlog after finishing this project.