Uppaal. The Model Checker.

Patryk Kiepas

March 22, 2016

Uppaal. What is it?

Uppaal is a model checker for real-time systems (in mind of embedded systems). What we can do with it?

1. Modeling
2. Simulation
3. Verification

Internal representation of model consists of:

- Network of timed automata
- Extended with data types

Where to use?

"Any system can be analyzed using a model checker, as long as it has *states* and *transitions* between states" (from Chapter 1: A First Introduction to Uppaal by Frits Vaandrager)

Reactive systems such as:

- Hardware components
- Embedded controllers
- Network protocols
- Others...

Whenever there is need to handle real-time issues (the timing of transitions).

Brief history

Uppaal was started by Uppsala University, Sweden and Aalborg University, Denmark.
Timeline of development:

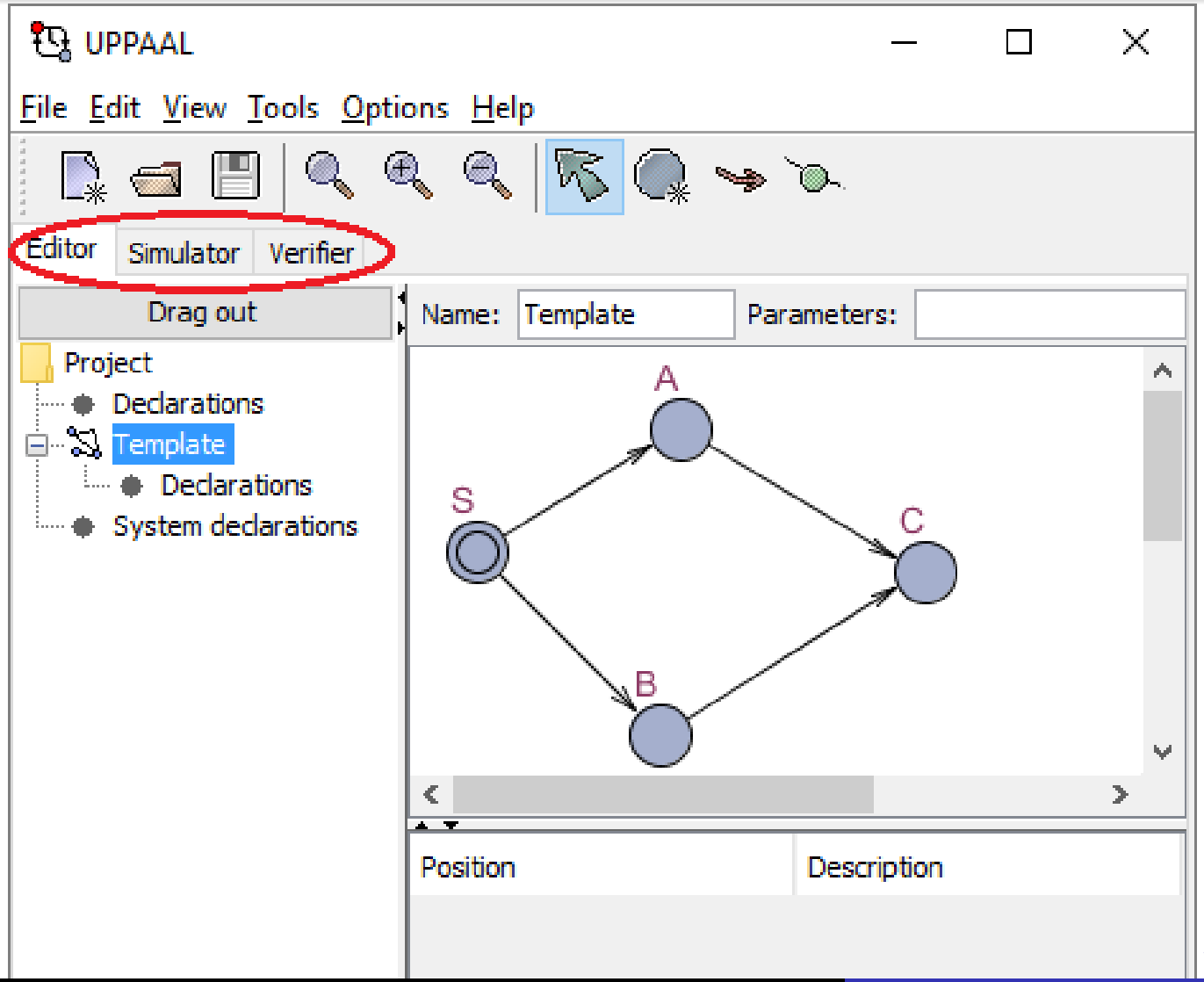
- 1995 - project started
- 1999 - first beta
- 1999/2000 - first stable release (v 3.0.X)
- September 27, 2010 - latest stable release (v 4.0.13)
- July 1, 2014 - preview release (v 4.1.19)

Uppaal variations

Versions: Windows, Mac, Linux, 32/64 bits
Available licenses:

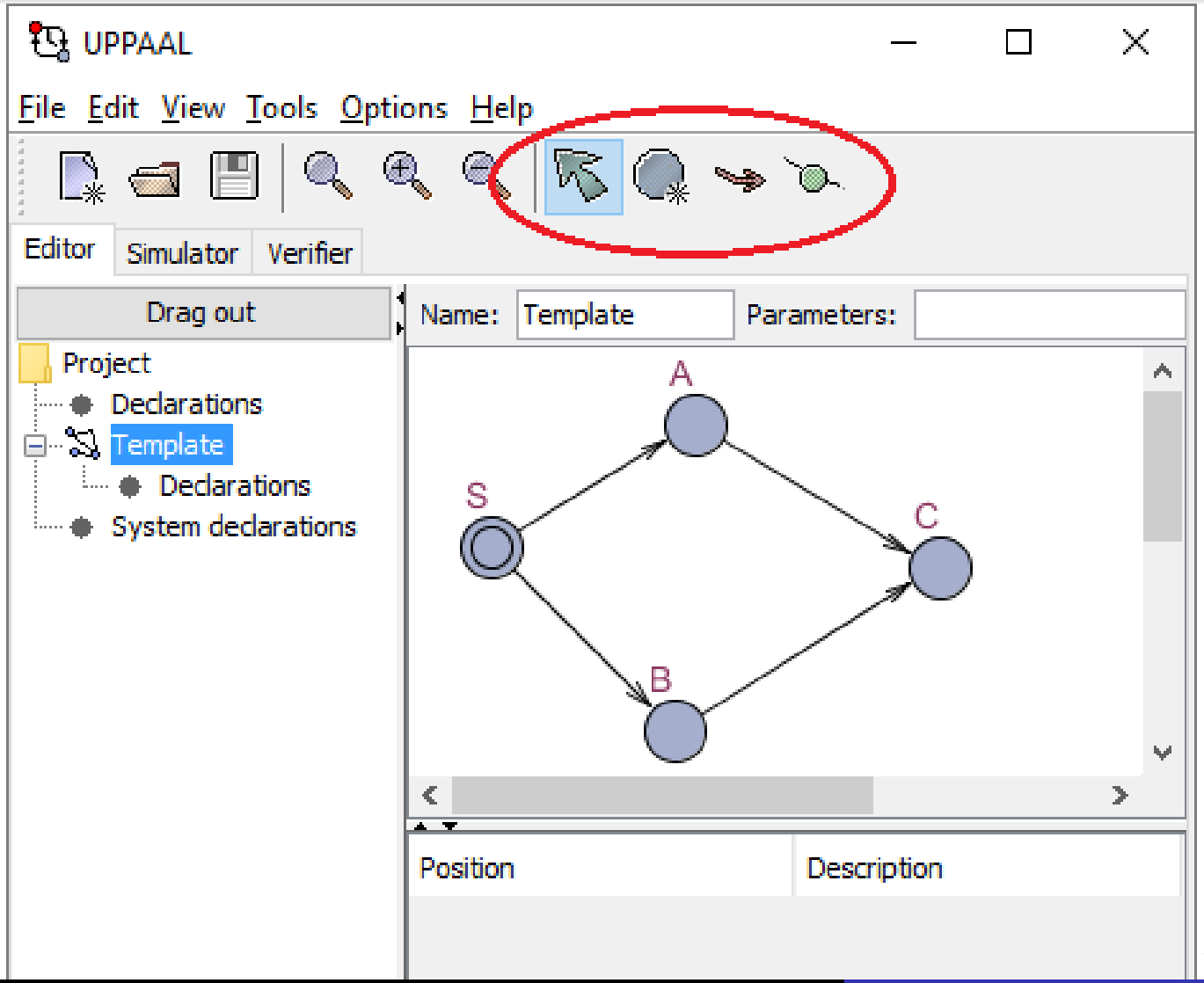
- Academic use (more info: http://www.uppaal.org/)
- Commercial use (more info: http://www.uppaal.com/)

Uppaal GUI - Main parts



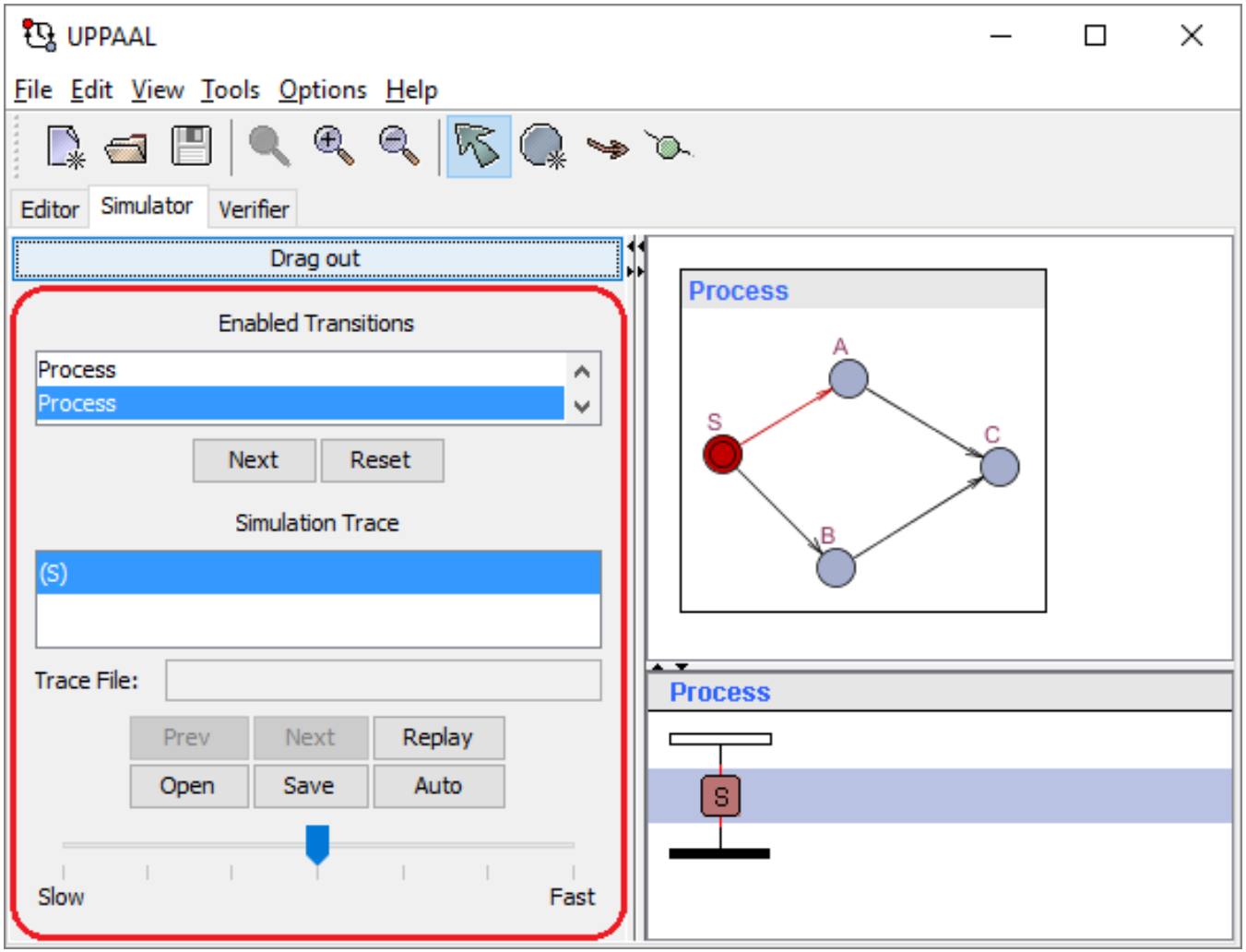
- System editor
- Simulator
- Verifier

Uppaal GUI - System editor



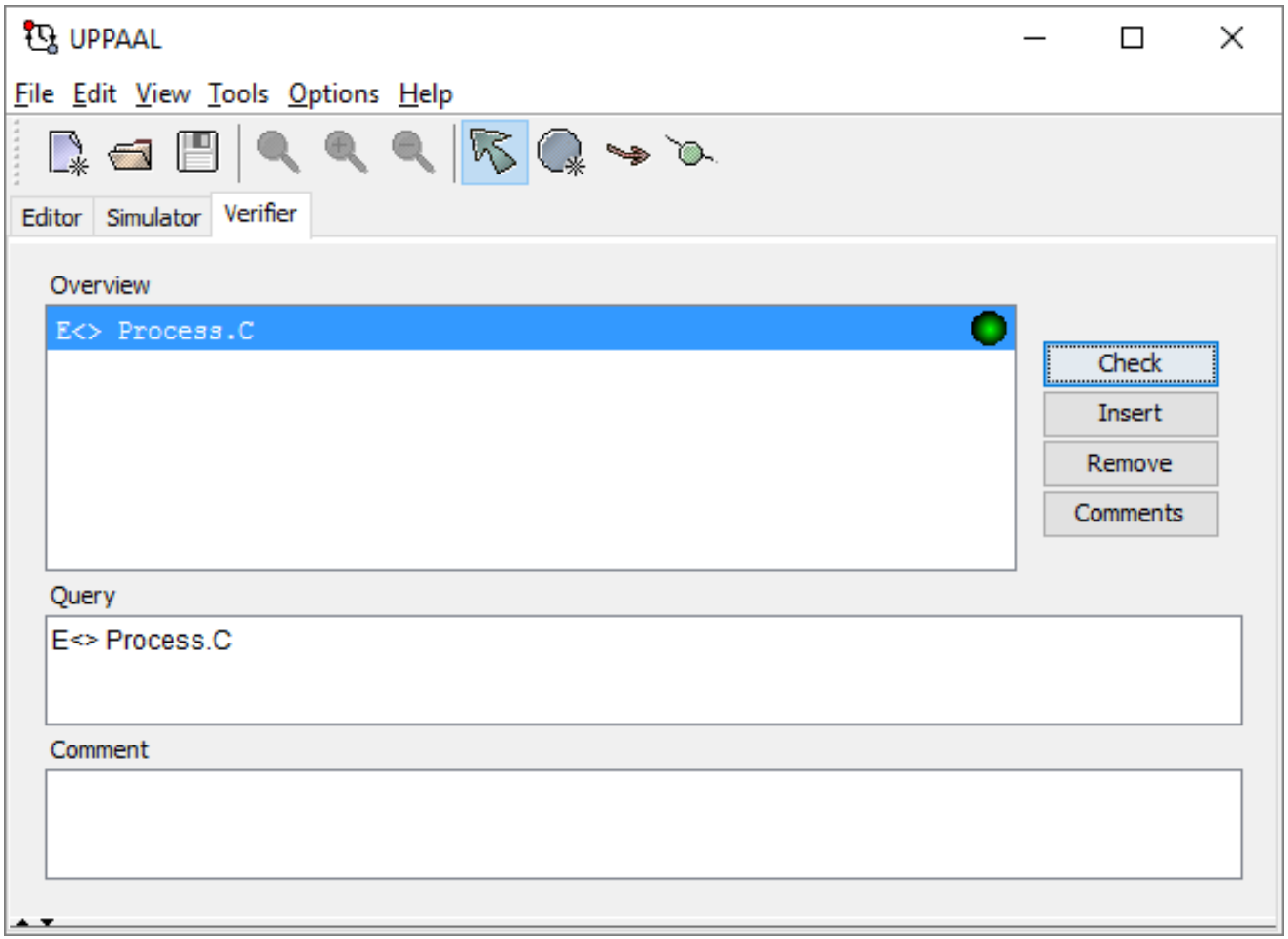
- Name: Template (defualt)
- Select
- Add location
- Add edge
- Add nail
- Syntax check (for global, local, system declarations)

Uppaal GUI - Simulator



- Select transition
- Track simulation
- Control (Prev/Next/Auto)
- Visualization

Uppaal GUI - Verifier



- Query editor
- Check query
- Overview
- Save/load

Project

All projects consist of:

- Global declaration for whole project
- Processes modelled using timed automata
- Local declaration for each process
- System description

Model/system

A Uppaal model (called system) is defined as a composition of a number of basic components (called automata or processes). Automata are diagrams with states (called locations) and transitions between states (called edges).

Process/automata

Each automaton has at most one initial location, marked by a double circle.
(!)Deadlock - is when no further transitions are possible and therefore a deadlock state has been reached.

Location/state

If a location is Urgent then this means that time can not elapse within this location, and hence a transition to another location will occur immediately.
Each automaton has at most one initial location, marked by a double circle.

Parameters

Parameters can be declared to have either call-by-value or call-by-reference semantics, that is, a template may have access to either a local copy of the argument or to the original. The syntax is taken from C++, where the identifier of a call-by-reference parameter is prefixed with an ampersend in the parameter declaration. Clocks and channels must always be call-by-reference parameters.
Uppaal does not allow data parameters for synchronization channels.
In template paraemetrs field: urgent chan &get, chan &put. When creating templates: Hammer = Tool(get_hammer, put_hammer); Mallet = Tool(get_mallet, put_mallet);

System declaration

Edge/transition

Choices for which the model does not specify how they are resolved are called nondeterministic.

Without prior specification, transitions occur instantaneously and do not take time.

Transition have fields:

- Guard
- Update

In fields we can declare arrays, Boolean variables, define new "record" types and new functions.

Channels

Uppaal does not allow data parameters for synchronization channels.
When a synchronization channel is urgent, this means that whenever a synchronization with this channel is enabled, time can not advance and a transition has to be taken immediately.

In order to model the synchronization between tools and jobbers, we use the notion of (synchronization) channels from Uppaal. Once a has been declared as a channel, transitions can be labeled with either a! or a?. This can be done by double clicking (within the Editor) transitions with the Select tool, and then writing a! or a? in the Sync field. When two automata synchronize on channel a, this means that an a! transition of one automaton occurs simultaneously with an a? transition of another automaton. An a! or a? transition can never occur on its own; a! always has to

Variables

Global/local variables

Examples:

const int $J = 10$;
$int[0, J]$ jobs; // integer with min. value 0 and max. value J

The domain of integer is always bounded. By default *int* has range $[-32768, 32768]$.

Variable assigned with a value outside of its domain generates "run-time error".

Transistion can have guards: expressions which use variables.
Every transition have guard, by defualt they return **true**

Expressions with variables

All are C-like:

- jobs++
- jobs = jobs + 1
- jobs := jobs + 1

Time and clocks

It allows to track:

- The ordering of events
- How fast some events occur

In Uppaal we can specify upper bounds on timing, using so-called invariants. When we double click the location work easy, a window appears with a field In-variant.

If we want to specify lower and upper bounds on the time that an automaton may stay in a certain location, we can do this in Uppaal using so-called clocks. A clock is a special type of variable, whose domain consists of the set of nonnegative real numbers. Just like other variables, clocks can be declared either as a global variable (which can be tested and updated by all automata) or as a local variable (which can only be used by one automaton). In the initial state, all clocks have value 0. When an automaton is waiting in a location and time elapses then the values of its clocks increase

Verification

Whereas the Verifier has an option to compute the fastest execution leading to a certain state, there is no corresponding option to compute the slowest execution.

Queries (part I)

Query is a property that may or may not hold for a given model. The Verifier can establish whether a Query is **satisfied** or **not**.
Notation:

- $A[]$ - for all reachable states...
- $E <>$ - there exists a reachable state...

Simple queries are in form of $A[]e$ and $E <> e$ where e is an expression build from **boolean combination** of **atomic propositions**.

Queries (part II)

Atomic propositions can be of the form $A.c$, for A an automaton and c a location. Such a proposition is true in a global state of the model if in this state automaton A is in location c.

Expression	**Name**	**True when...**
e && f	and	e and f evaluate to **true**
$e \mid\mid f$	or	e or f evaluate to **true**
$e == f$	equality	e and f evaluate to the same value
e *imply* f	implication	e evaluate to **false** or f evaluate to **true**
e *not* f	negation	e evaluates to **false**

Query examples

- $A[]$ not deadlock
- $E <>$ Process_1.C
- $E <>$ (Process_1.C && Process_2.C)
- $A[]$ now $>=$ 200 imply
  (Belt.end && Jobber1.begin &&Jobber2.begin)

Reasoning

(!)The tool just uses brute force to explore all the reachable global states of the model and to check for each of these states whether both jobbers are working on a hard job.

Diagnostic traces

(!)Uppaal can also provide a concrete example that illustrates why the property holds (for $E <>$ properties) or not (for $A[]$ properties, so called: *counterexample*).

(!)In the case of $E <>$ properties that do not hold, or $A[]$ properties that hold, Uppaal can only report that it exhaustively checked all the reachable states of the model and didnt find anything.

(!)We choose under *Options* the entry *Diagnostic Trace* and then select the option *Shortest*. Simulator will then show found diagnostic trace.

Saving data

We can save various data:

- Model/system - **.xml* file
- Queries - **.q* file
- Traces (binary) - **.xtr* file

- F.W. Vaandrager. A First Introduction to Uppaal. In J. Tretmans, editor. Quasimodo Handbook. To appear.