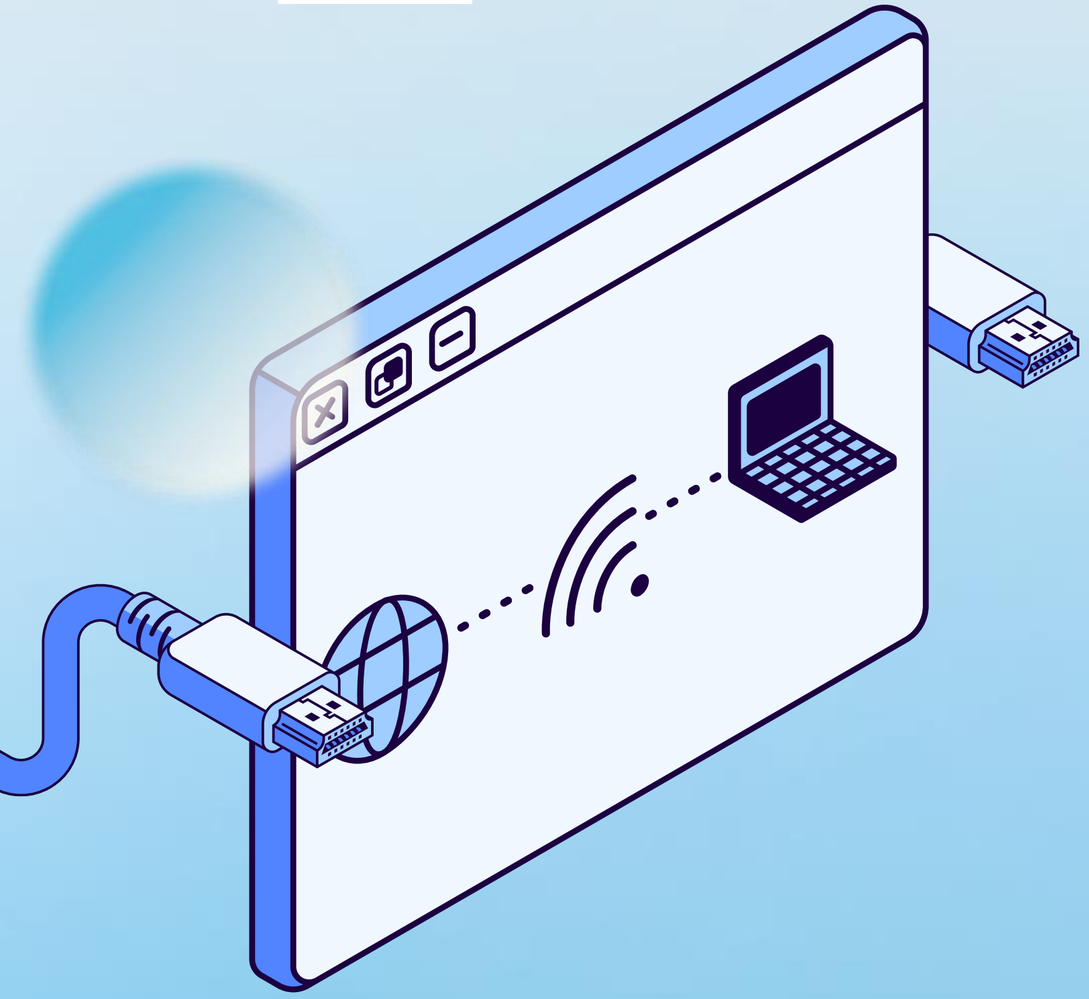
UFRPE

SBMF 2024 - 05/12/2024

A CSP semantics for UML state machines aiming at hidden formal methods verification

Diego Ferreira, **Lucas Lima**
Computer Science Department - **UFRPE**



Advantages of using Formal Models

- Risk Reduction
- Precision and Assurance
- Early Fault Detection

Disadvantages of using Formal Models

- Complexity and Expertise
- Interpretation of Results
- Scalability

Hidden Formal Methods

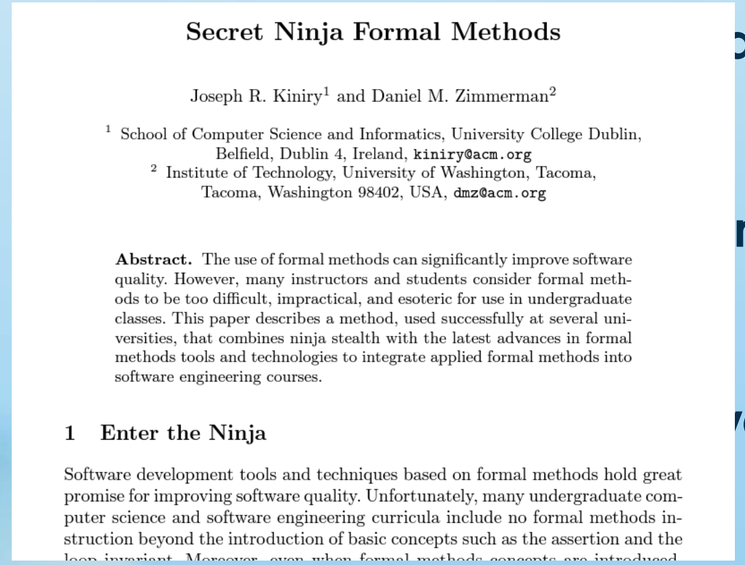
al verification

).

m the comple

4

verification.



Secret Ninja Formal Methods

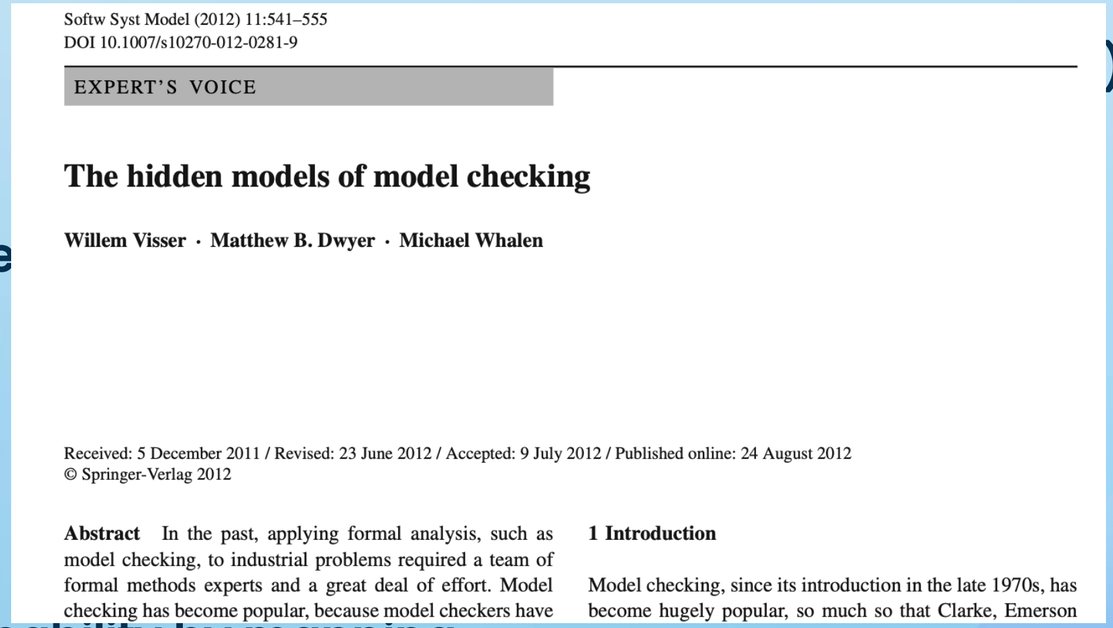
Joseph R. Kiniry[1] and Daniel M. Zimmerman[2]

[1] School of Computer Science and Informatics, University College Dublin, Belfield, Dublin 4, Ireland, kiniry@acm.org
[2] Institute of Technology, University of Washington, Tacoma, Tacoma, Washington 98402, USA, dmz@acm.org

Abstract. The use of formal methods can significantly improve software quality. However, many instructors and students consider formal methods to be too difficult, impractical, and esoteric for use in undergraduate classes. This paper describes a method, used successfully at several universities, that combines ninja stealth with the latest advances in formal methods tools and technologies to integrate applied formal methods into software engineering courses.

1 Enter the Ninja

Software development tools and techniques based on formal methods hold great promise for improving software quality. Unfortunately, many undergraduate computer science and software engineering curricula include no formal methods instruction beyond the introduction of basic concepts such as the assertion and the



Softw Syst Model (2012) 11:541–555
DOI 10.1007/s10270-012-0281-9

EXPERT'S VOICE

The hidden models of model checking

Willem Visser · Matthew B. Dwyer · Michael Whalen

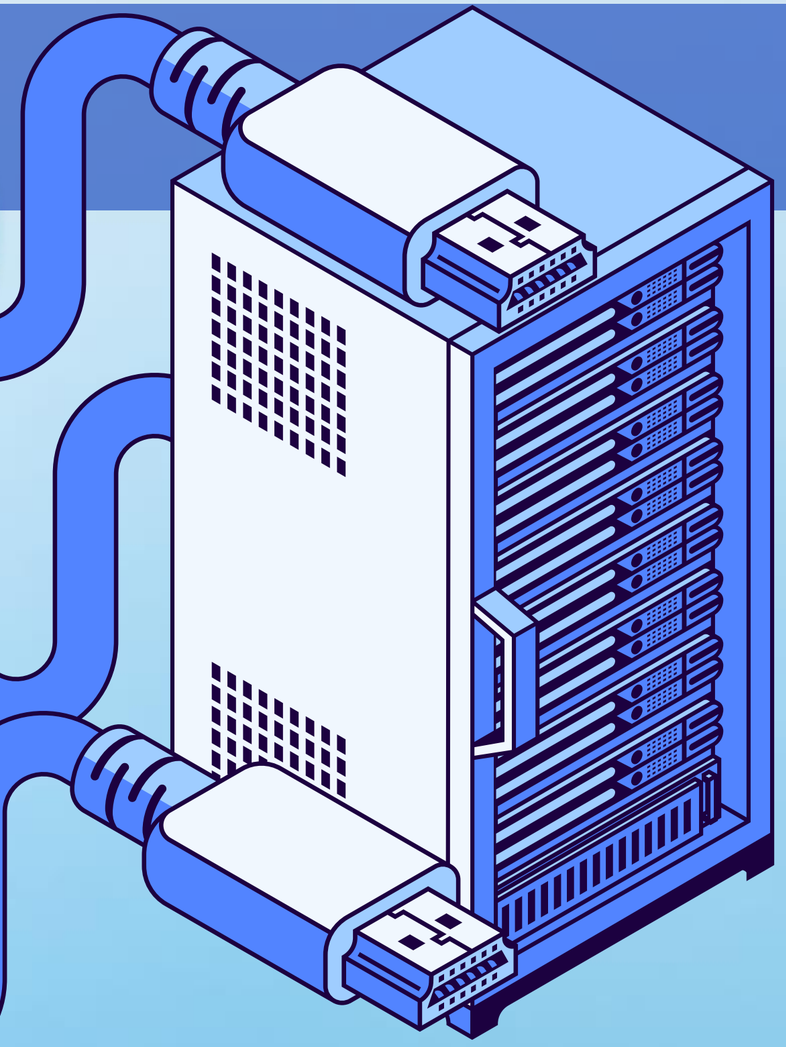
Received: 5 December 2011 / Revised: 23 June 2012 / Accepted: 9 July 2012 / Published online: 24 August 2012
© Springer-Verlag 2012

Abstract In the past, applying formal analysis, such as model checking, to industrial problems required a team of formal methods experts and a great deal of effort. Model checking has become popular, because model checkers have

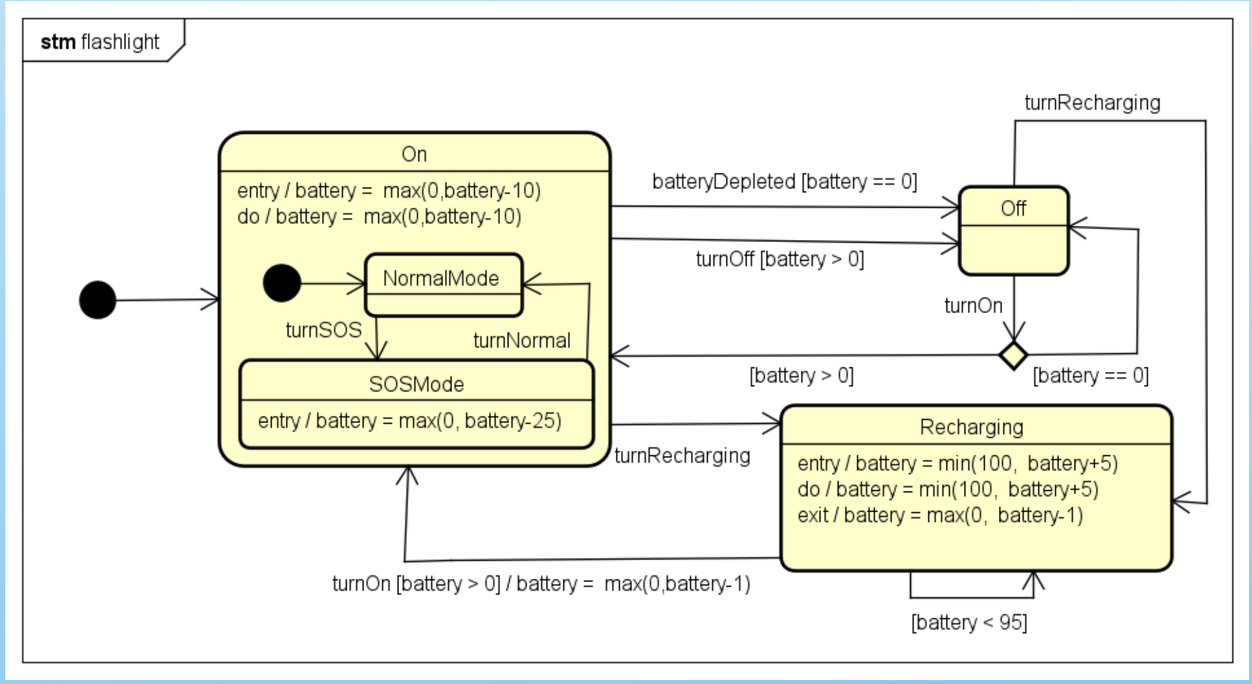
1 Introduction

Model checking, since its introduction in the late 1970s, has become hugely popular, so much so that Clarke, Emerson

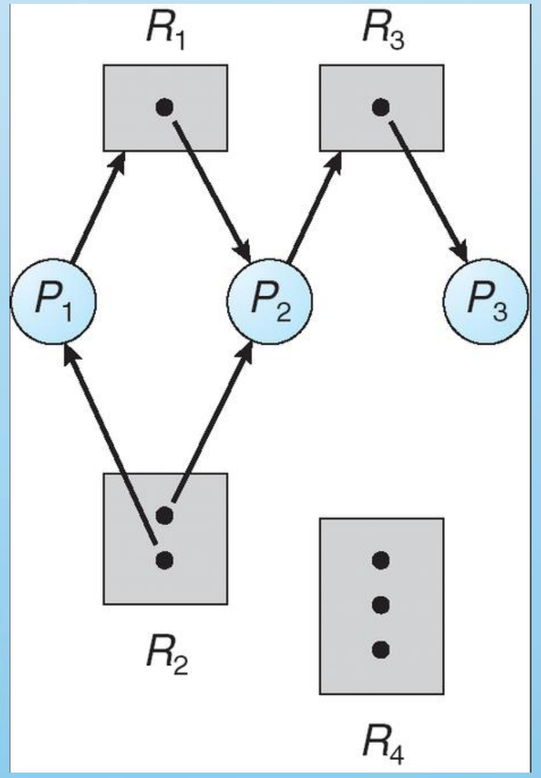
- Enables the possibility of traceability by mapping counterexamples back to diagrams.

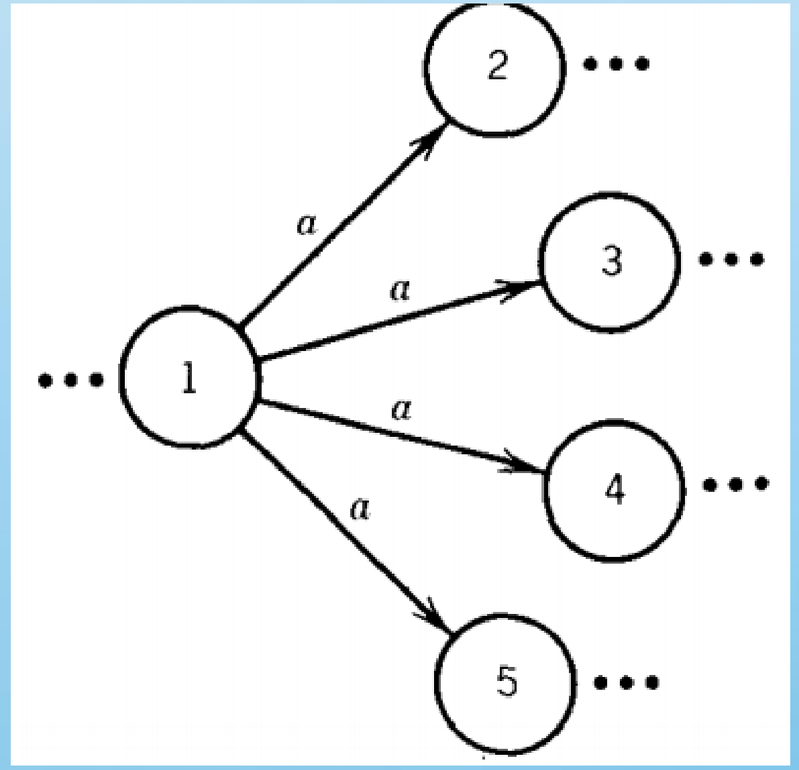


State Machine Diagram

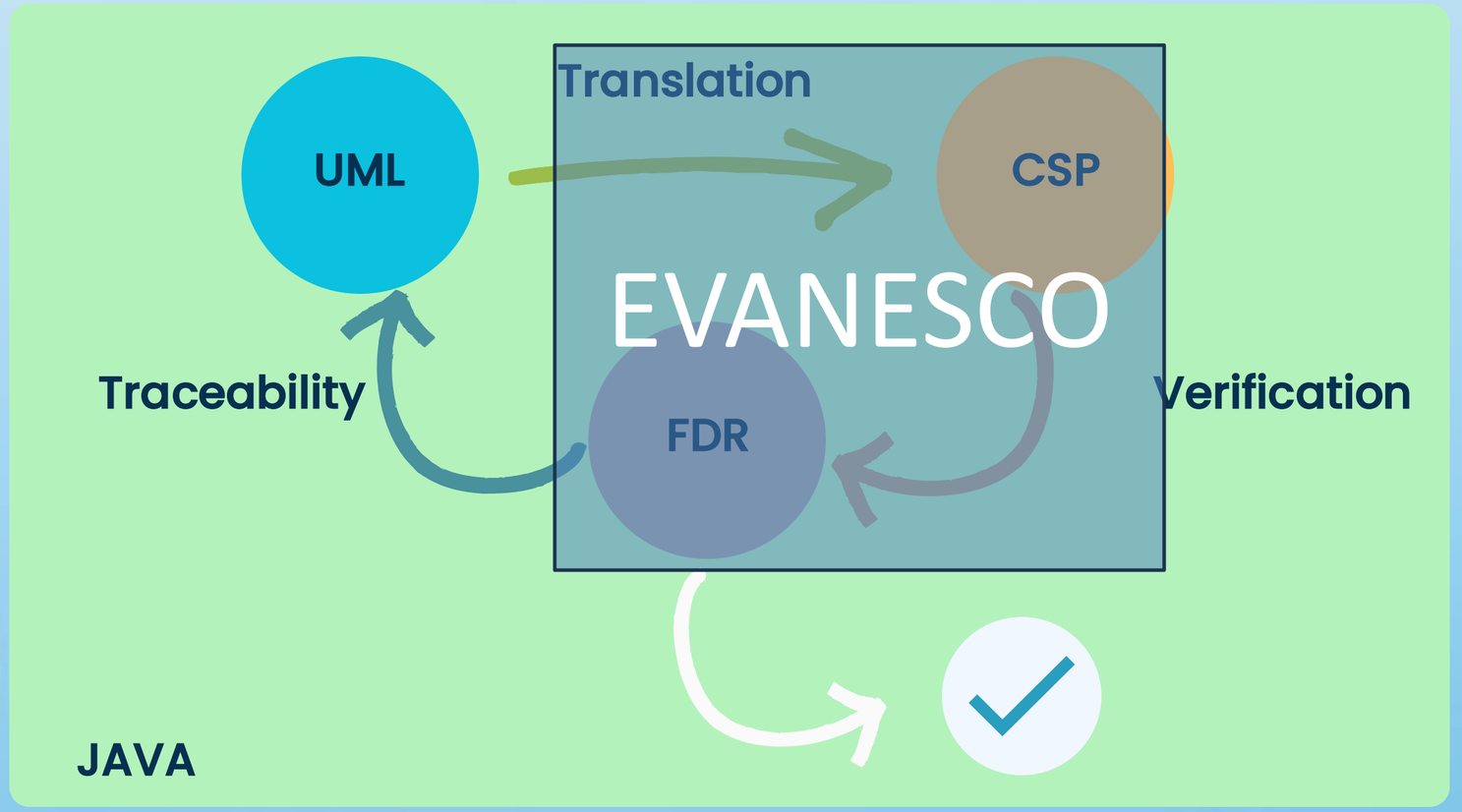


Deadlock and NonDeterminsm





Overview of the approach



Why CSP?

- Expressiveness of the language
- Compositional Operators
- Mature model checker (FDR)
- Established refinement theory

A glimpse of CSP

```
NAT = {0..MAX}
MAX = 5
```
Types and Values

```
channel put, get: NAT
```
Channel declaration

PROCESS

```
Buffer(b) = ( length(b) < MAX & put?x -> Buffer(b^[x]) )
            []
             ( length(b) > 0 & get!(hd b) -> Buffer(tail(b)))
```
External Choice

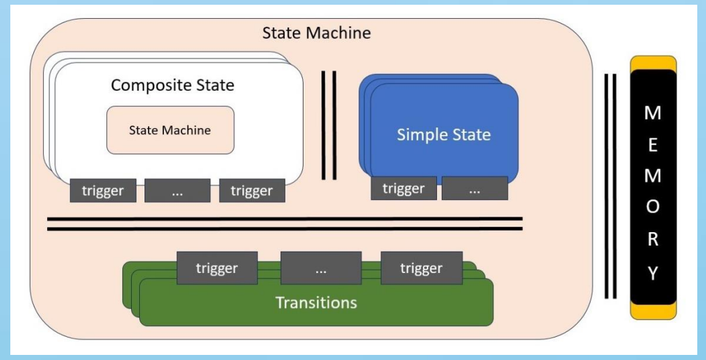
```
Producer = put!1 -> Producer
Consumer = get?x -> Consumer
System = (Buffer(<>) [|{|put,get|}|] (Producer ||| Consumer))
```
Interleaving

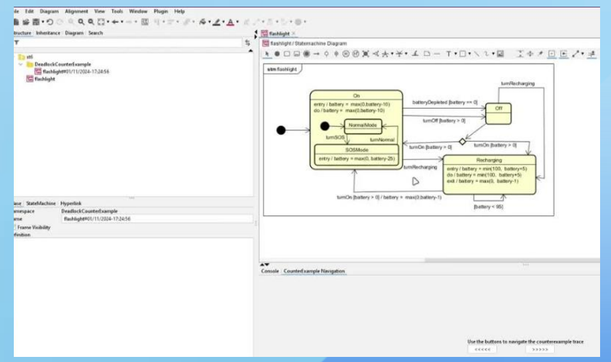
Synchronized Parallelism

Research Project

Define a formal semantic framework for UML state machines, translating them into CSP for automated verification.

Main Contributions:

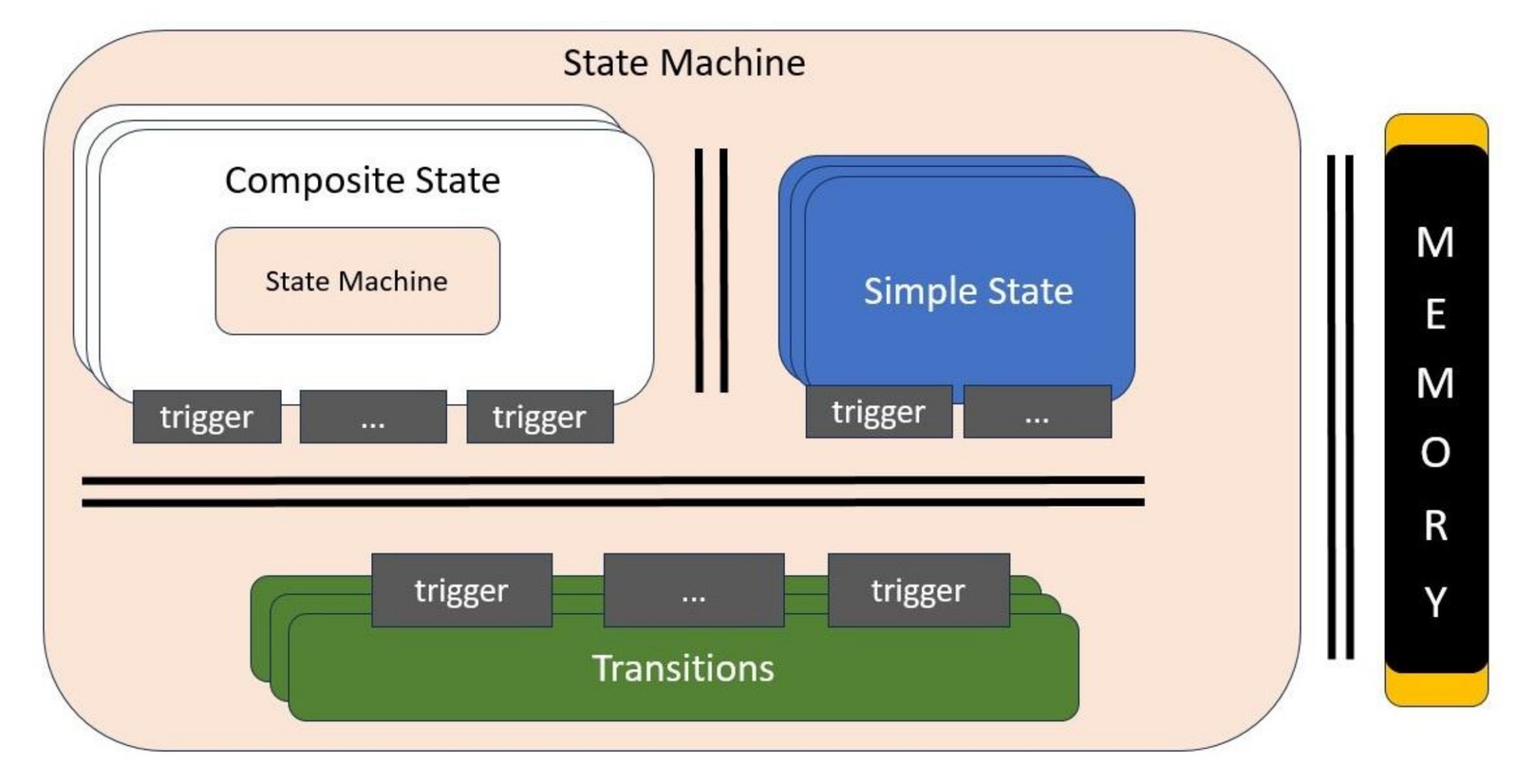




State Machine Diagram Semantics

UFRPE

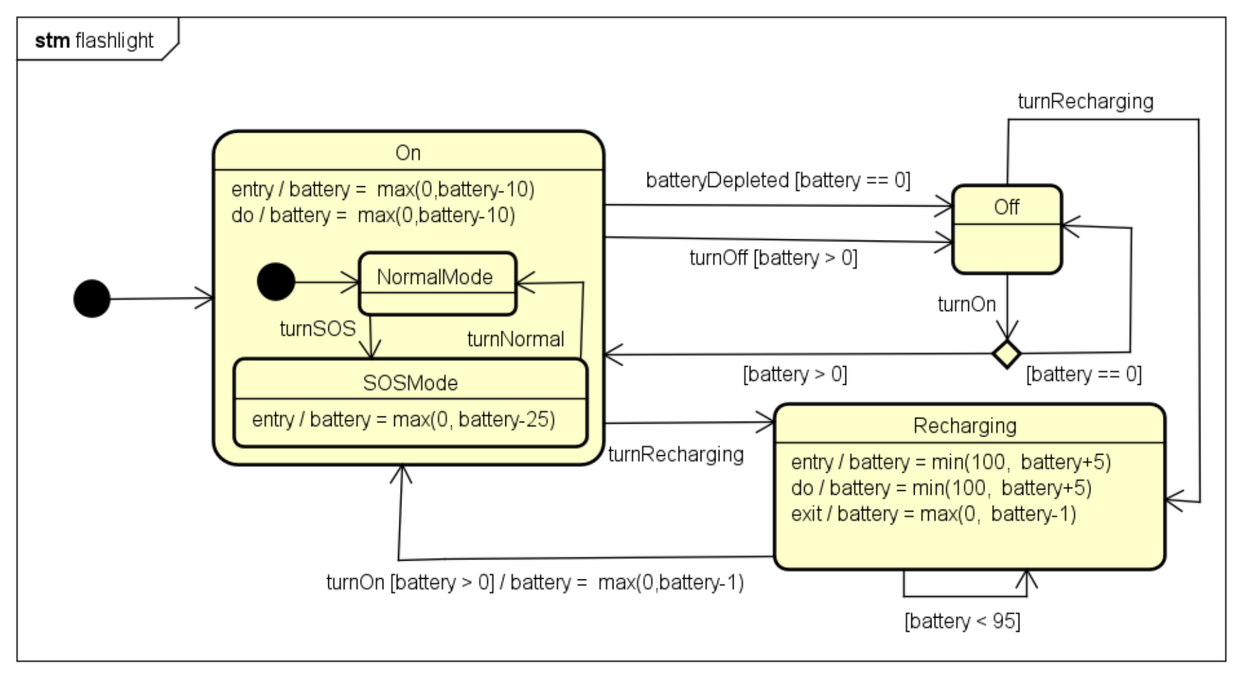
SBMF 2024

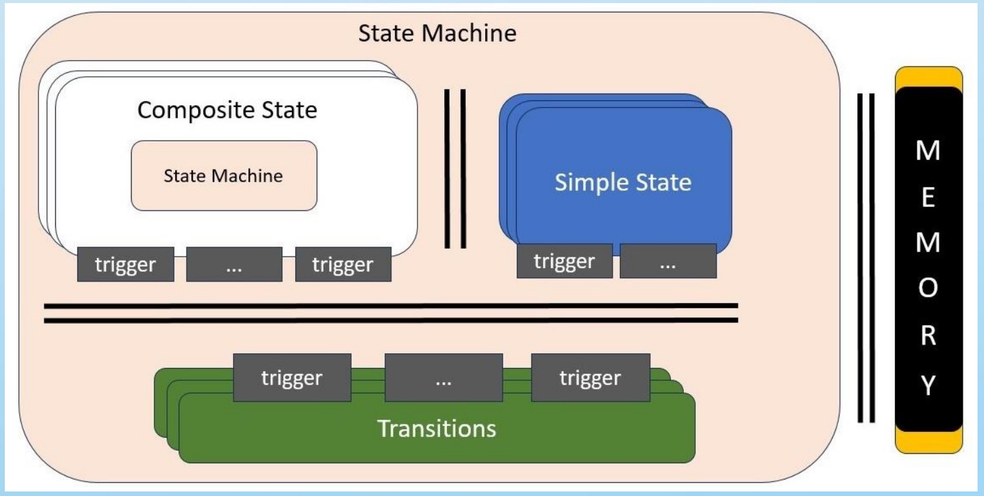


Parser – StartSync Controller

UFRPE

SBMF 2024





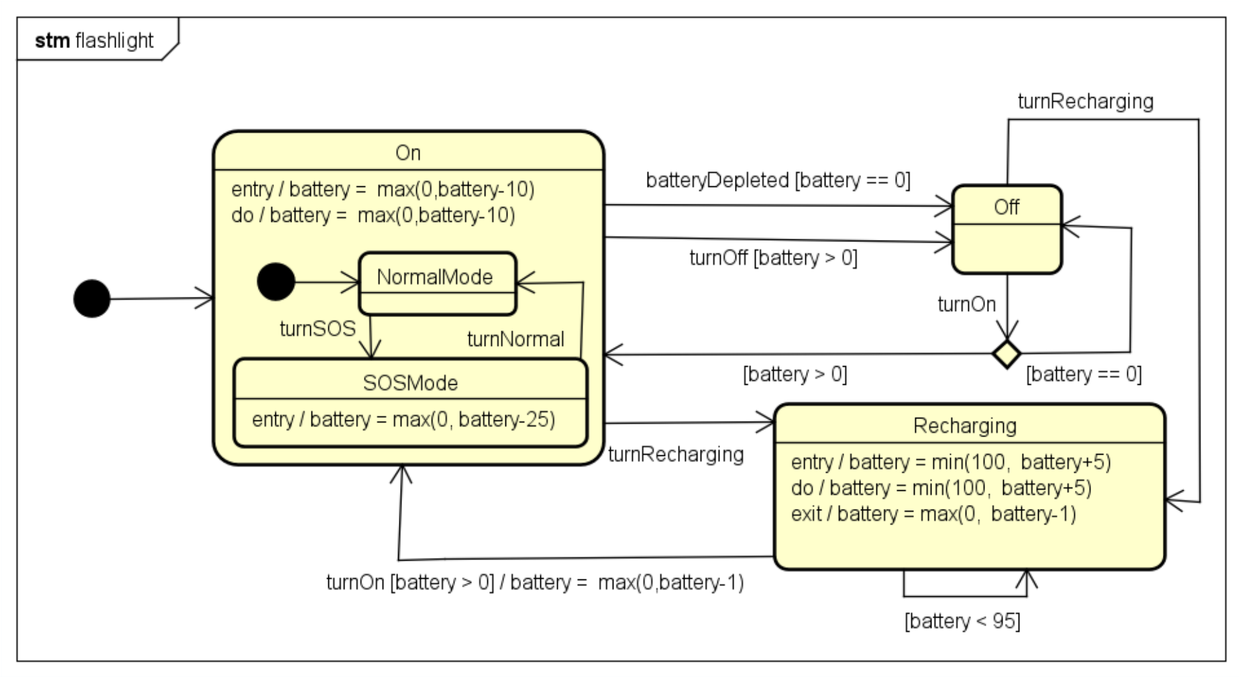
$$States\_id = On \underset{end}{\parallel} Off \underset{end}{\parallel} Recharging \underset{end}{\parallel} Choice$$

$$Transitions\_id = Tr\_id\_1;Transitions\_id \,\Box\, \ldots \,\Box\, Tr\_id\_n;Transitions\_id \,\Box\, end \rightarrow SKIP$$

$$StartSync\_id = States\_id\ [[interrupt.On <- batteryDepleted, interrupt.On <- turnOff, \ldots, interrupt.Off <- turnRecharging]] \underset{triggers,events,end}{\parallel} Transitions\_id \underset{triggers,actions,end}{\parallel} MEMORY(30)$$

Parser - Memory

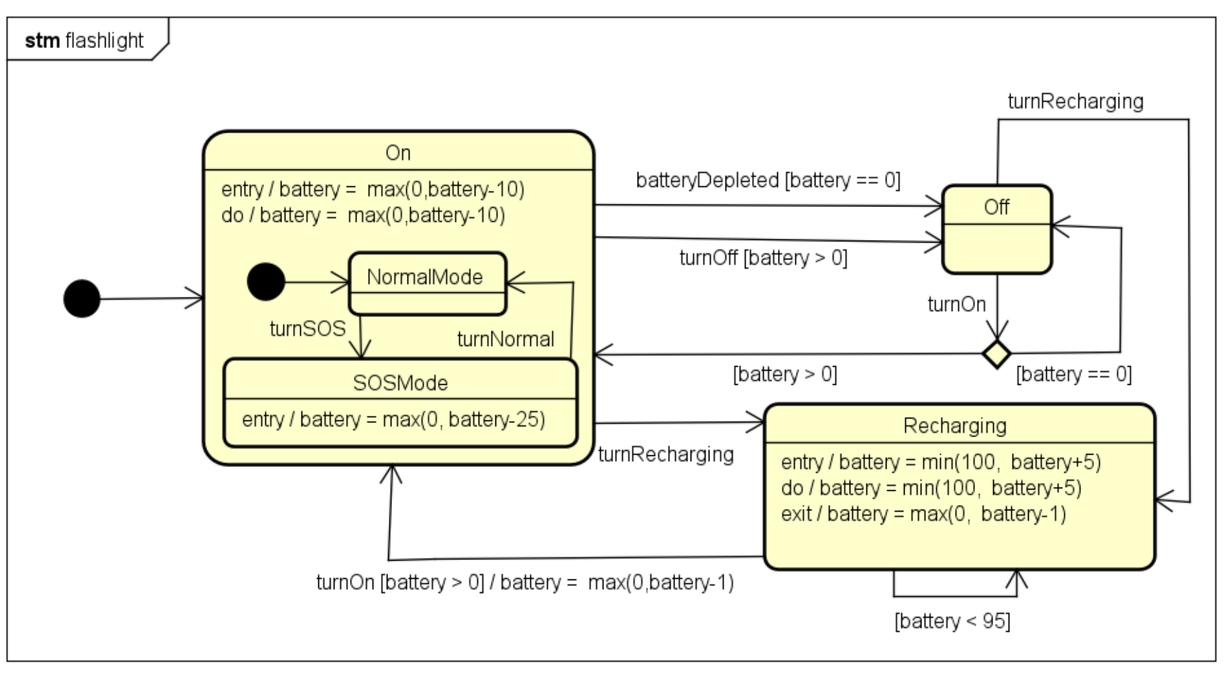
SBMF 2024



$$
\begin{aligned}
&\texttt{MEMORY(battery)} = (\texttt{end} \rightarrow \texttt{SKIP}) \;\square\; \texttt{get\_battery!battery} \rightarrow \texttt{MEMORY(battery)} \\
&\square\; \texttt{set\_battery?y} \rightarrow \texttt{MEMORY(y)} \\
&\square\; (\texttt{battery>0}) \;\&\; \texttt{turnOff} \rightarrow \texttt{MEMORY(battery)} \\
&\square\; (\texttt{battery==0}) \;\&\; \texttt{batteryDepleted} \rightarrow \texttt{MEMORY(battery)} \\
&\ldots \\
&\square\; (\texttt{battery>0}) \;\&\; \texttt{turnOn} \rightarrow \texttt{MEMORY(battery)}
\end{aligned}
$$

Parser – Internal Actions

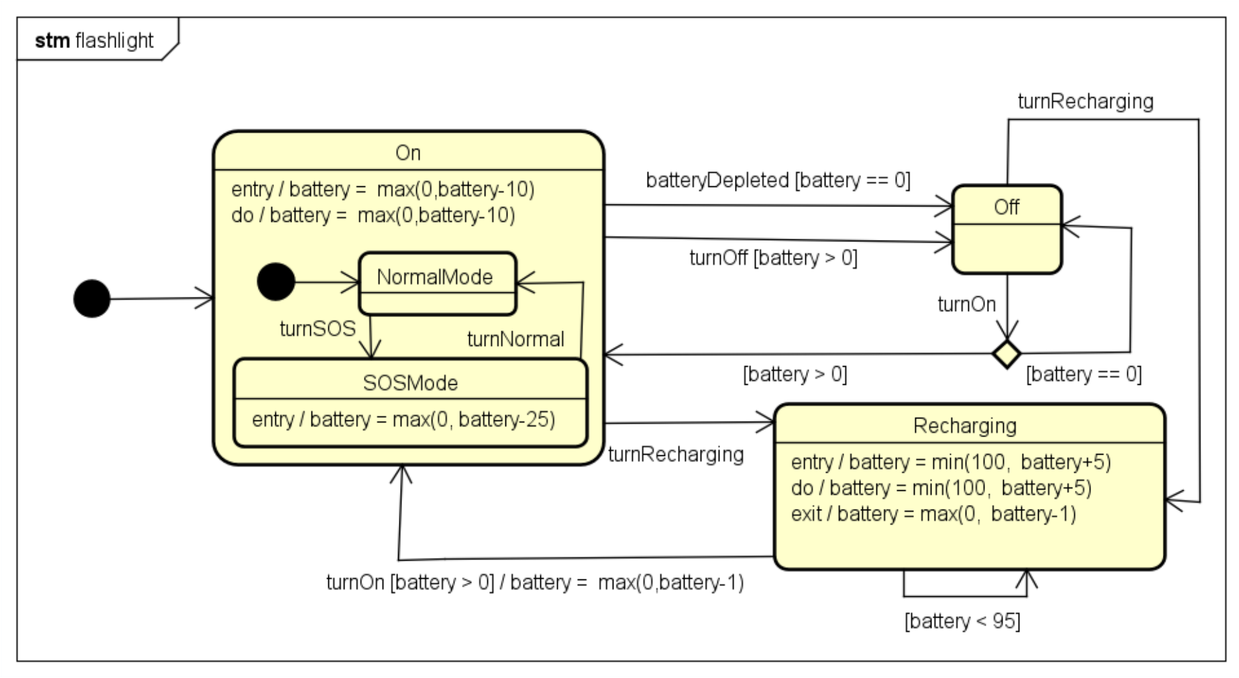
SBMF 2024




```
EntryProc(Recharging) = entry.Recharging → ActionBehaviourEntry;
  State_Recharging_Do
DoProc(Recharging) = do.Recharging → ActionBehaviourDo △
  interrupt.Recharging → SKIP
ExitProc(Recharging) = exit.Recharging → ActionBehaviourExit;
  exited.Recharging → State_Recharging
```

Parser – Simple State

SBMF 2024

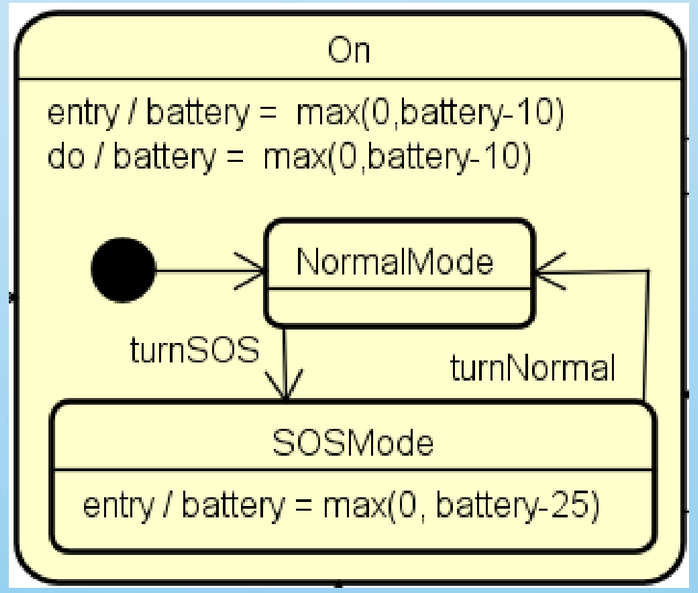


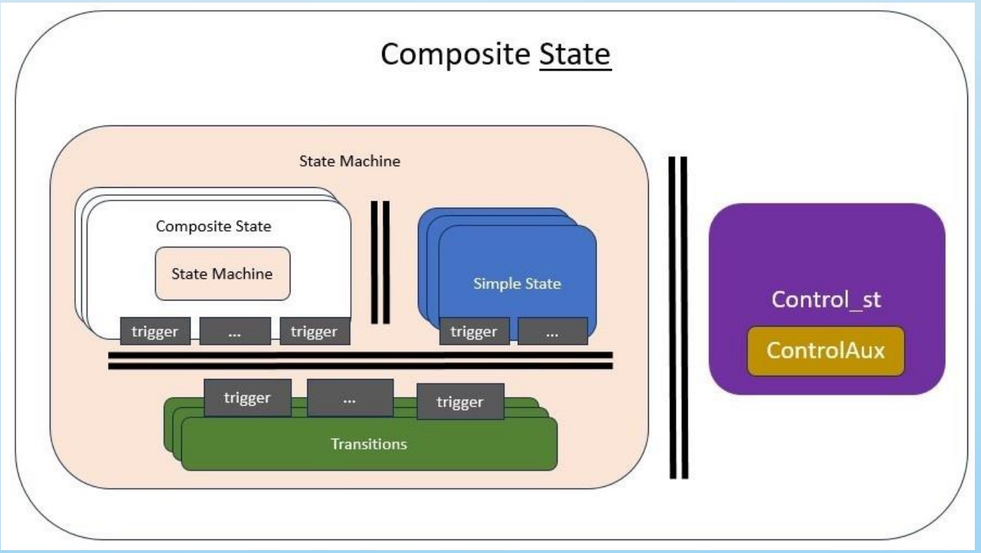

```
State_Recharging = ((enter.Recharging
→ State_Recharging_Entry) □ (end → SKIP)
State_Recharging_Entry = EntryProc(Recharging)
State_Recharging_Do = (DoProc(Recharging)); ExitProc(Recharging)
```

Parser - Composite State

UFRPE

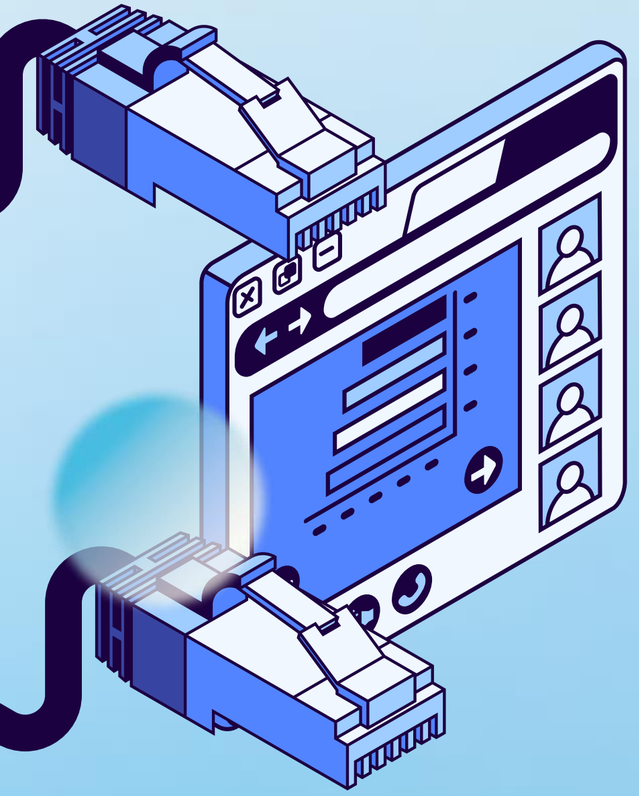
SBMF 2024

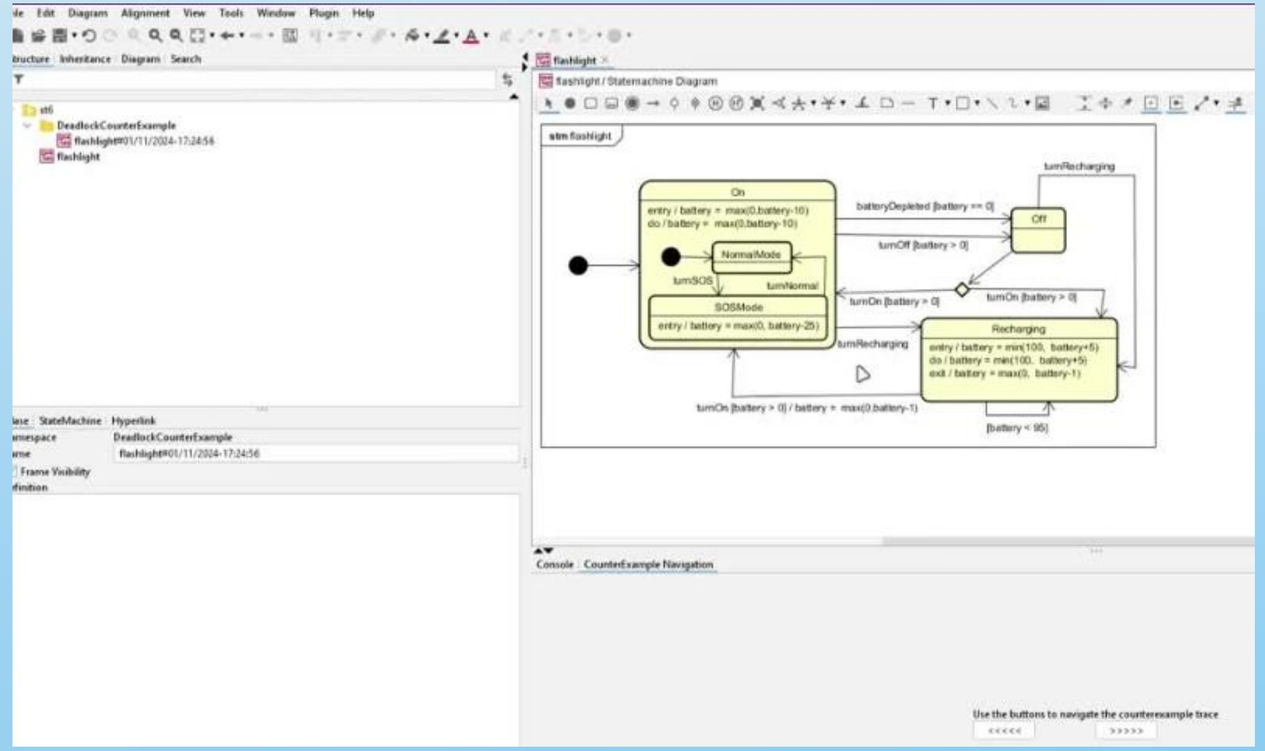





```
CtrlAux(s) = #s>0 & head(s) → CtrlAux(tail(s)) □ #s==0 & SKIP
Control_On_CompositeState(s) = #s==0 & enter.On →
  Control_On_CompositeState(<exit.On, exited.On>^s)
□ #s<n & enter?state:{NormalMode, SOSMode} →
   Control_On_CompositeState(<exit.state, exited.state>^s)
□ #s>0 & head(s) → Control_On_CompositeState(tail(s))
□ #s<n & interrupt.On → ControlAux(s);
   Control_On_CompositeState(<>)) □  (end → SKIP)
```

Tool Support

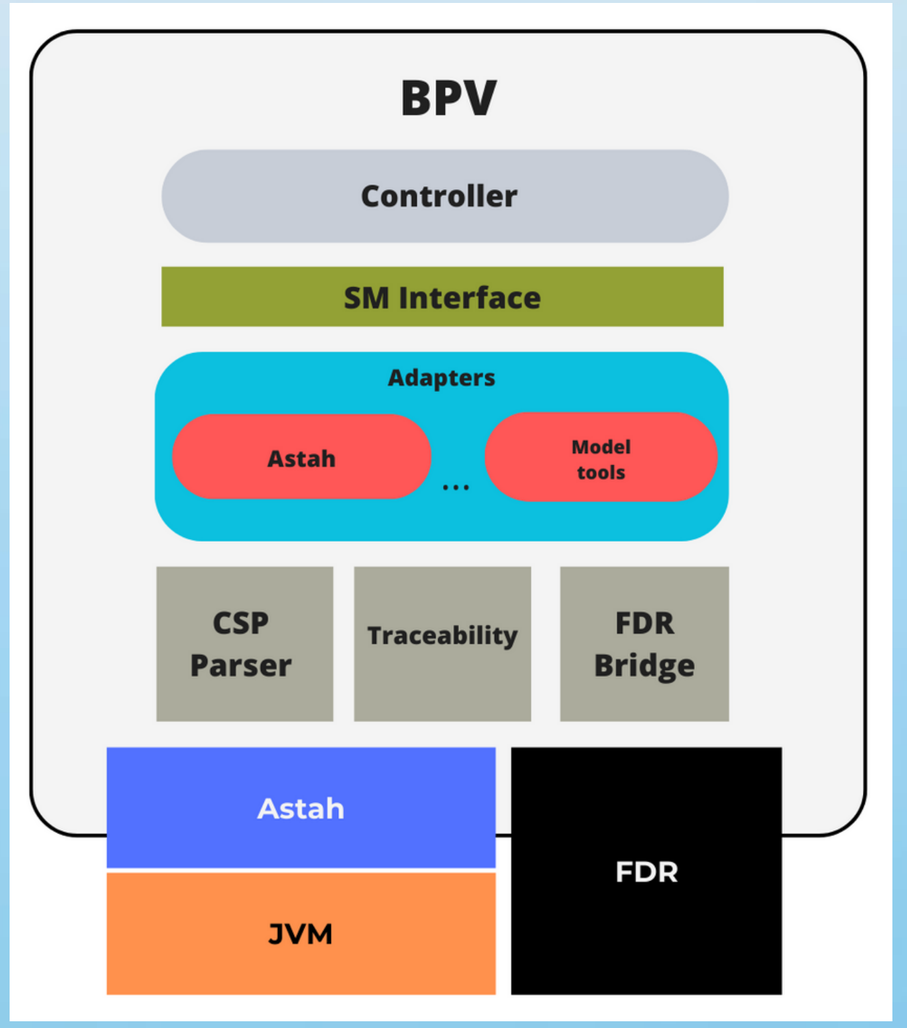


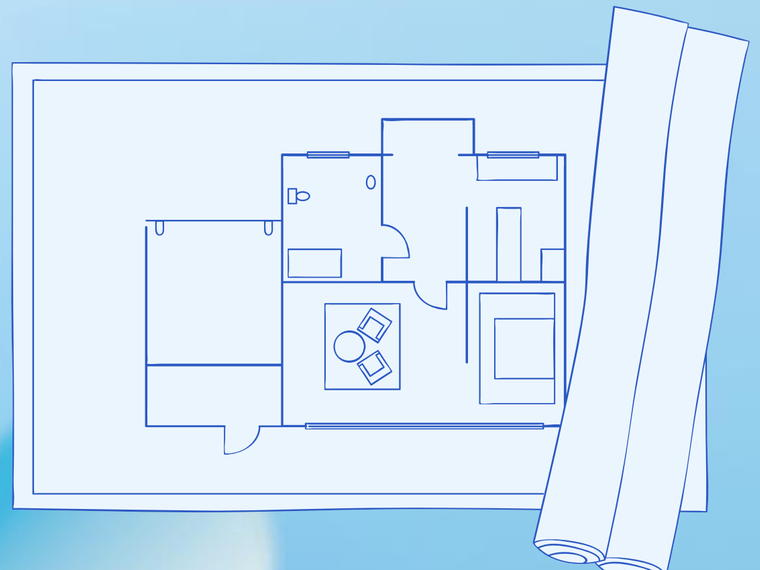


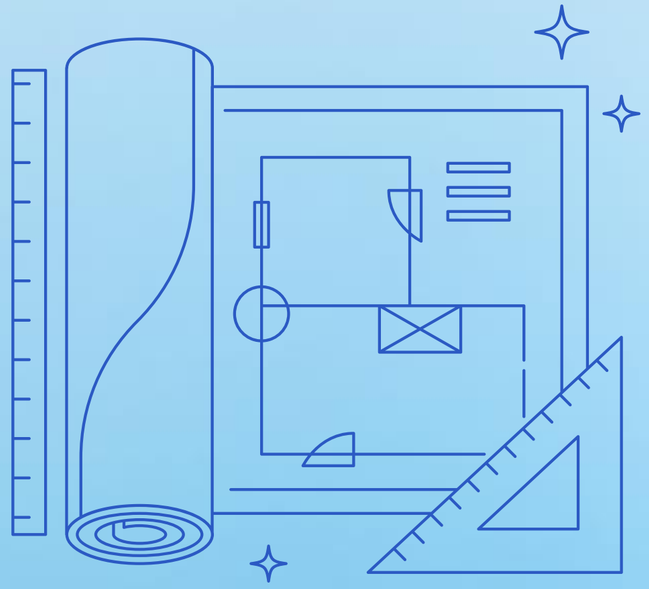
Architecture

UFRPE

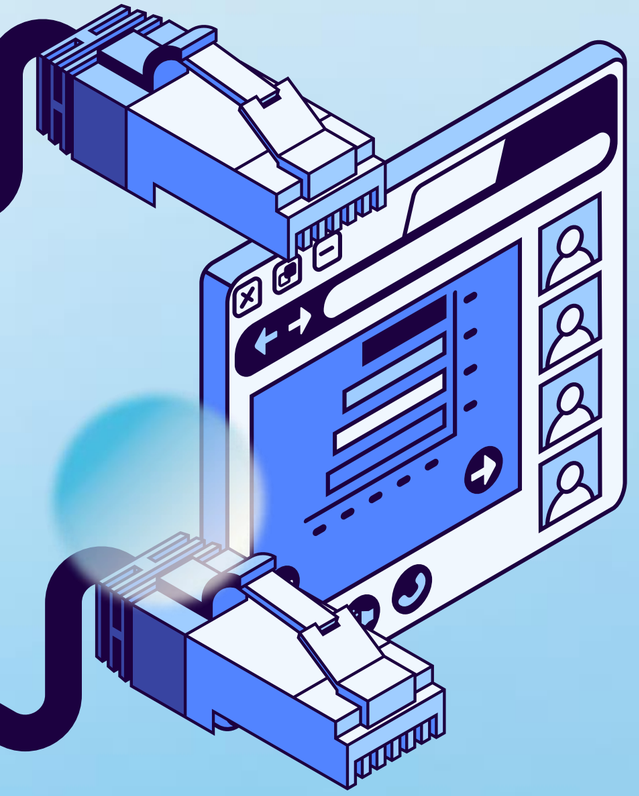
SBMF 2024

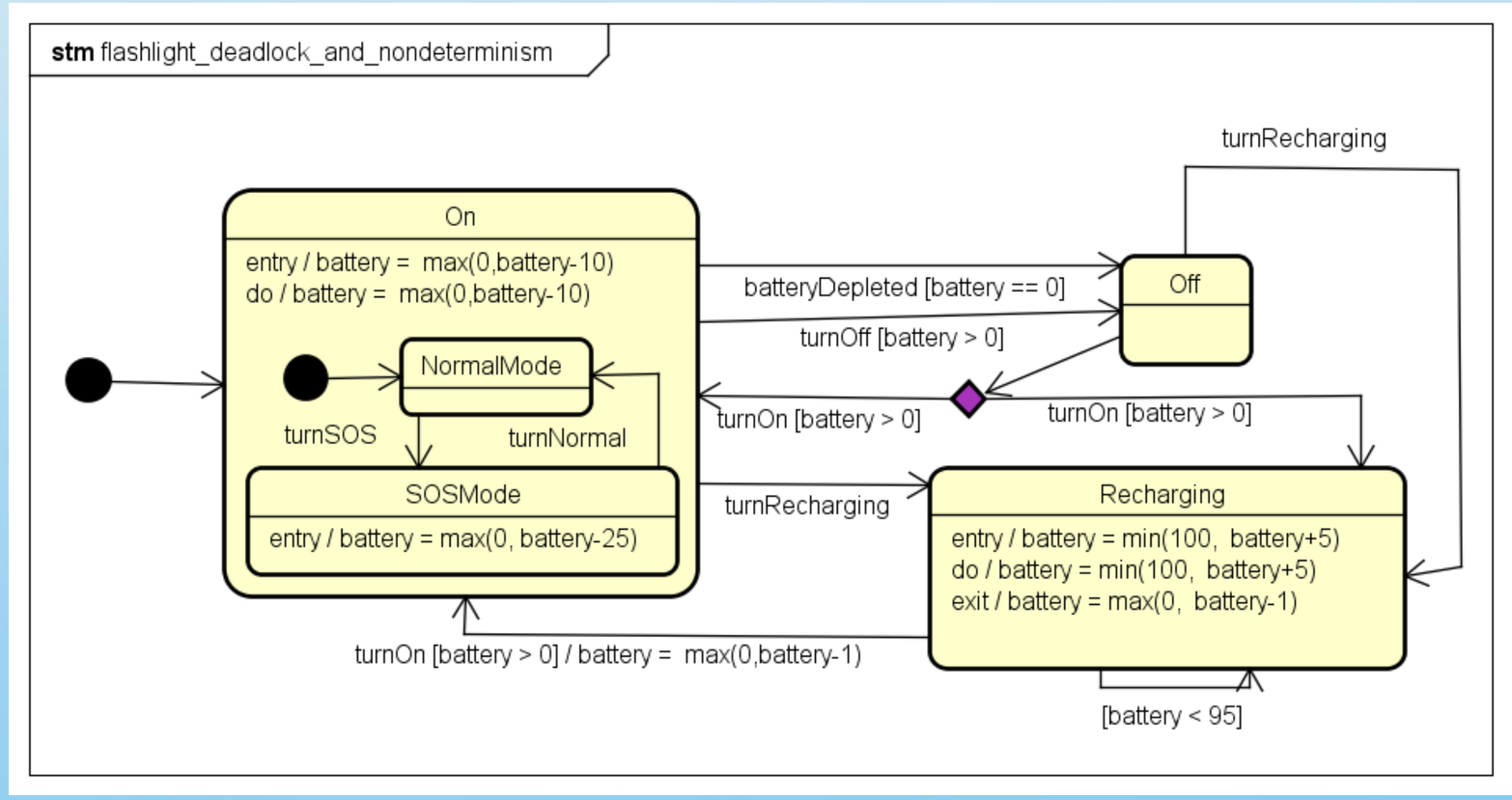




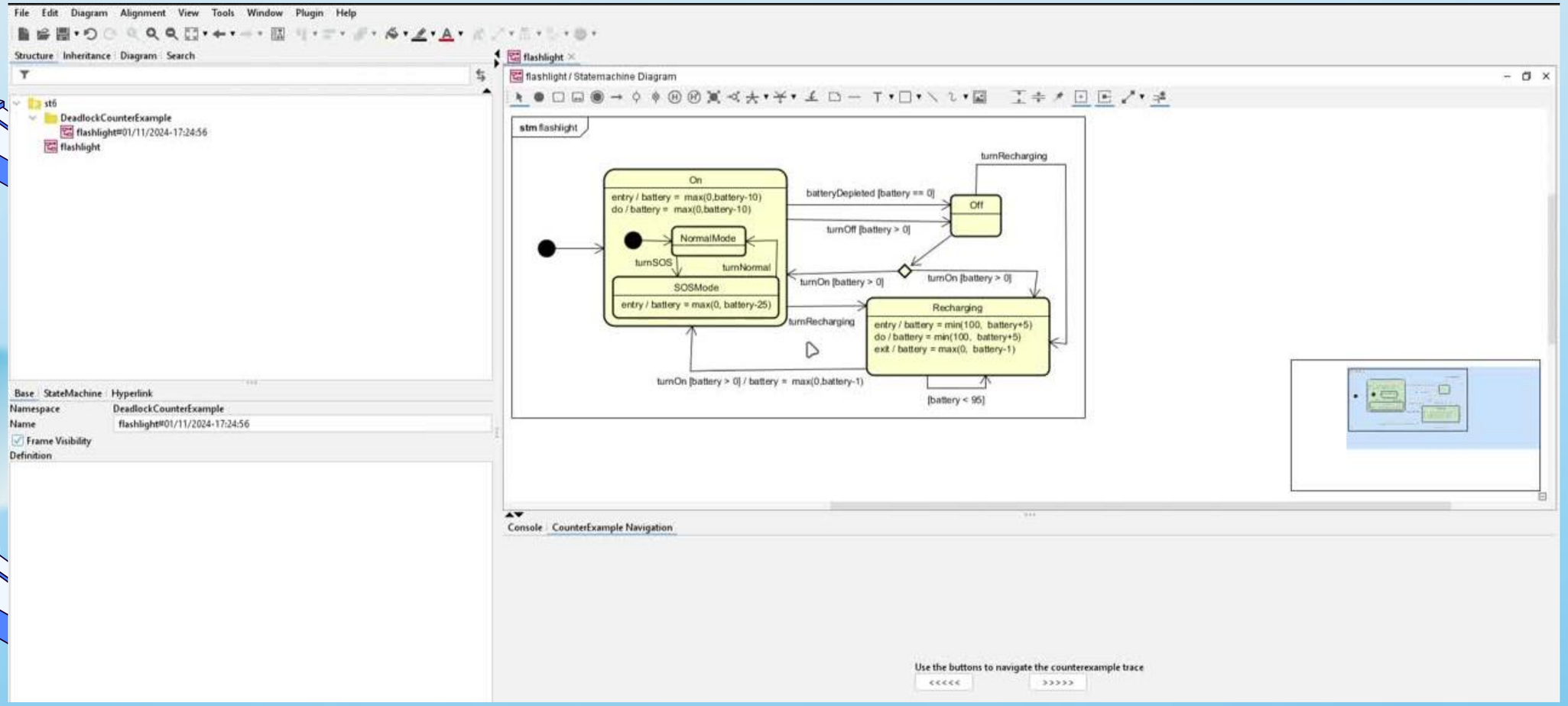


Counter Example





Demonstration



Related Work

	Internal Actions	Composite State	Memory	Automated Translation	Hidden Verification	Traceability	Formalism	Purpose
Börger et al.	✓	✗	✗	✗	✗	✗	ASM	Semantic definition
Djaaboub et al.	✓	✓	✗	✓	✗	✗	LOTOS	Automated translation
Graics et al.	✗	✓	✓	✓	✓	✓	Gamma	Reachability properties
Miyazawa et al.	✓	✓	✓	✓	✗	✗	CSP	Classical and User-defined properties
Ng et. al	✓	✓	✗	✓	✗	✗	CSP	Classical and User-defined properties
Zhang et al.	✓	✓	✗	✓	✗	✗	CSP#	Safety and liveness properties
Our Work	✓	✓	✓	✓	✓	✓	CSP	Deadlock and nondeterminism

Next Steps

Extend the Framework

Enhance the framework to support additional UML diagram types and improve scalability for verification processes.

Explore Complex Systems

Investigate verification methods for larger, more complex systems to broaden the framework's applicability.

Integrate state machines with other diagrams

Consider broader and realistic scenarios where behaviour is described using a combination of several diagrams.

Thank You!

Questions?

Contact:
diego.pires@ufrpe.br
lucas.albertins@ufrpe.br