UC-PACT: Universal Composability for Preventing Adversarial Composition Techniques

Tool Overview

Mike Clark
harden@riversideresearch.org

12 August 2025

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Naval Information Warfare Center Pacific (NIWC Pacific) under N66001-22-C-4020.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA and NIWC Pacific.

Overview

- Released tools to make your life easier in using the UC DSL
- **UC-PACT WebApp**: UC model designer GUI to make it easier to start designing UC models
- **ucdsl-tools-docker**: some dockerfiles that make using the uc-dsl typechecker and interpreter easier (including setting up Emacs)
- Breakout session on webapp (Wednesday, 1:30pm)

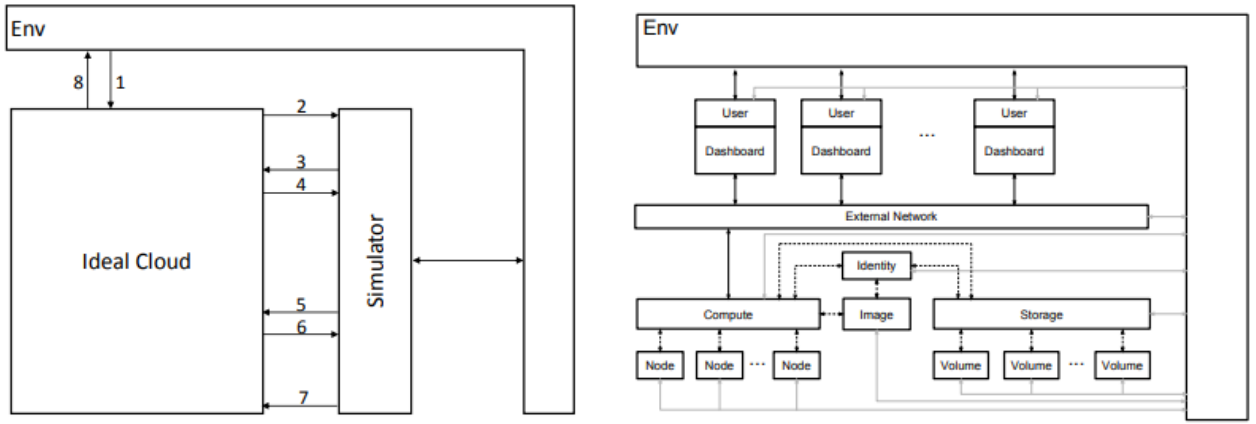
UC-PACT WebApp

ucdsl-tools-docker

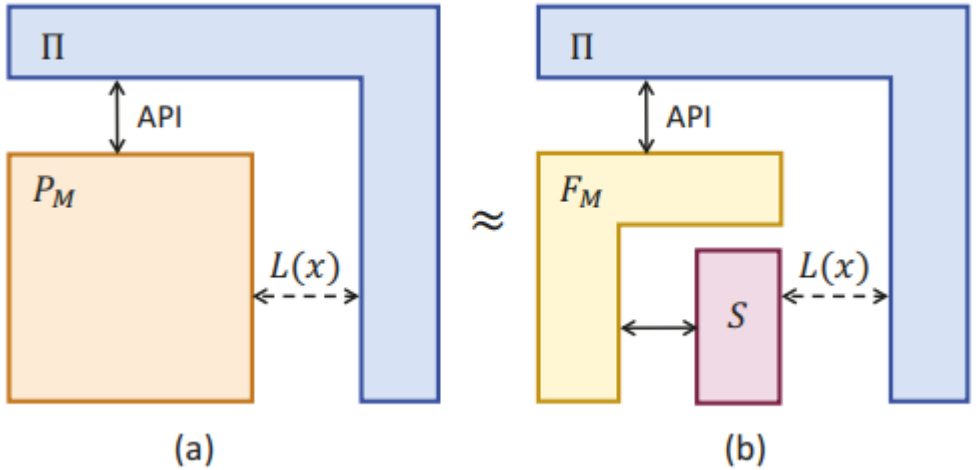
BOSTON UNIVERSITY

RIVERSIDE RESEARCH

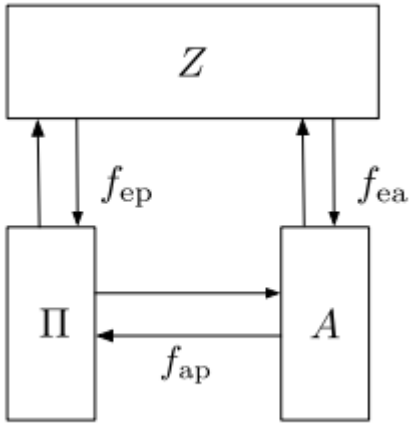
Motivating the problem



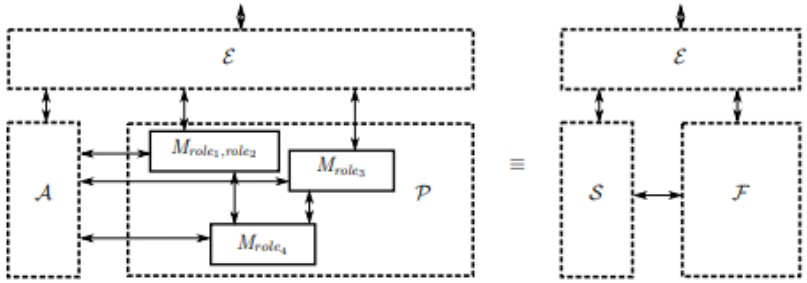
On the universally composable security of OpenStack, Hogan, et al.



Using Universal Composition to Design and Analyze Secure Complex Hardware Systems, Canetti, et al.

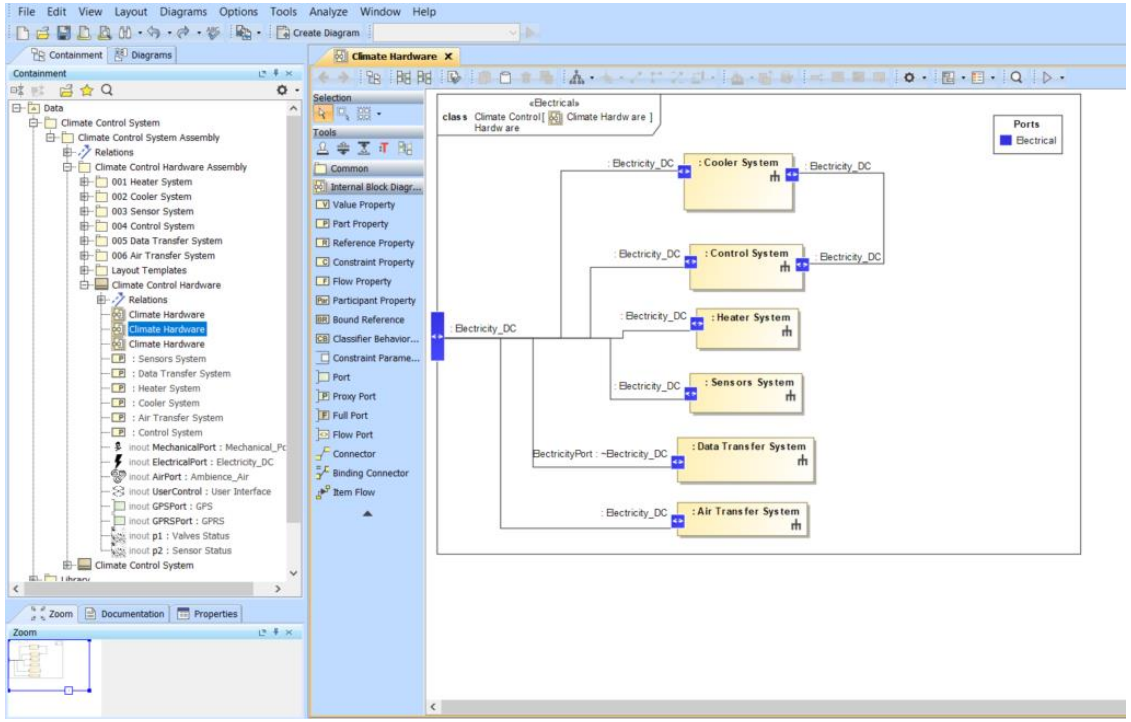


GNUC: A New Universal Composability Framework, Hofheinz, et al.

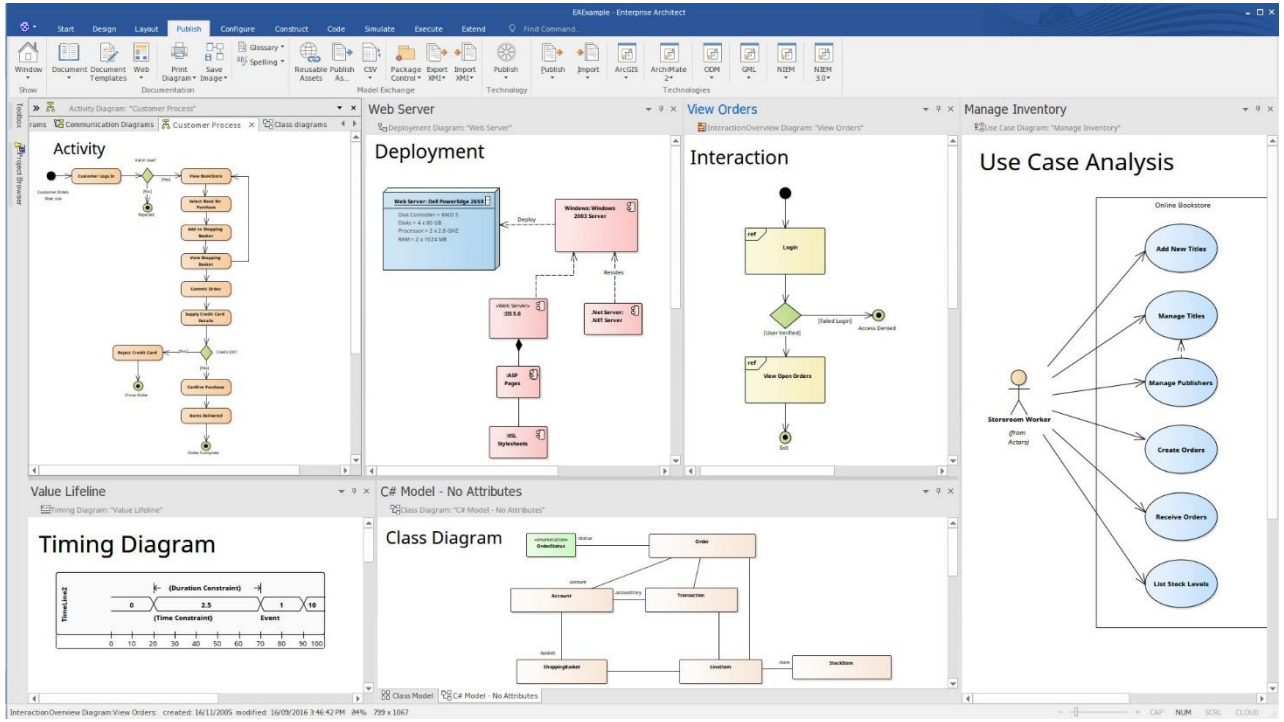


iUC: Flexible Universal Composability Made Simple, Camenisch, et al.

Motivating the problem

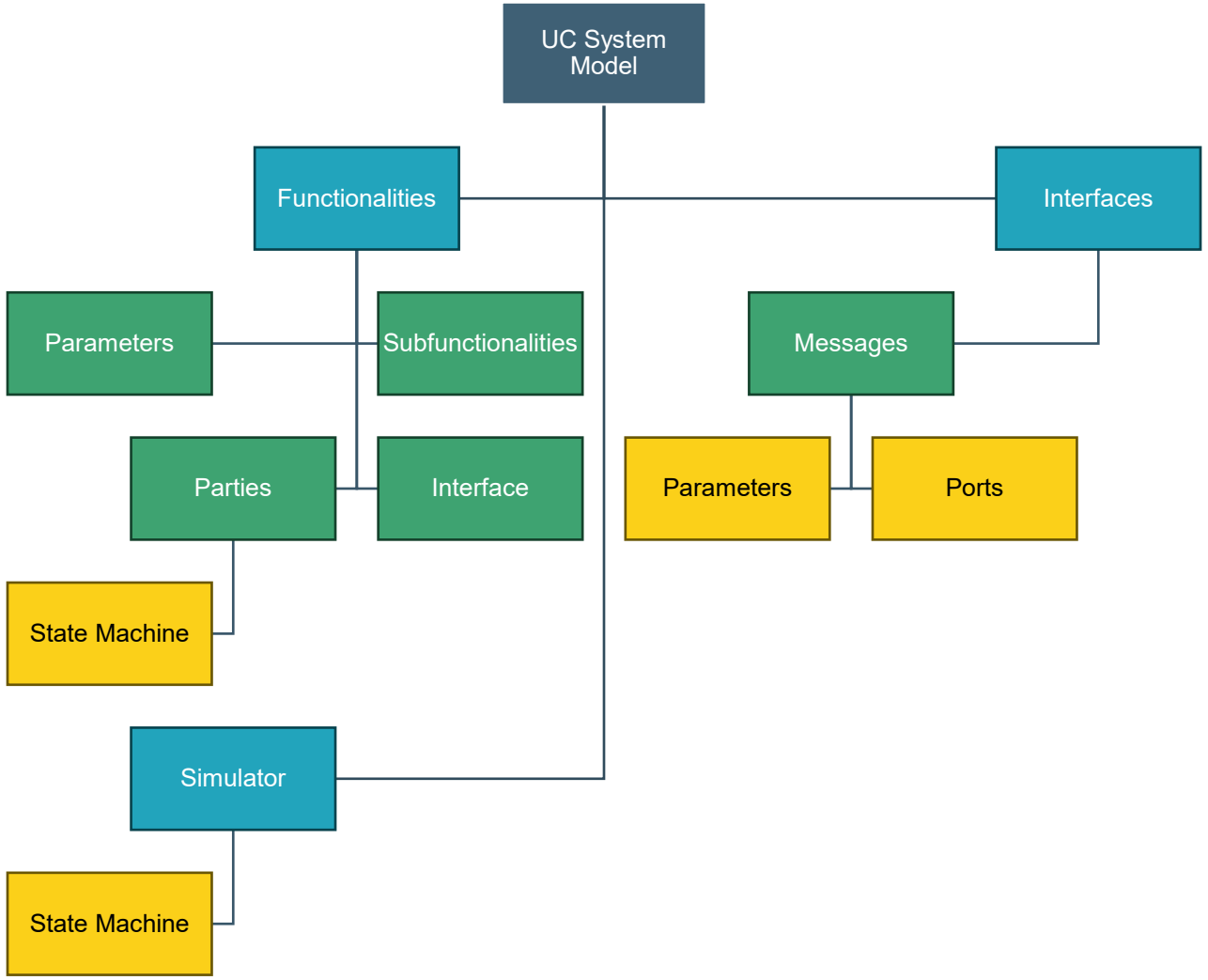


https://www.qualicen.de/real-magic-building-custom-interface-tables-with-cameo-magic-draw-and-generic-tables/



https://en.wikipedia.org/wiki/Enterprise_Architect_%28software%29

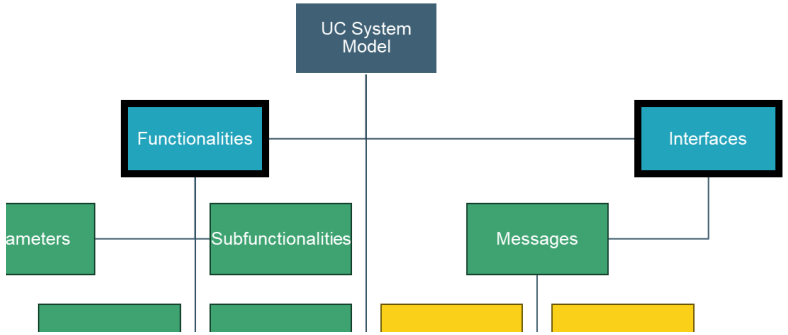
Building Blocks of a UC Model

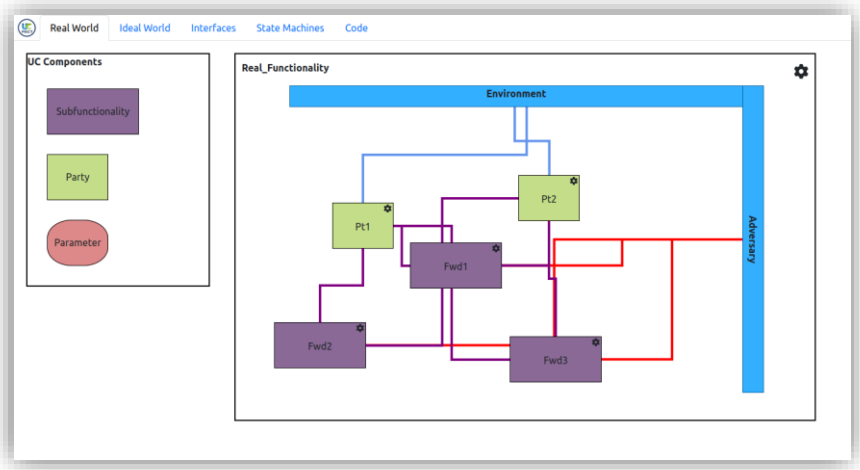


BOSTON UNIVERSITY

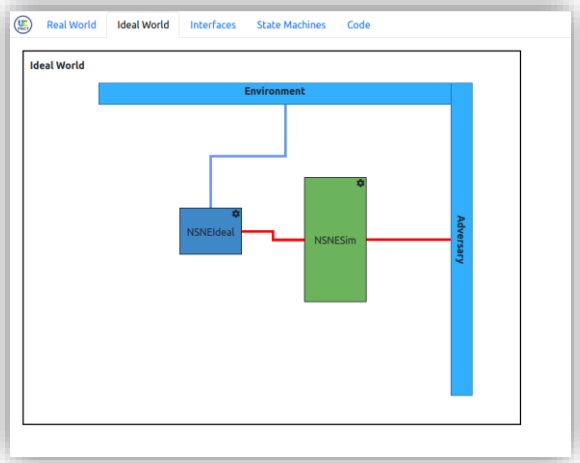
RIVERSIDE RESEARCH

Modeling Basics

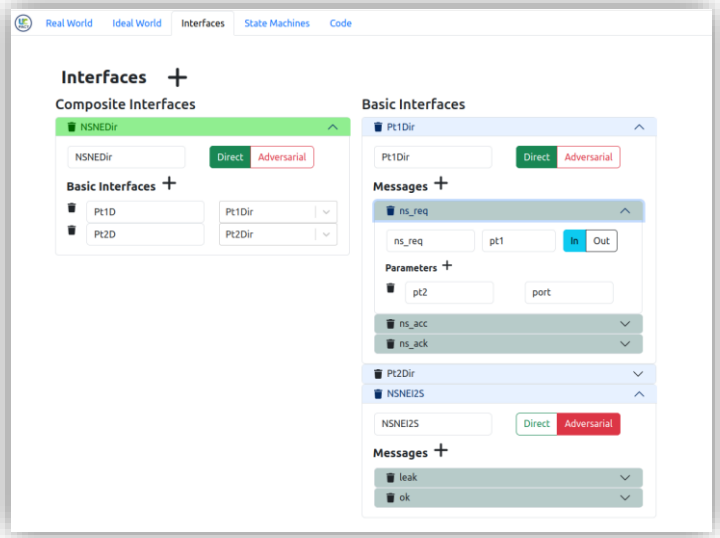




Real World Tab



Ideal World Tab

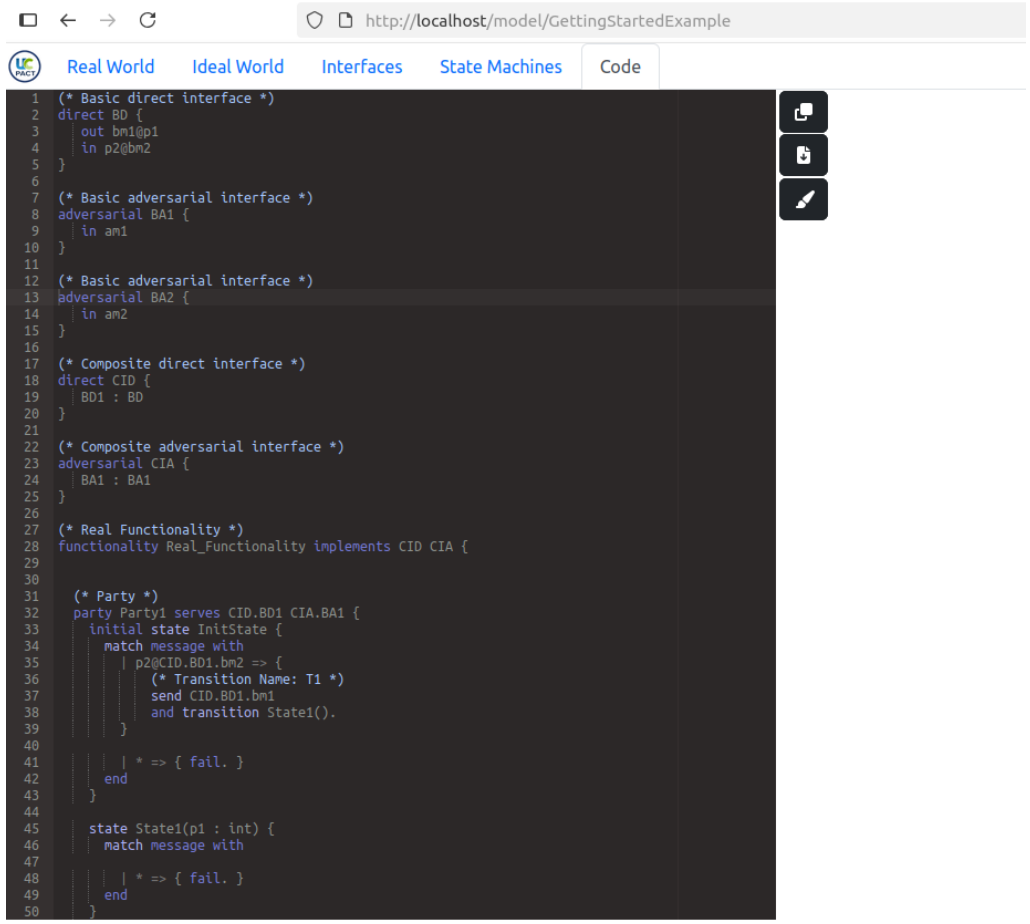


Interfaces Tab

Automatically Generated uc-dsl Code

- What it does
  - Automatically generate a skeleton of the DSL code
  - As much fidelity as you create in the app

- What it doesn't do
  - Allow you to specify ec files
  - Allow you to specify uci files
  - Typecheck



Running Example

- Authenticated Messaging
  - Two parties (Sender and Receiver)
  - Establish authentication token using Lamport's One-time password scheme
  - Sender then includes authentication token with subsequent messages so Receiver knows they are valid

- Subfunctionalities
  - Channel for Sender to push messages to Receiver
    - First-in, First-out messaging
  - Random Oracle for computing hashes

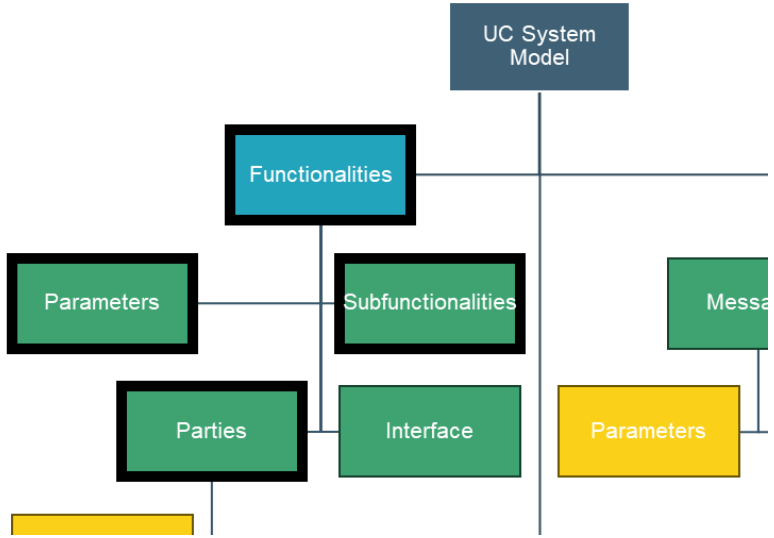
- Adversary capabilities
  - No access to parties
  - No access to RO
  - Channel leaks all messages, authentication tokens, etc. to the adversary
    - Adversary can also inject messages into the channel

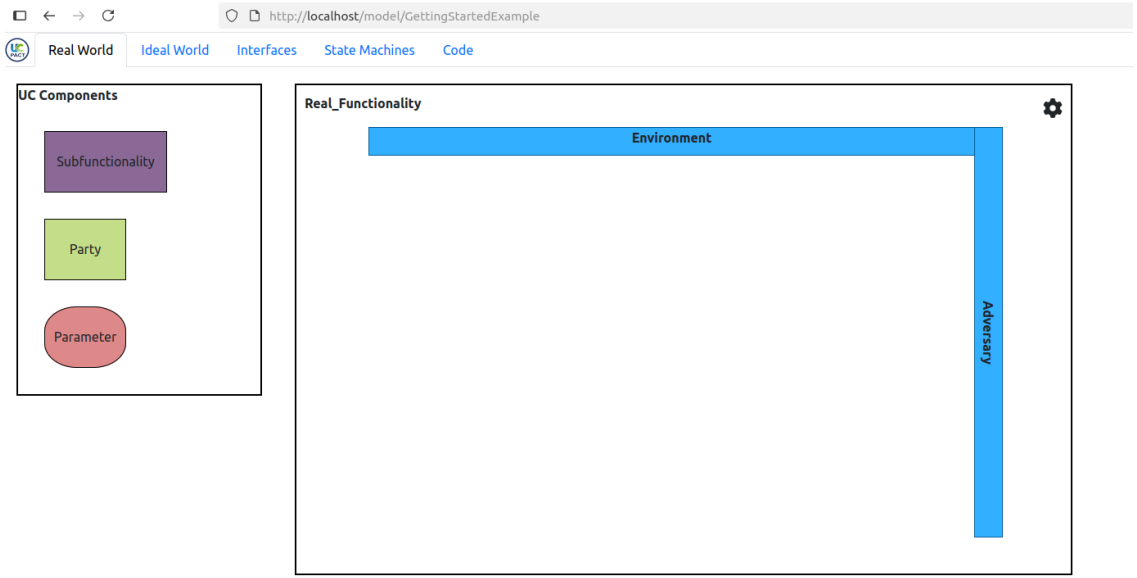
BOSTON UNIVERSITY
RIVERSIDE RESEARCH

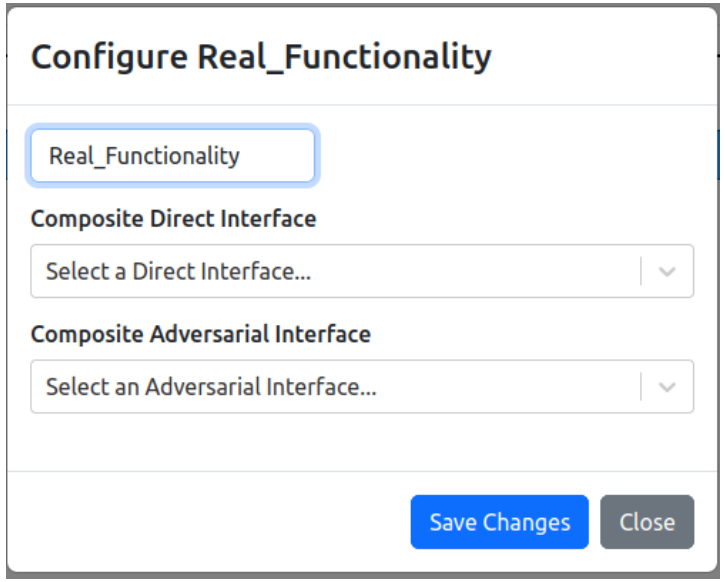
Real World Deep Dive

› Real Functionalities must implement an interface

› Real Functionalities must have at least one party



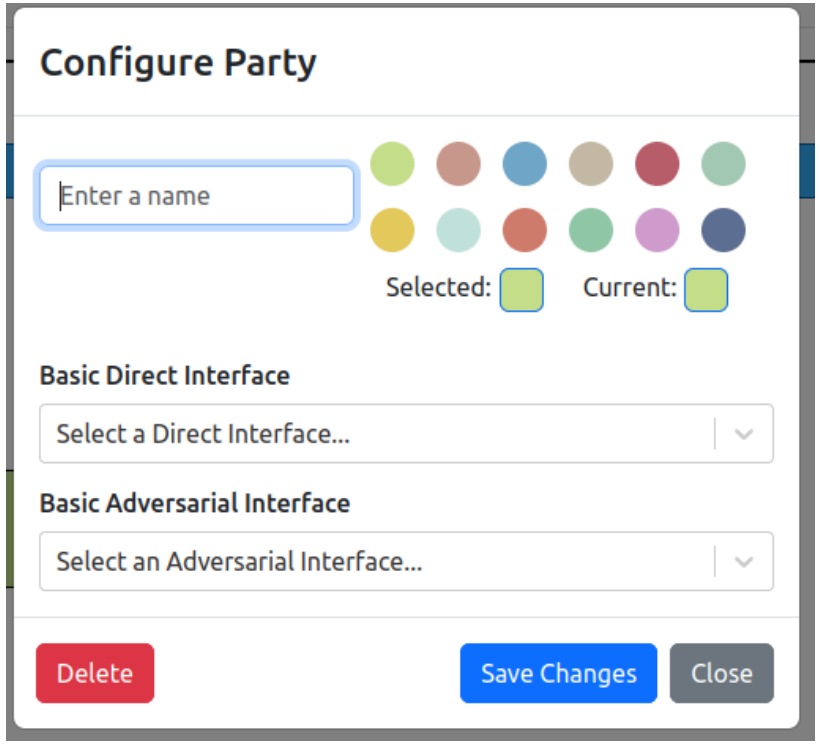


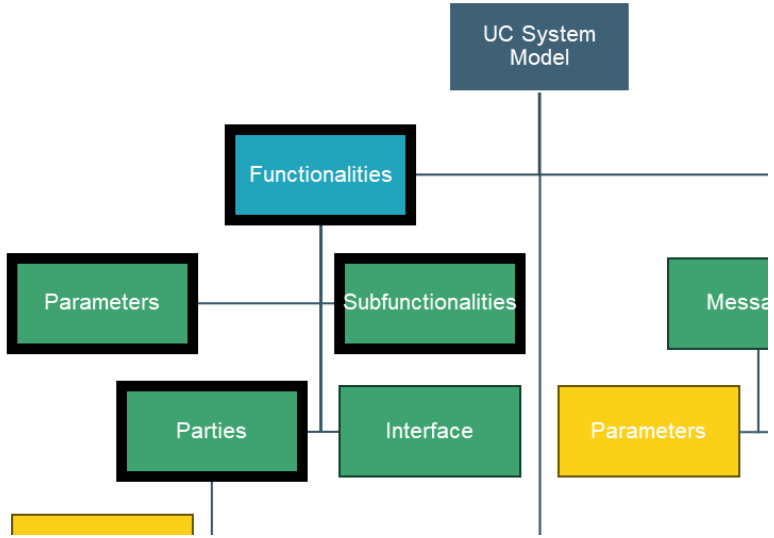


Real World Deep Dive

› Parties are named

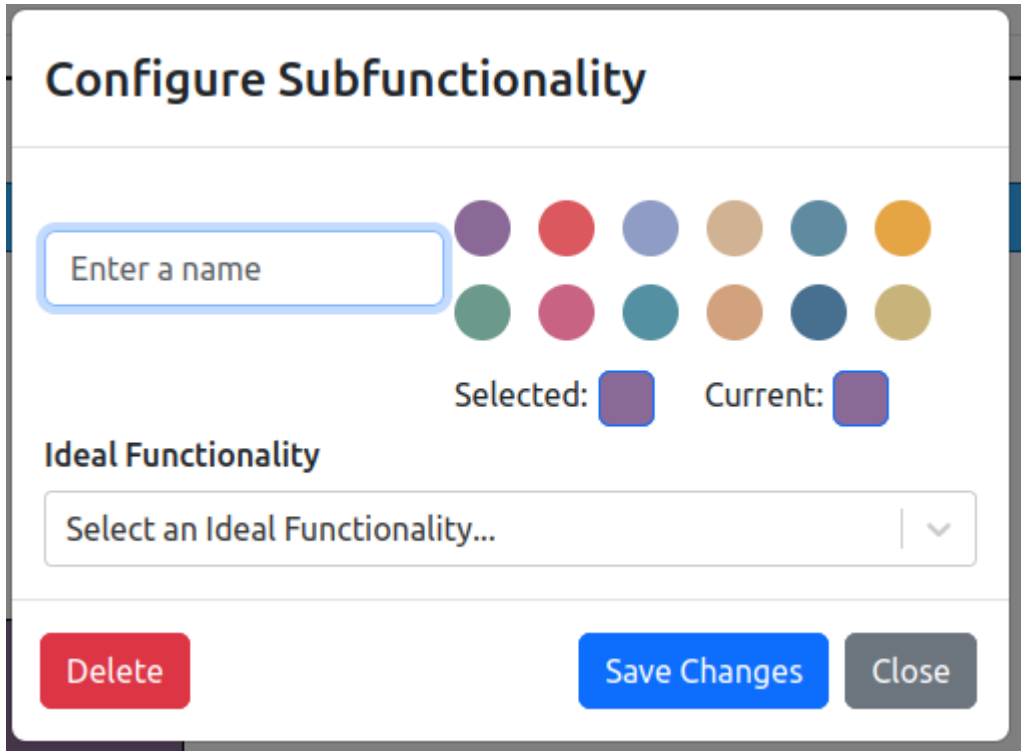
› Parties serve interfaces

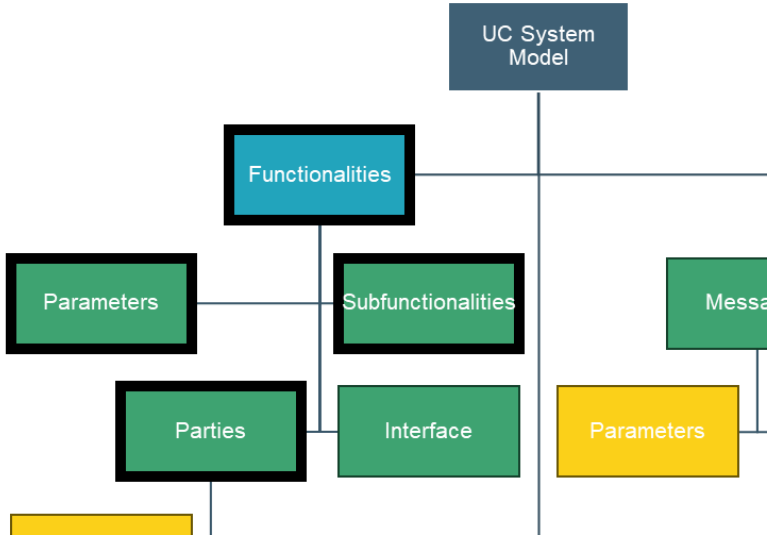




Real World Deep Dive

› Subfunctionalities are idealized components



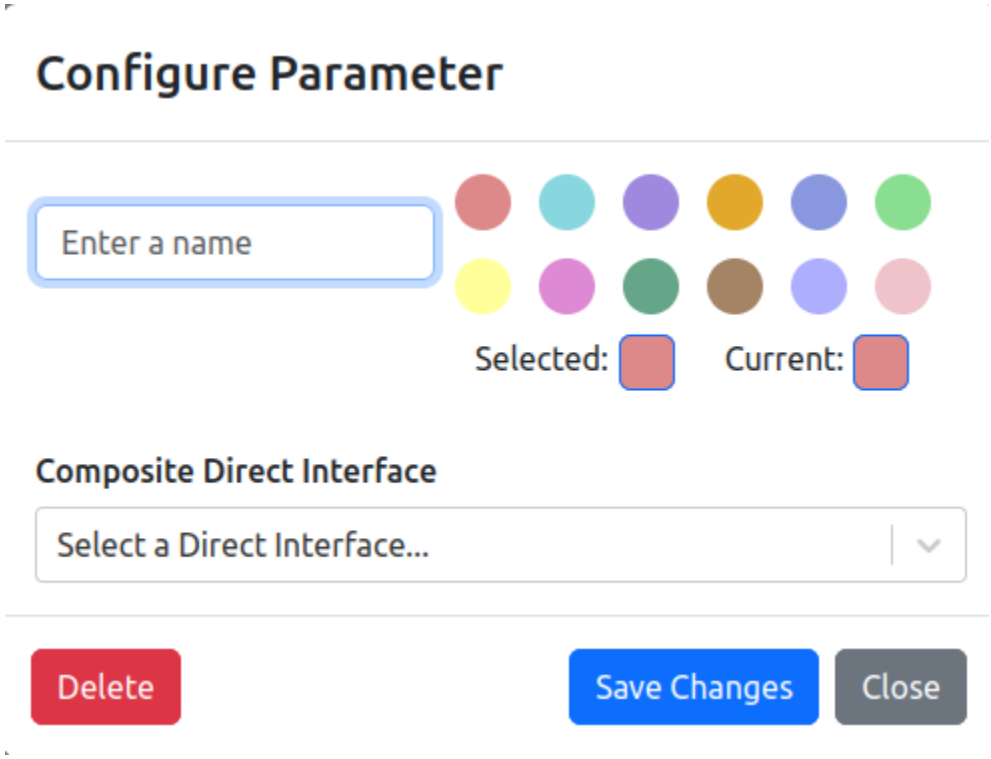


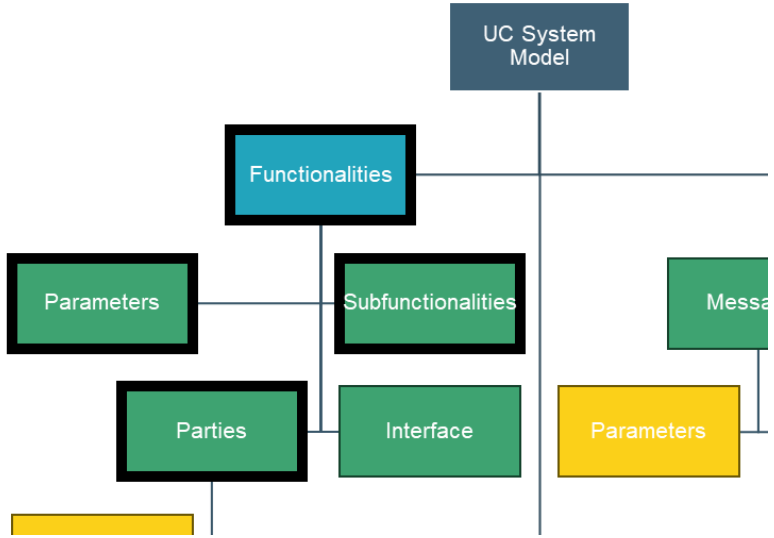
BOSTON UNIVERSITY

RIVERSIDE RESEARCH

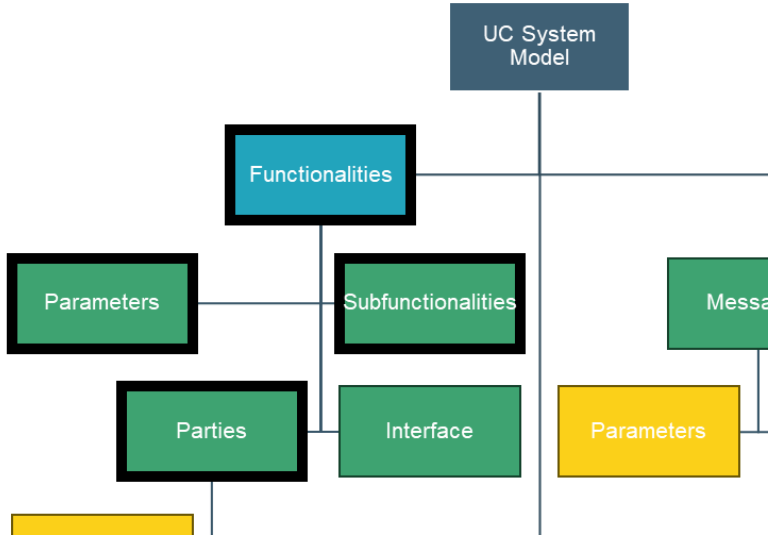
Real World Deep Dive

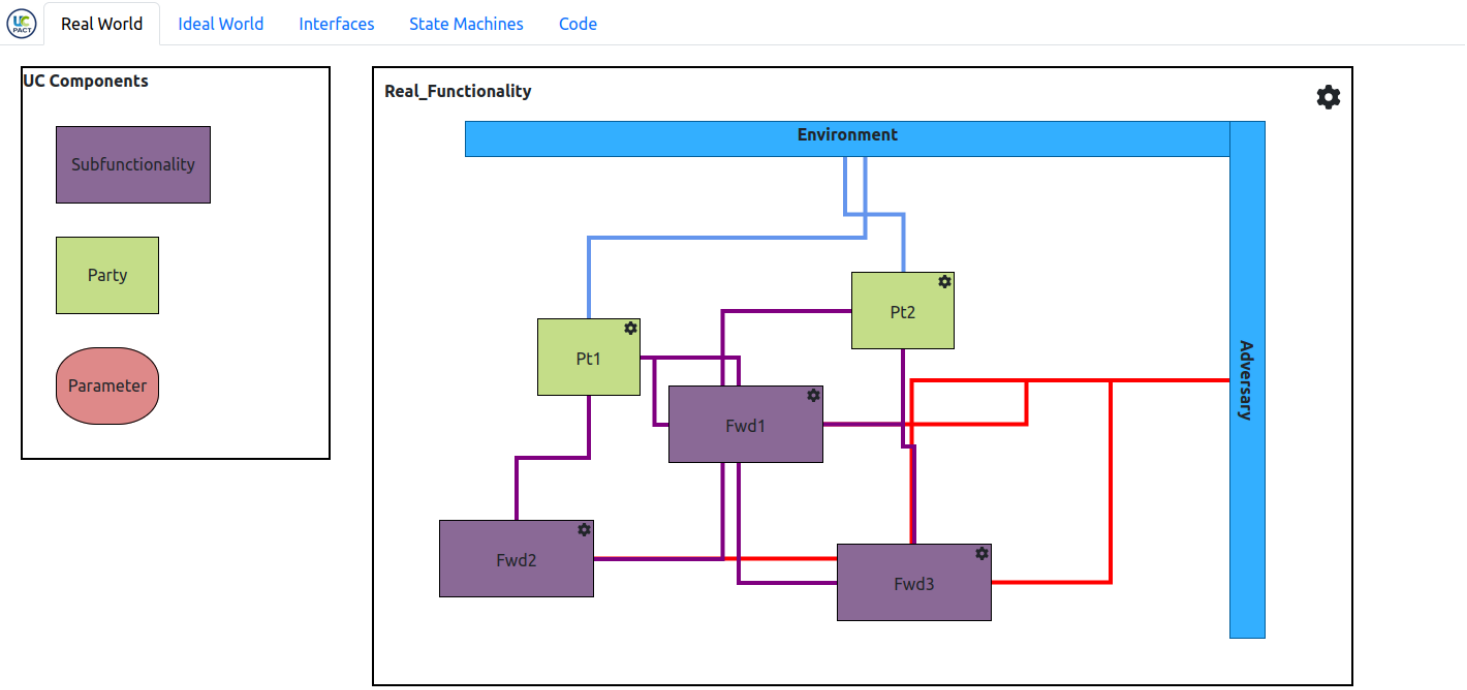
› Parameters are components the parties can pass messages to





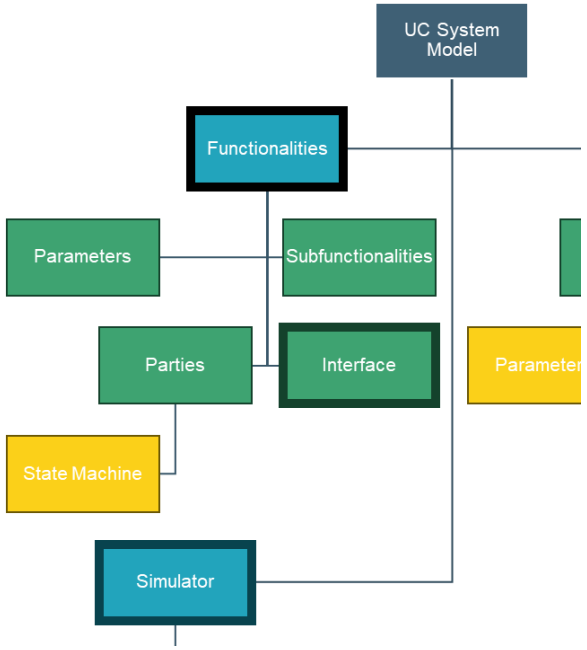
Real World Deep Dive

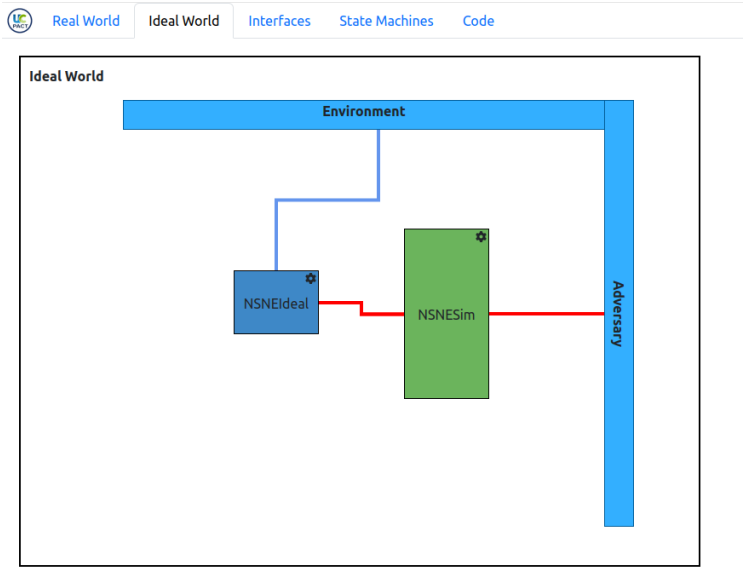


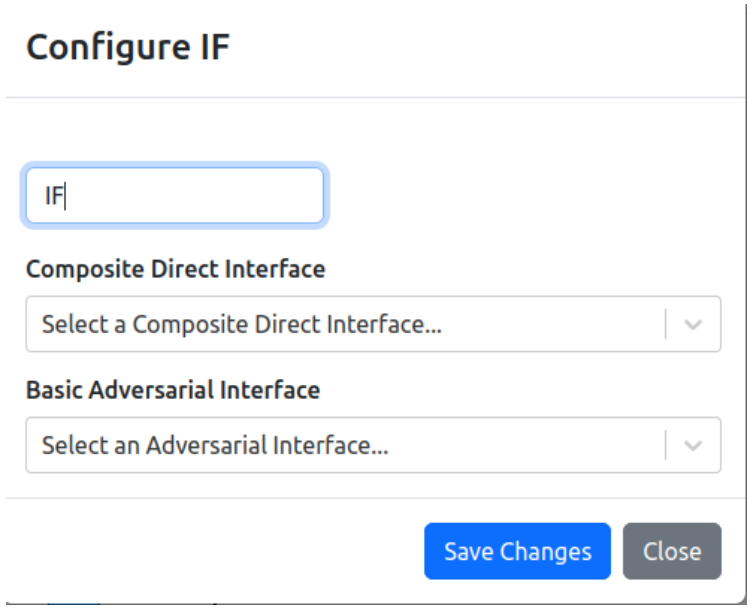


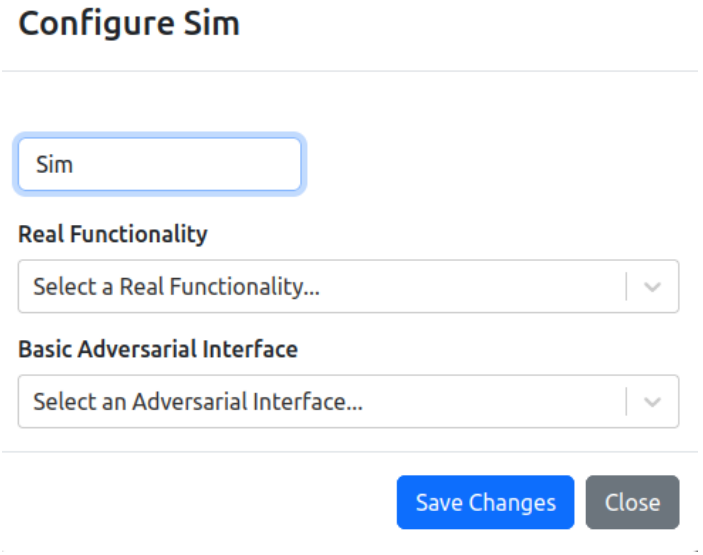
Ideal World Deep Dive

- Ideal World design is static; there is an ideal functionality and a simulator
- Ideal Functionality implements interfaces
- Simulator uses an interface to communicate with ideal functionality and simulates a Real Functionality





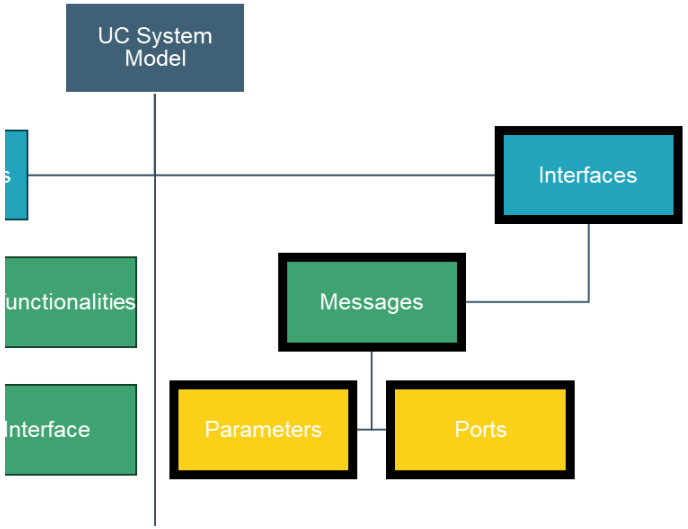


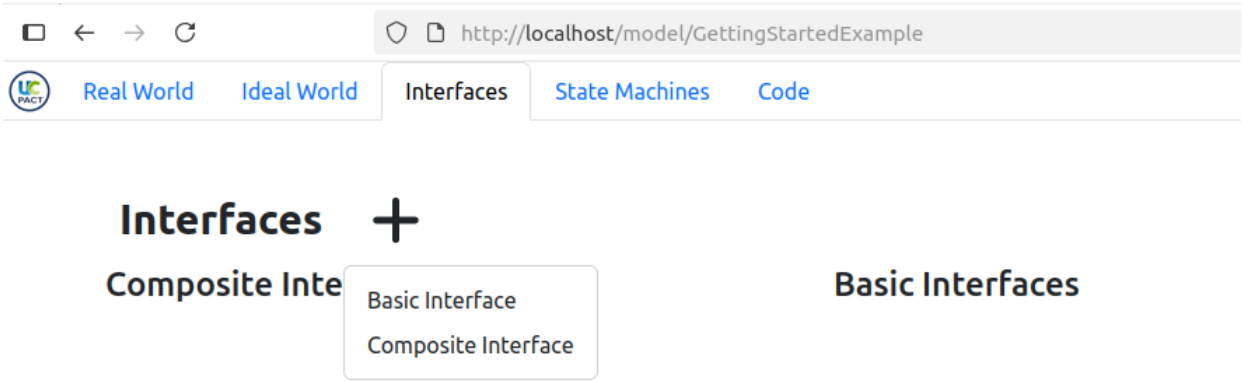


Interfaces Deep Dive

› Interfaces are either Adversarial or Direct

› Interfaces are either Basic or Composite

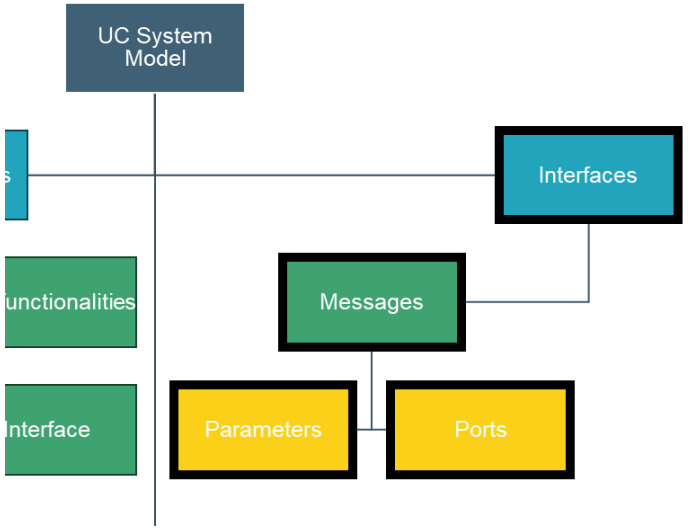


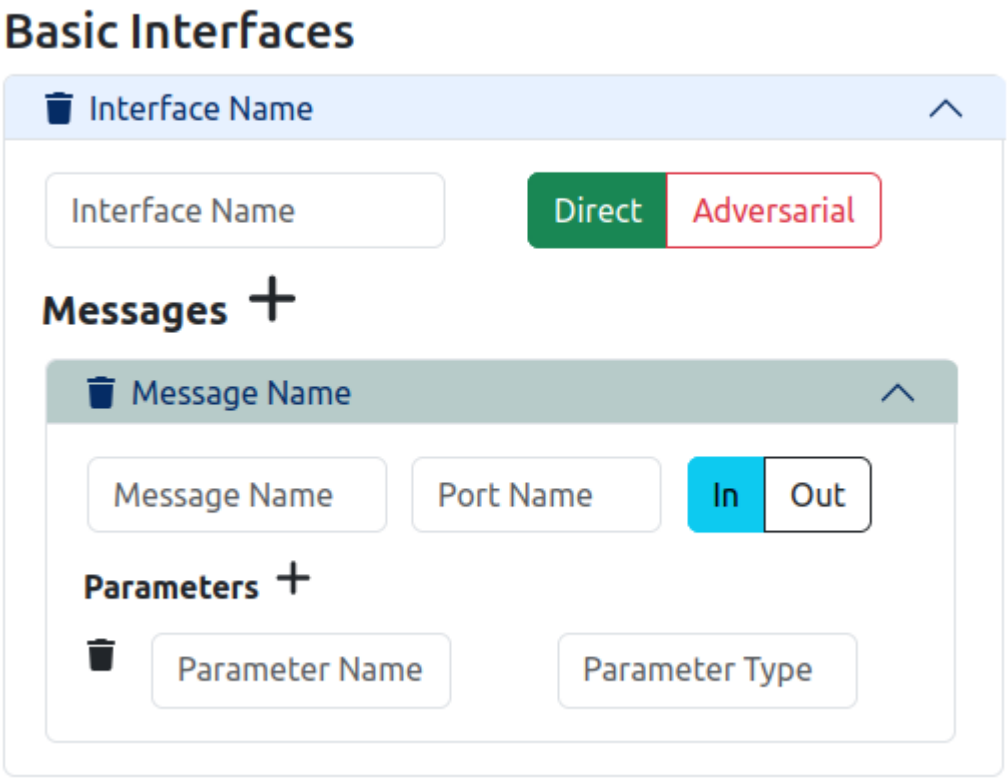


Interfaces Deep Dive

› Parties serve basic interfaces

› The messages form the API for that party

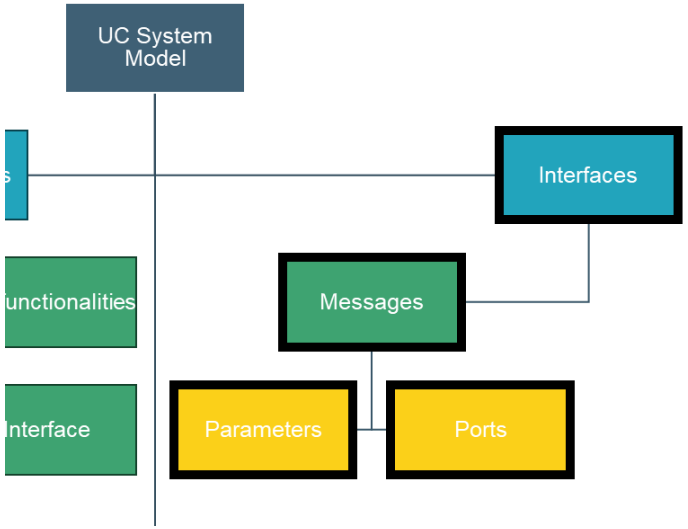


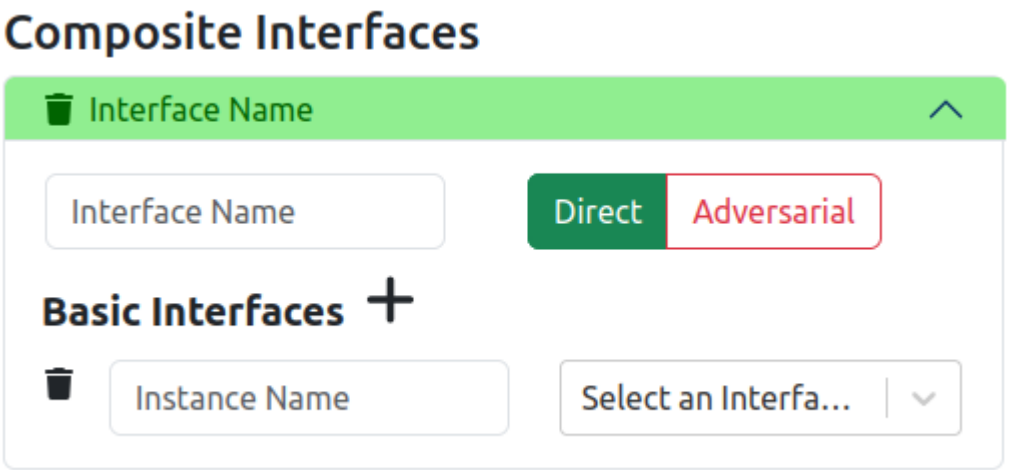


Interfaces Deep Dive

› Functionalities implement composite interfaces

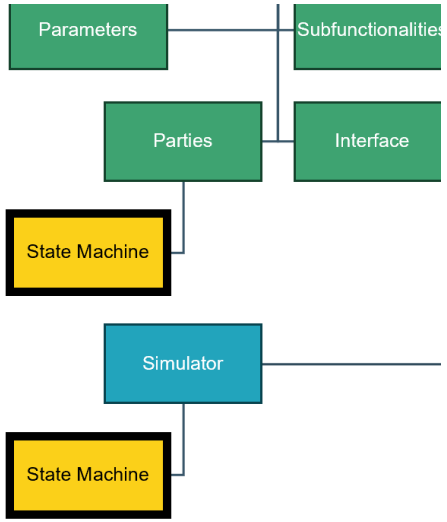
› These are a collection of basic interfaces

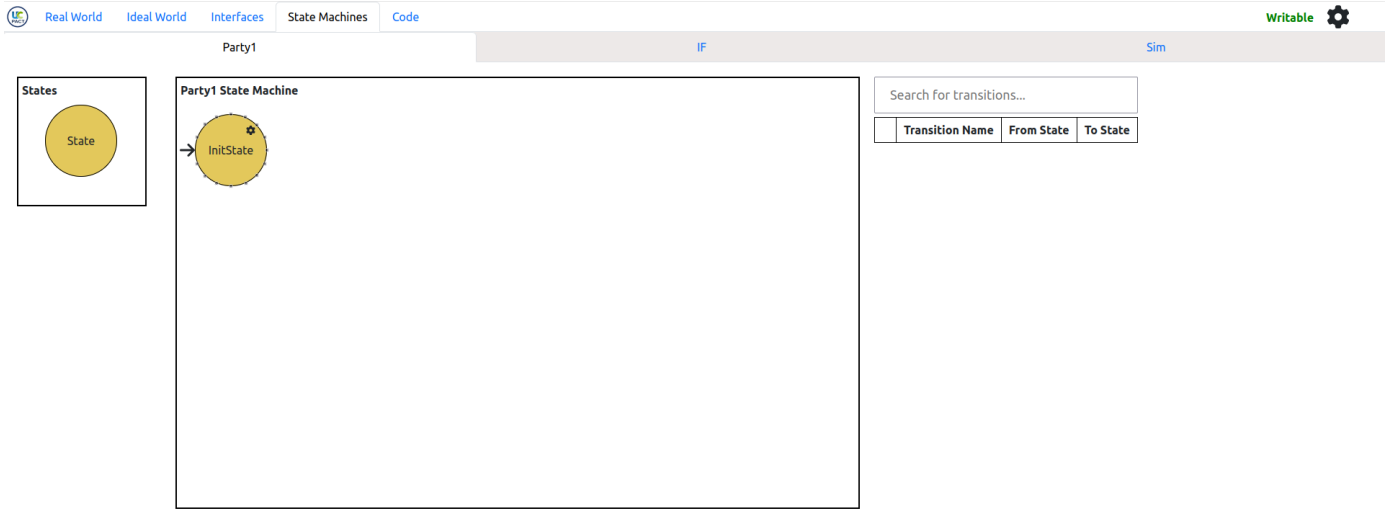


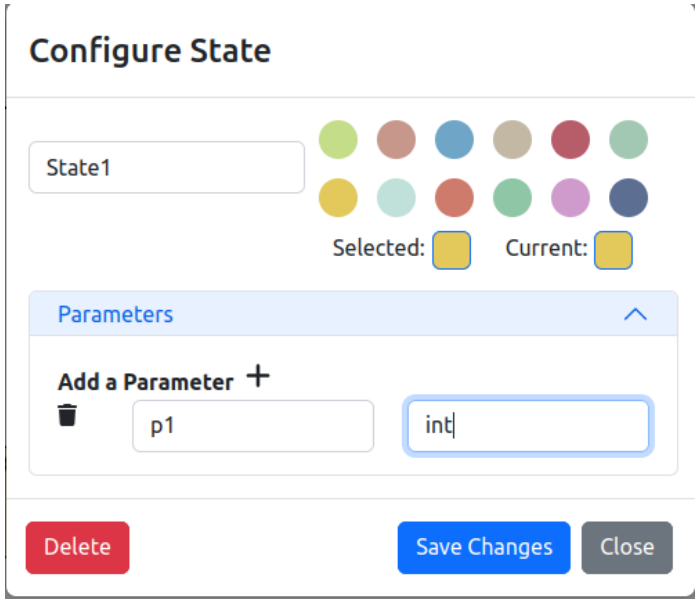


State Machines Deep Dive

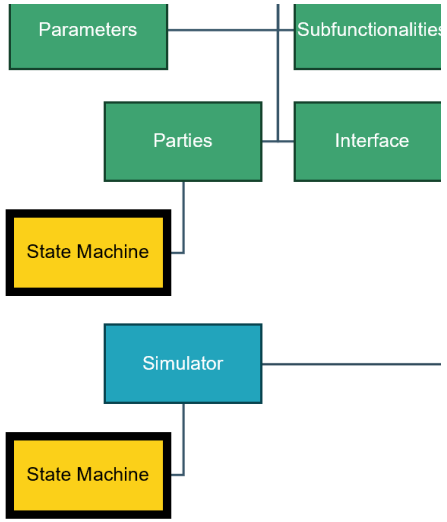
- Parties, Ideal Functionalities, and Simulators have state machines
- State machines identify the behavior of a component when acting on a message



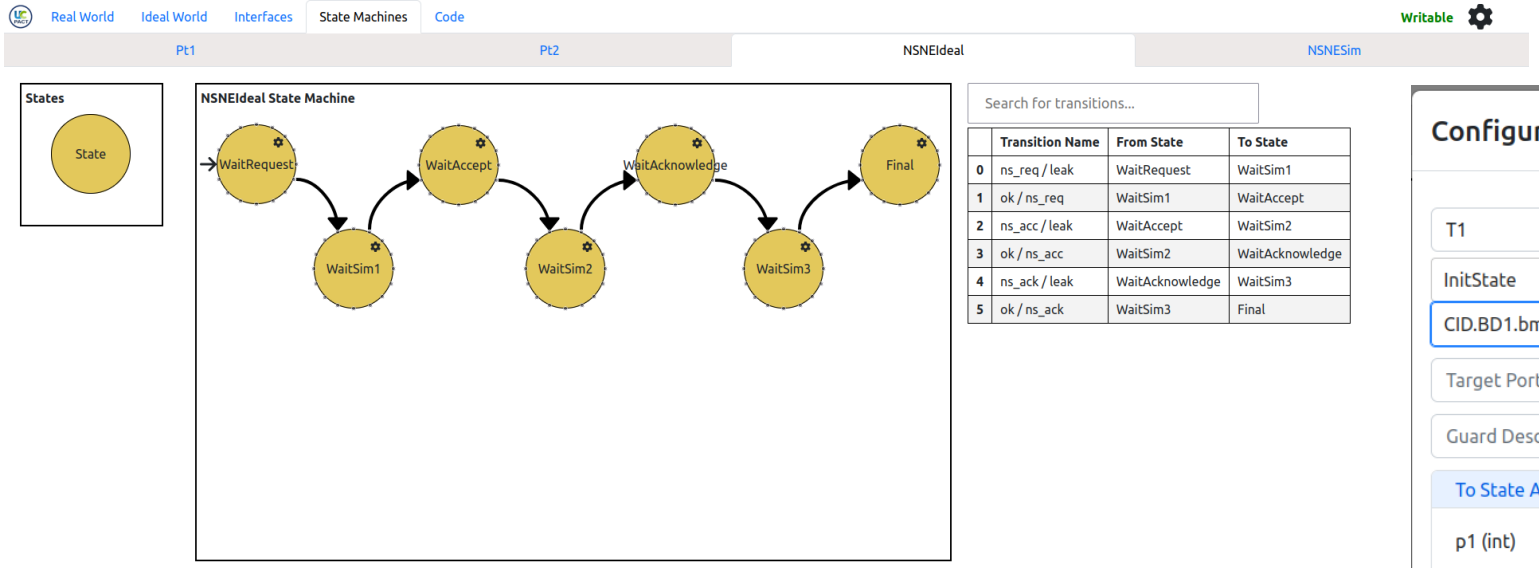




State Machines Deep Dive

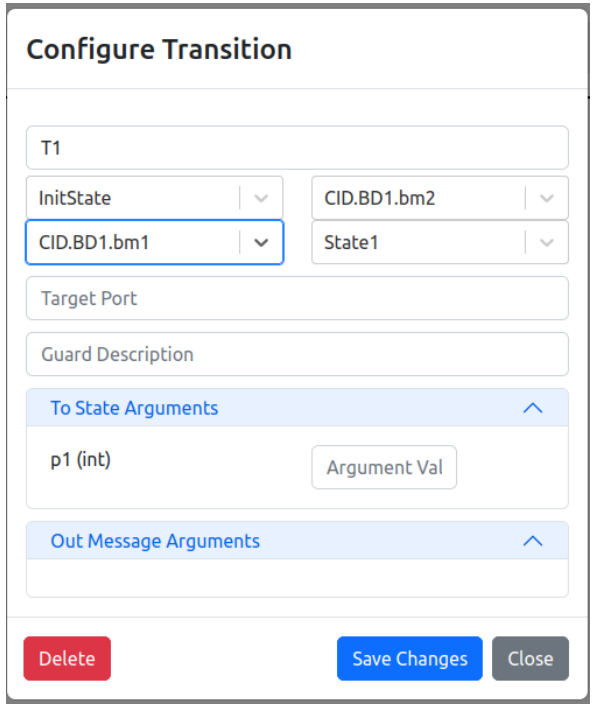


- Processing of a message ends either in a failure or by sending a message somewhere else and transitioning to a new state
- This is configured by adding transitions



Search for transitions...

	Transition Name	From State	To State
0	ns_req / leak	WaitRequest	WaitSim1
1	ok / ns_req	WaitSim1	WaitAccept
2	ns_acc / leak	WaitAccept	WaitSim2
3	ok / ns_acc	WaitSim2	WaitAcknowledge
4	ns_ack / leak	WaitAcknowledge	WaitSim3
5	ok / ns_ack	WaitSim3	Final



Code Generator

- Allows for export of the generated code so that it can be pulled into the other tools (typechecker and interpreter)
- Any new types will need to be declared
- Conditional transitions will need to be configured
- Additional logic around variables or other processing will need to be added

http://localhost/model/GettingStartedExample

Real World | Ideal World | Interfaces | State Machines | Code

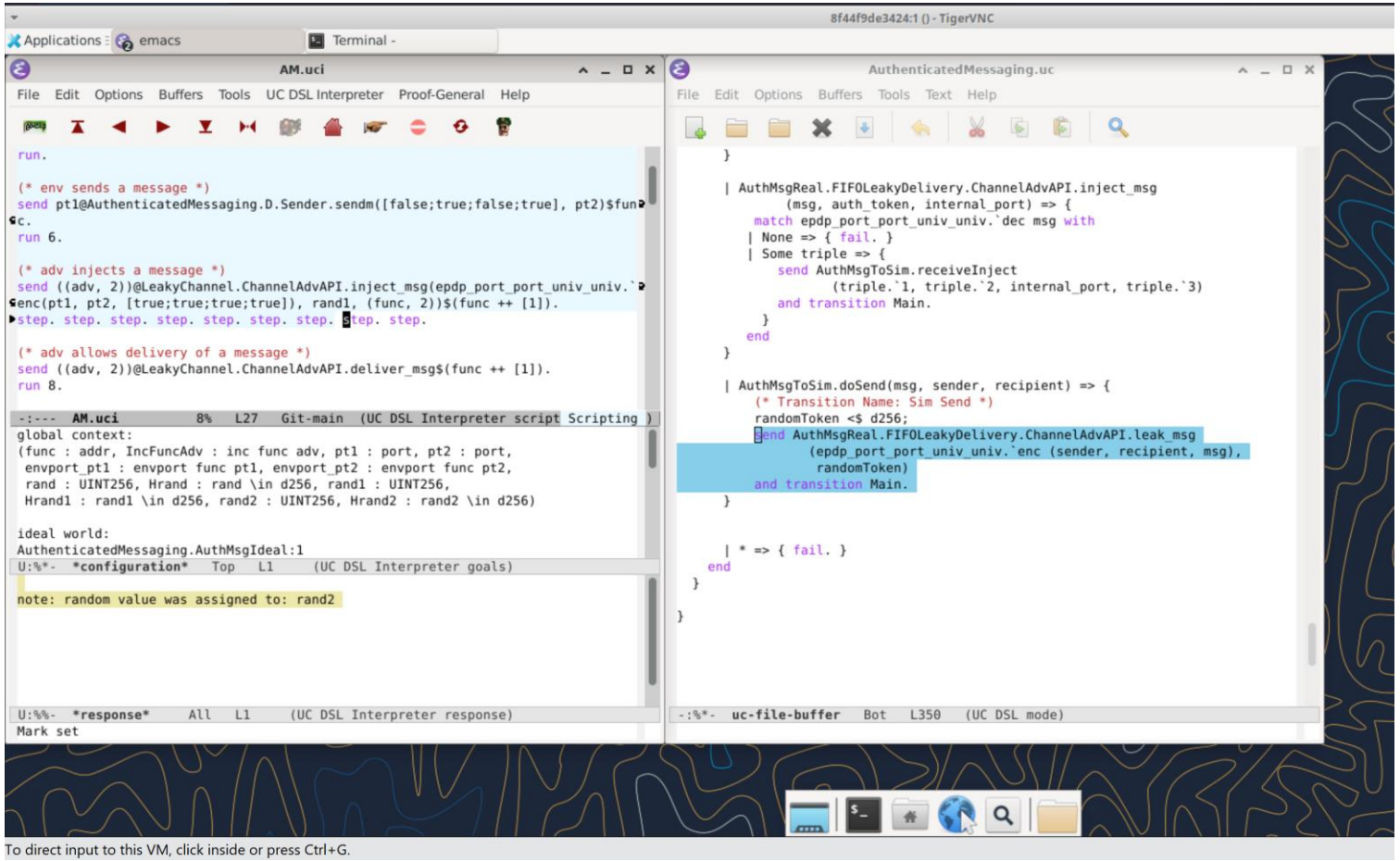
```
(* Basic direct interface *)
direct BD {
  out bm1@p1
  in p2@bm2
}

(* Basic adversarial interface *)
adversarial BA1 {
  in am1
}

(* Basic adversarial interface *)
adversarial BA2 {
  in am2
}

(* Composite direct interface *)
direct CID {
  BD1 : BD
}

(* Composite adversarial interface *)
adversarial CIA {
  BA1 : BA1
}

(* Real Functionality *)
functionality Real_Functionality implements CID CIA {


  (* Party *)
  party Party1 serves CID.BD1 CIA.BA1 {
    initial state InitState {
      match message with
        | p2@CID.BD1.bm2 => {
            (* Transition Name: T1 *)
            send CID.BD1.bm1
            and transition State1().
        }

        | * => { fail. }
      end
    }

    state State1(p1 : int) {
      match message with

        | * => { fail. }
      end
    }
```


BOSTON UNIVERSITY

RIVERSIDE RESEARCH

What next?

› **ucdsl-tools-docker**: some dockerfiles that make using the uc-dsl typechecker and interpreter easier (including setting up Emacs)



ucdsl-tools-docker

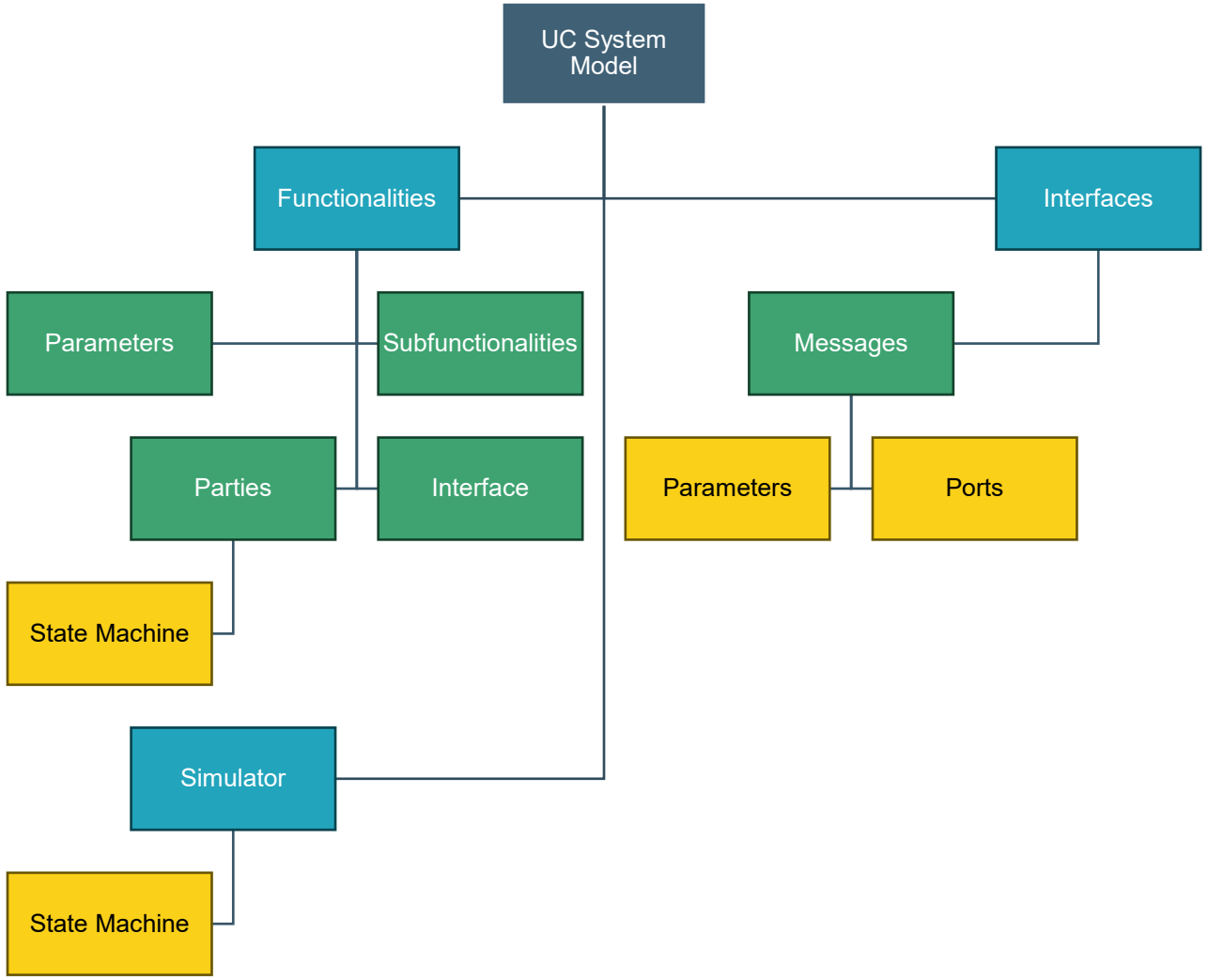
Easy UC User Guide

BOSTON UNIVERSITY

RIVERSIDE RESEARCH

RIVERSIDE RESEARCH

Questions?


UC System Model
Functionalities
Interfaces
Parameters
Subfunctionalities
Messages
Parties
Interface
Parameters
Ports
State Machine
Simulator
State Machine