

UC-PACT: Universal Composability for Preventing Adversarial Composition Techniques



└ Tool Overview
Mike Clark
harden@riversideresearch.org

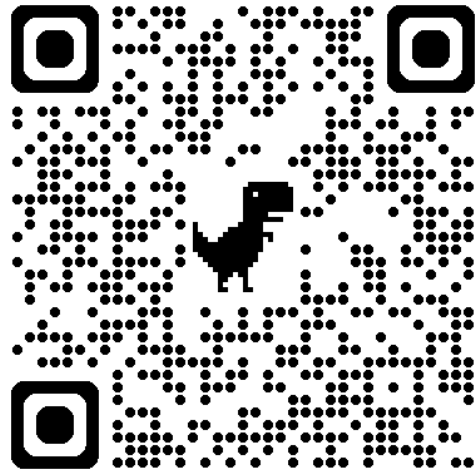
12 August 2025

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.

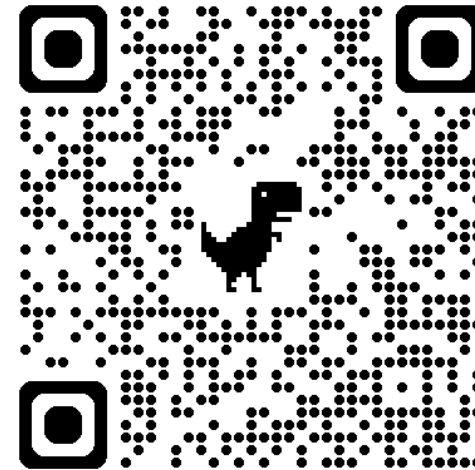
Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA and NIWC Pacific.

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)

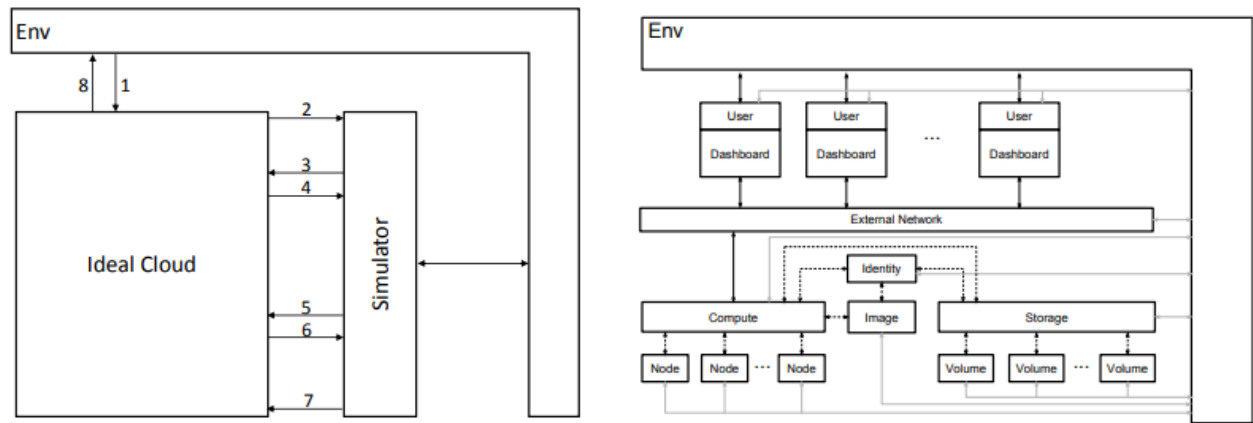


UC-PACT WebApp

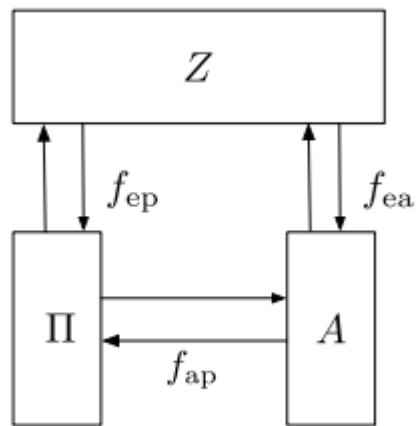


ucdsl-tools-docker

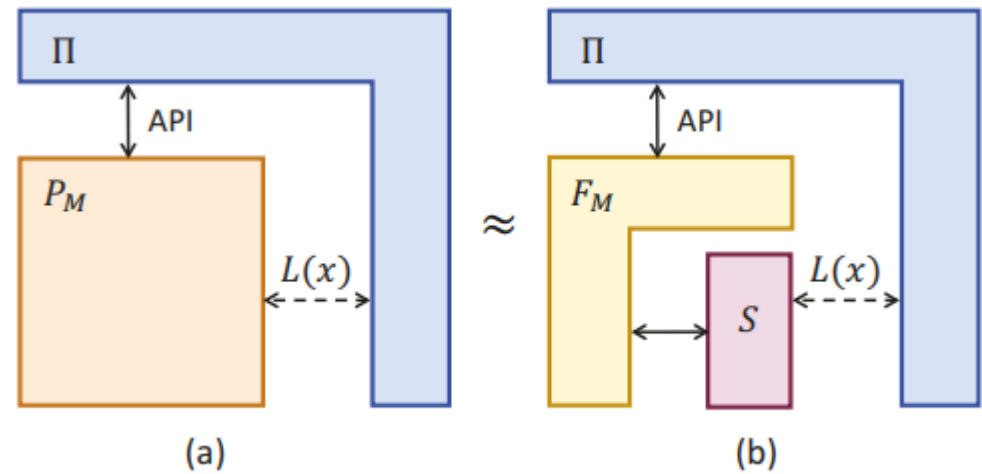
Motivating the problem



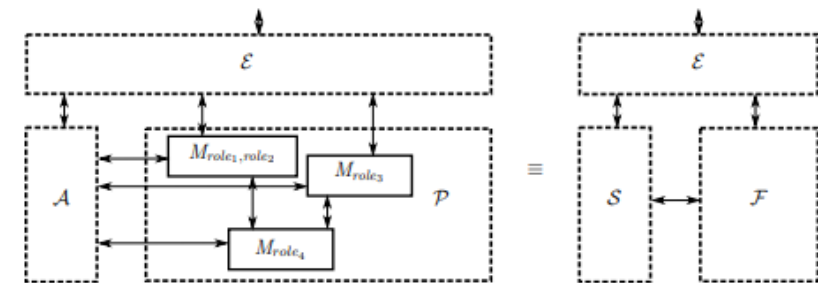
On the universally composable security of OpenStack, Hogan, et al.



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

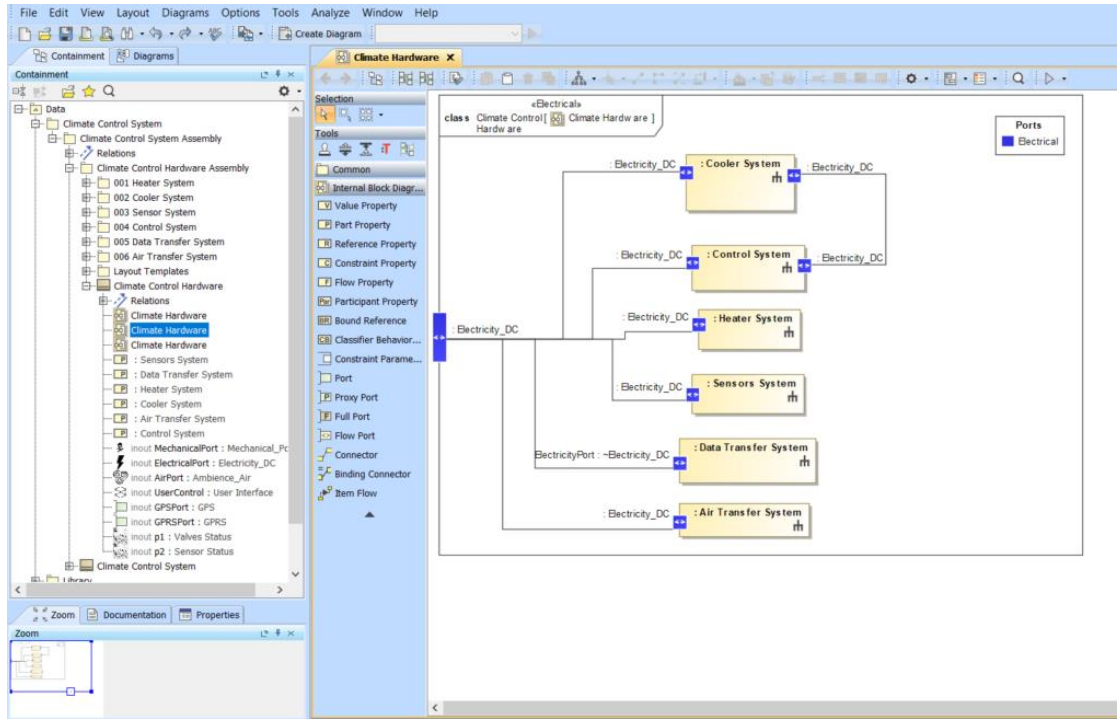


Using Universal Composition to Design and Analyze Secure Complex Hardware Systems, Canetti, et al.

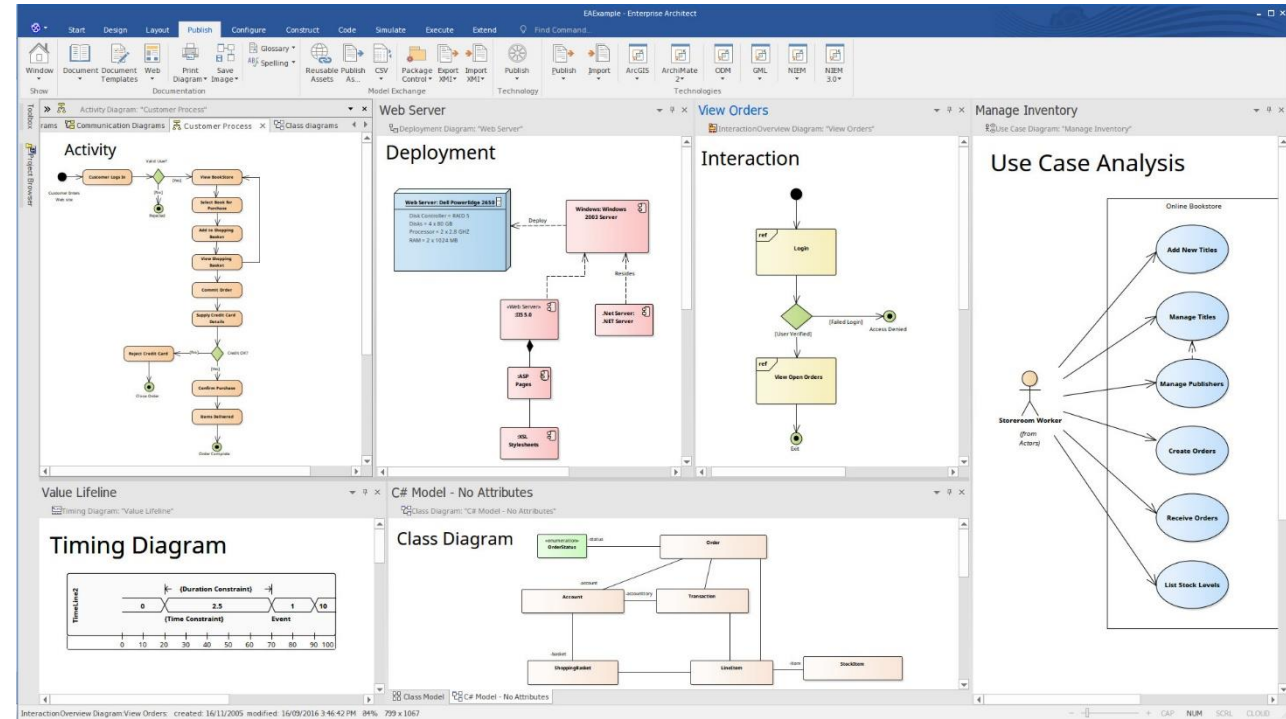


iUC: Flexible Universal Composability Made Simple, Camenisch, et al.

Motivating the problem

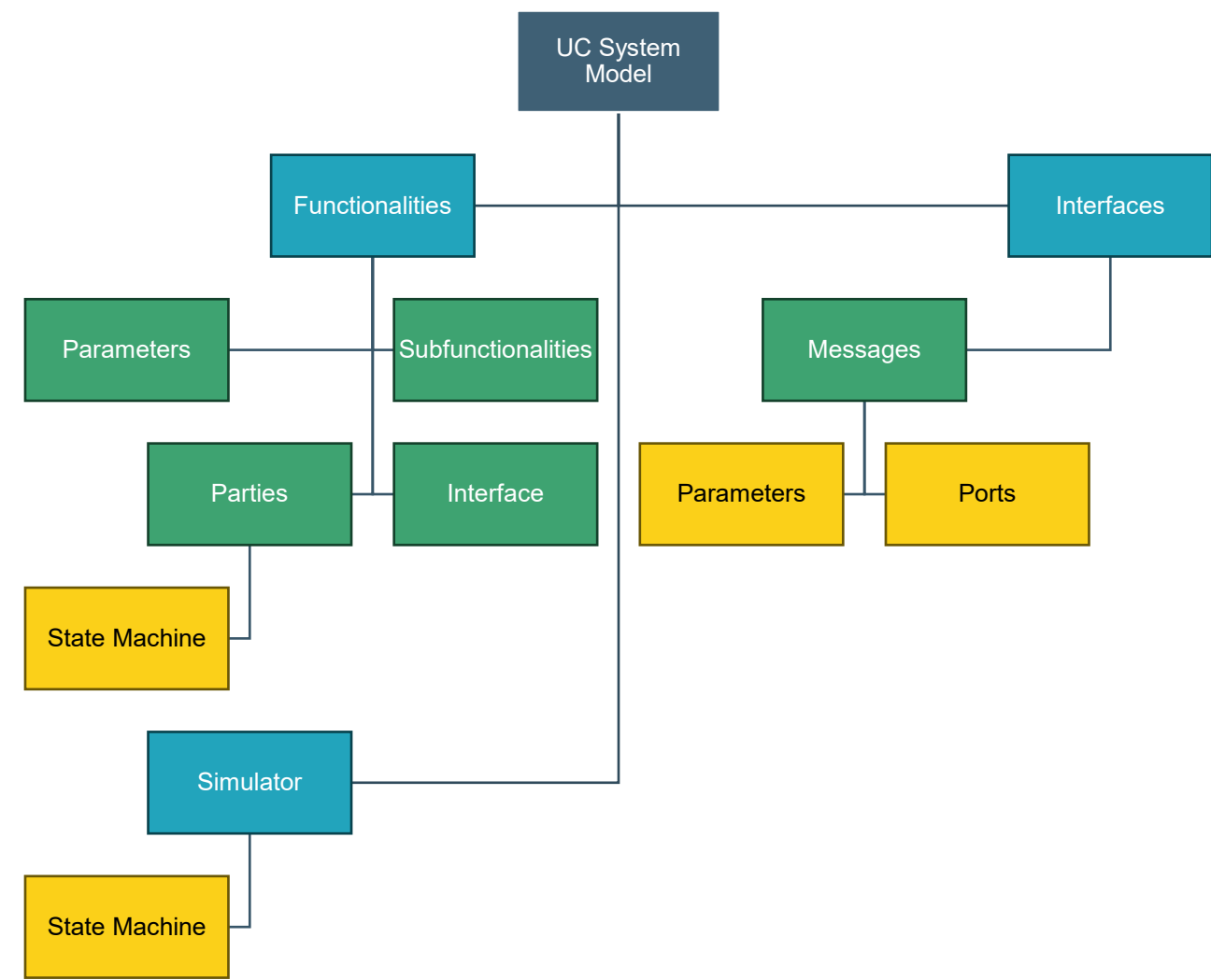


<https://www.qualicen.de/real-magic-building-custom-interface-tables-with-cameo-magic-draw-and-generic-tables/>

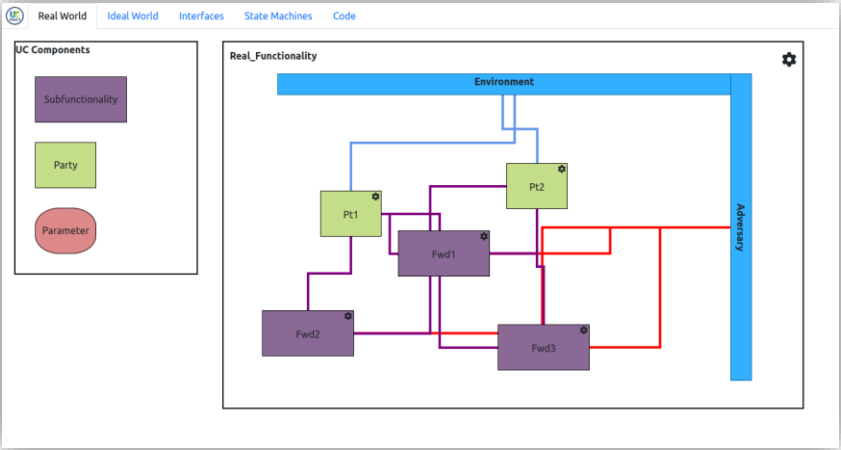
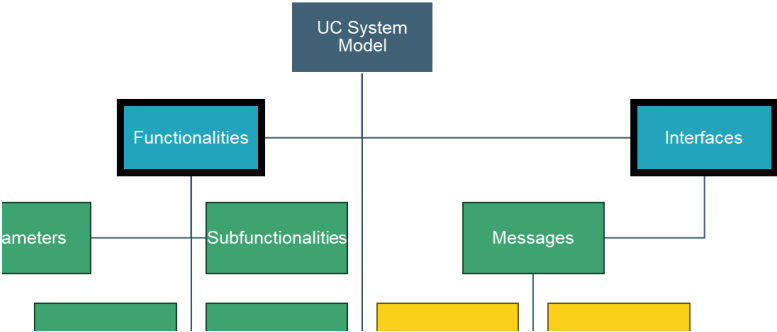


https://en.wikipedia.org/wiki/Enterprise_Architect_%28software%29

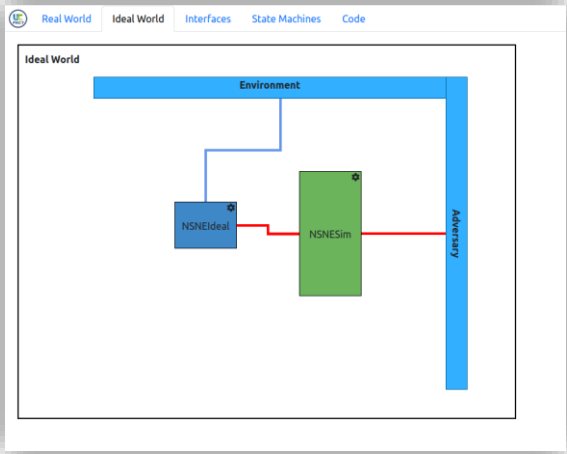
Building Blocks of a UC Model



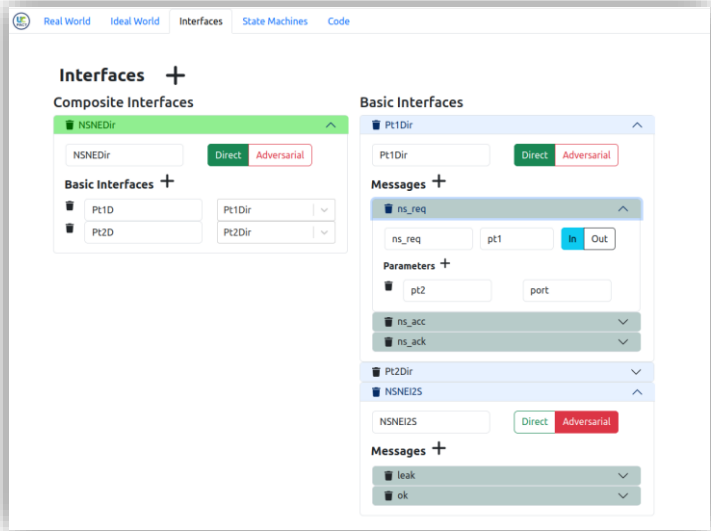
Modeling Basics



Real World Tab



Ideal World Tab



Interfaces Tab

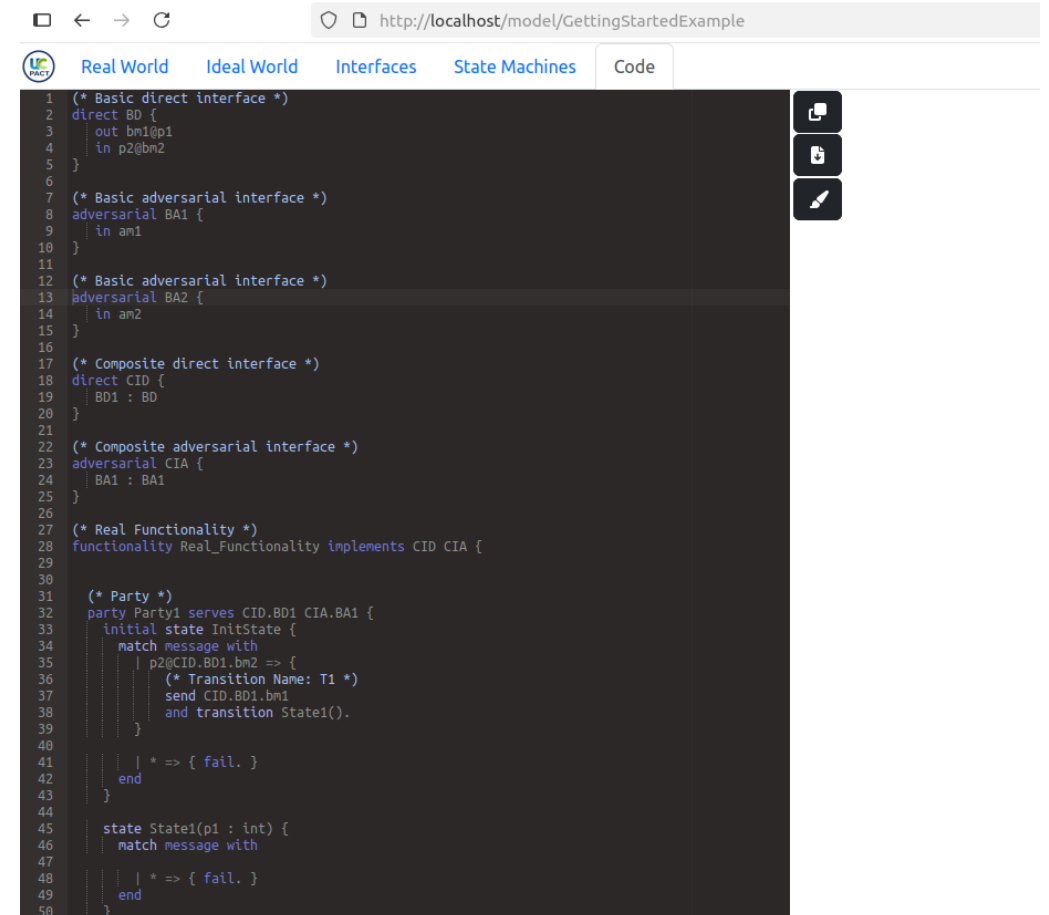
Automatically Generated uc-dsl Code

› 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



The screenshot shows a web browser window with the address bar displaying `http://localhost/model/GettingStartedExample`. The browser has tabs for 'Real World', 'Ideal World', 'Interfaces', 'State Machines', and 'Code'. The 'Code' tab is active, showing a code editor with UC-DSL code. The code is as follows:

```
1 (* Basic direct interface *)
2 direct BD {
3   out bm1@p1
4   in p2@bm2
5 }
6
7 (* Basic adversarial interface *)
8 adversarial BA1 {
9   in am1
10 }
11
12 (* Basic adversarial interface *)
13 adversarial BA2 {
14   in am2
15 }
16
17 (* Composite direct interface *)
18 direct CID {
19   BD1 : BD
20 }
21
22 (* Composite adversarial interface *)
23 adversarial CIA {
24   BA1 : BA1
25 }
26
27 (* Real Functionality *)
28 functionality Real_Functionality implements CID CIA {
29
30
31 (* Party *)
32 party Party1 serves CID.BD1 CIA.BA1 {
33   initial state InitState {
34     match message with
35     | p2@CID.BD1.bm2 => {
36       (* Transition Name: T1 *)
37       send CID.BD1.bm1
38       and transition State1().
39     }
40
41     | * => { fail. }
42   end
43 }
44
45 state State1(p1 : int) {
46   match message with
47   | * => { fail. }
48   end
49 }
50 }
```


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

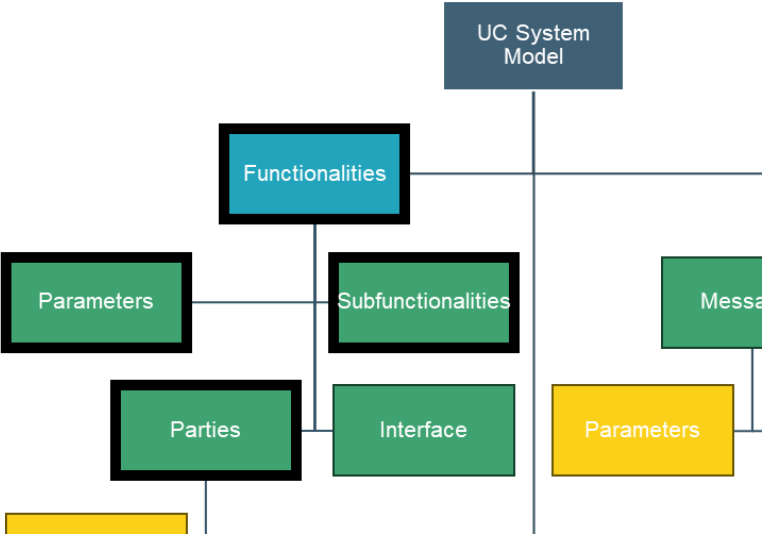
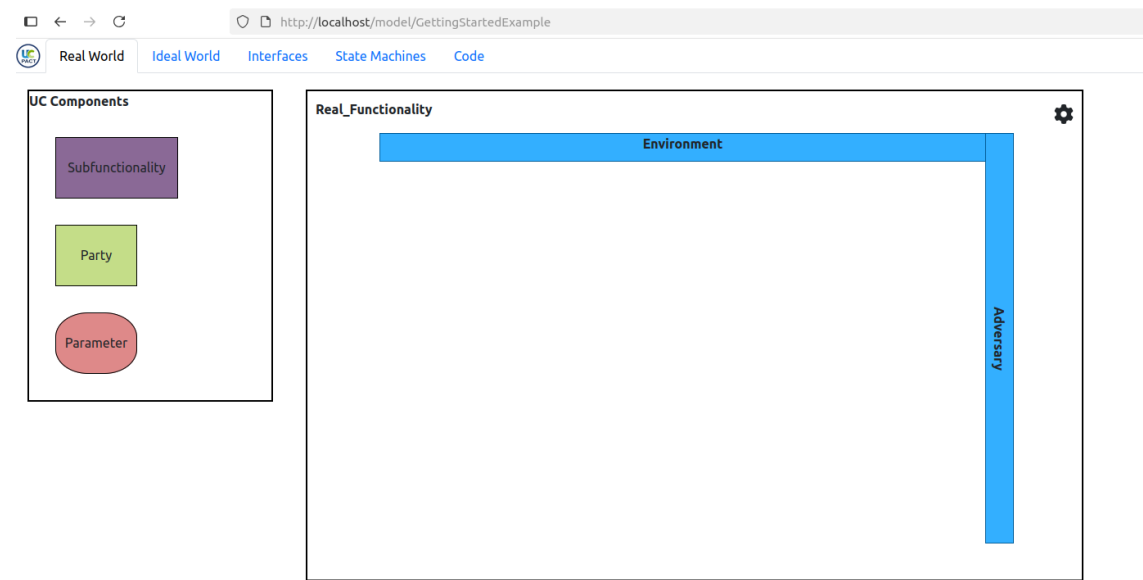
- › Channel for Sender to push messages to Receiver
 - › First-in, First-out messaging
- › Random Oracle for computing hashes

› 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

Real World Deep Dive

- › Real Functionalities must implement an interface
- › Real Functionalities must have at least one party



Configure Real_Functionality

Real_Functionality

Composite Direct Interface

Select a Direct Interface...

Composite Adversarial Interface

Select an Adversarial Interface...

Save Changes

Close

Real World Deep Dive

- › Parties are named
- › Parties serve interfaces

Configure Party

Enter a name

Selected: Current:

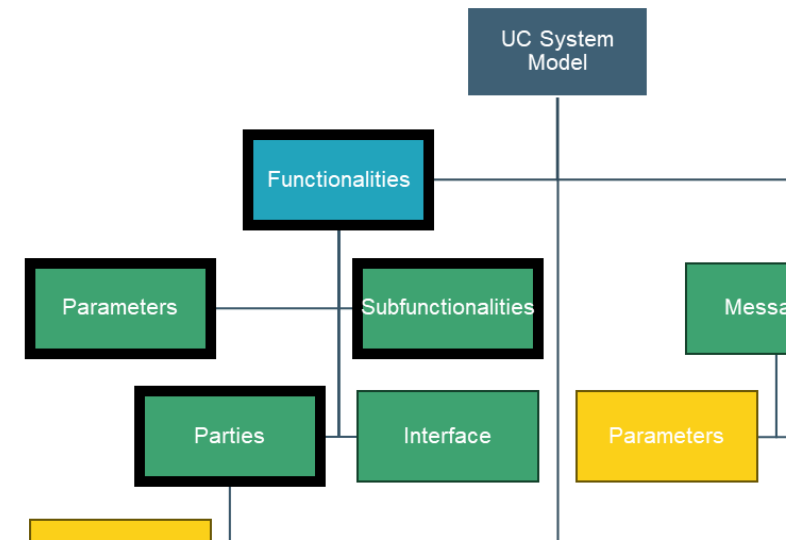
Basic Direct Interface

Select a Direct Interface... ▼

Basic Adversarial Interface

Select an Adversarial Interface... ▼

Delete Save Changes Close



Real World Deep Dive

› Subfunctionalities are idealized components

Configure Subfunctionality

Selected: Current:

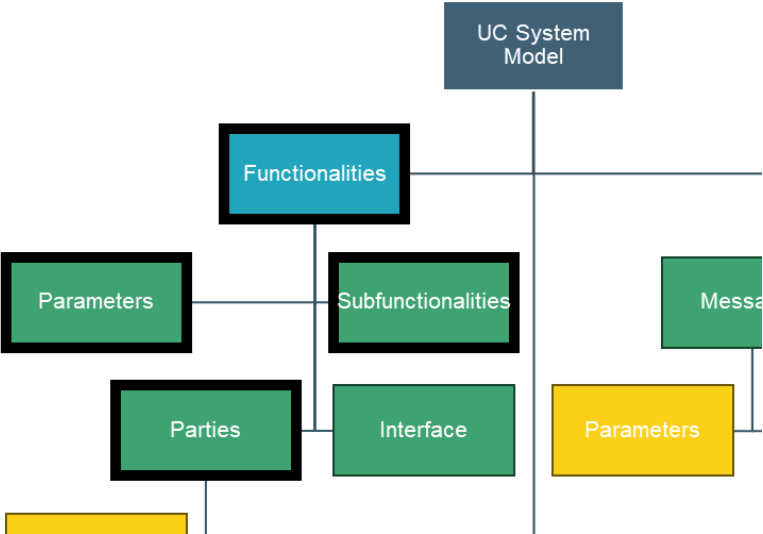
Ideal Functionality

Select an Ideal Functionality...

Delete

Save Changes

Close



Real World Deep Dive

- › Parameters are components the parties can pass messages to

Configure Parameter

Selected: Current:

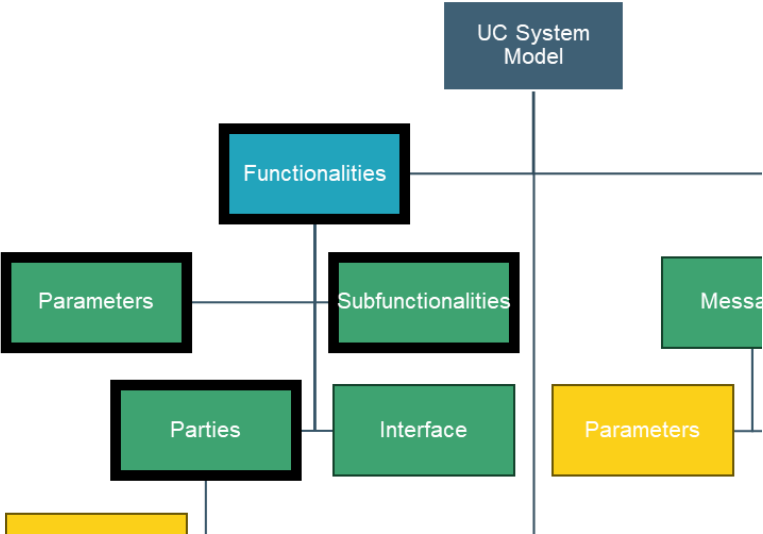
Composite Direct Interface

Select a Direct Interface... | ▾

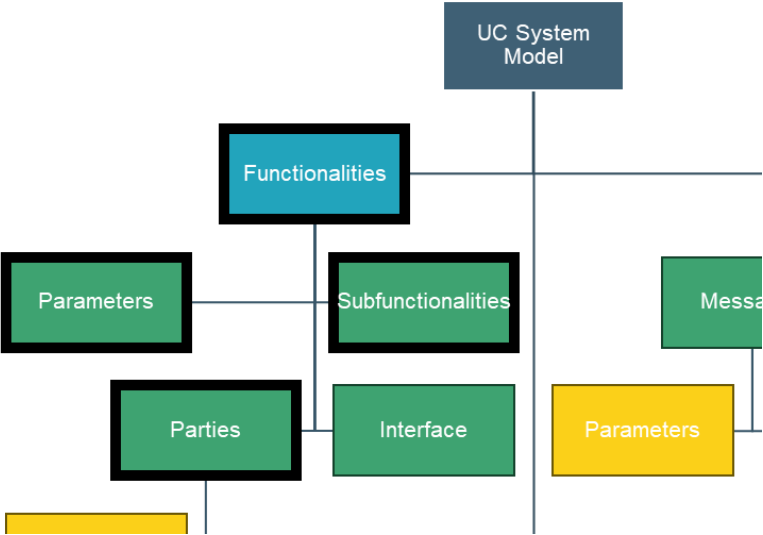
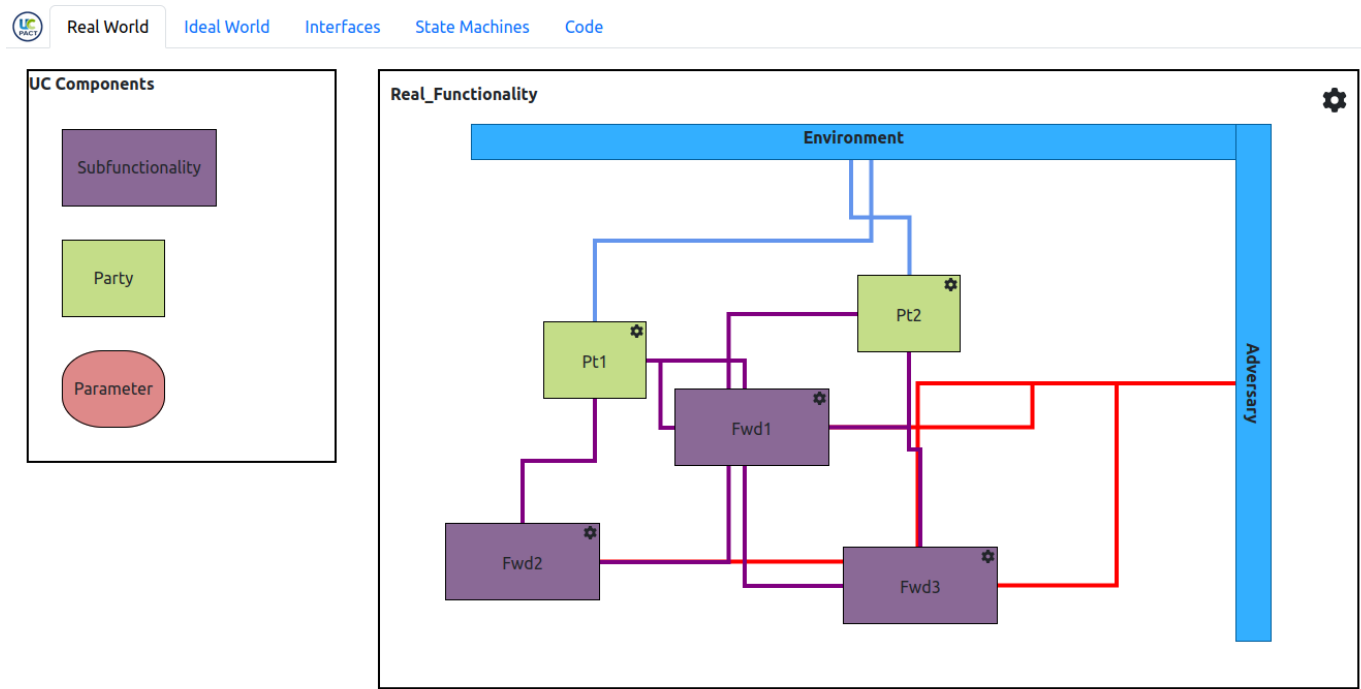
Delete

Save Changes

Close

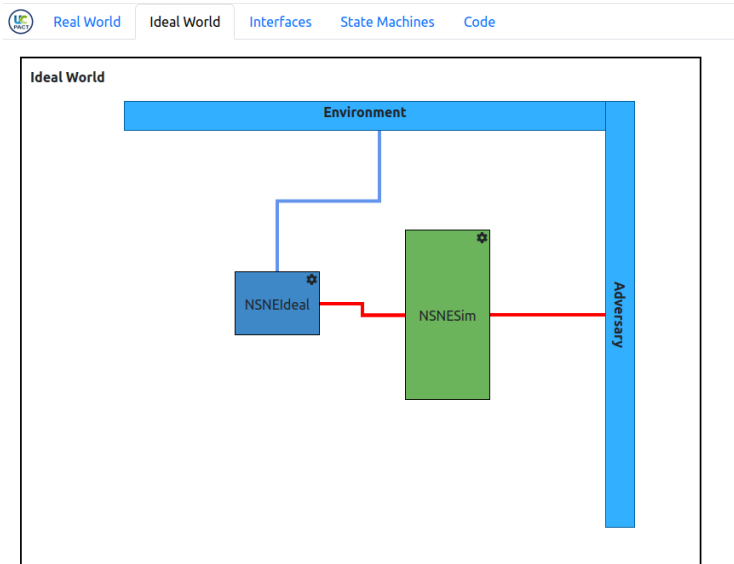
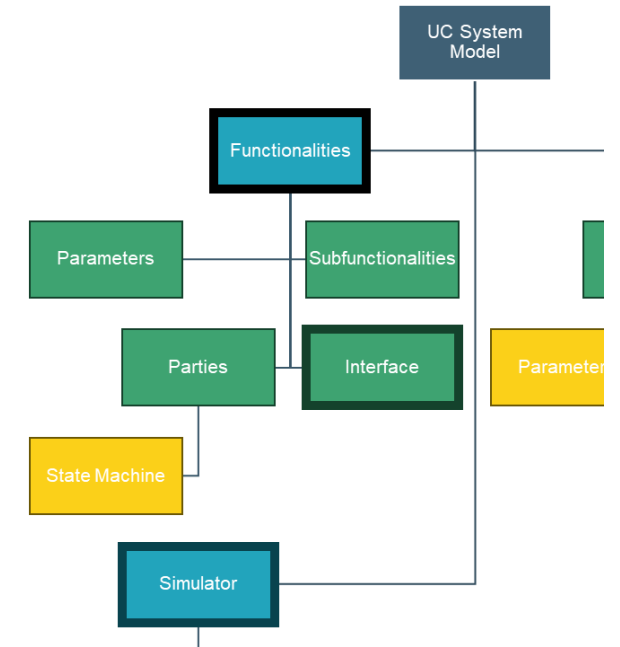


Real World Deep Dive



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



Configure IF

IF

Composite Direct Interface

Select a Composite Direct Interface...

Basic Adversarial Interface

Select an Adversarial Interface...

Save Changes

Close

Configure Sim

Sim

Real Functionality

Select a Real Functionality...

Basic Adversarial Interface

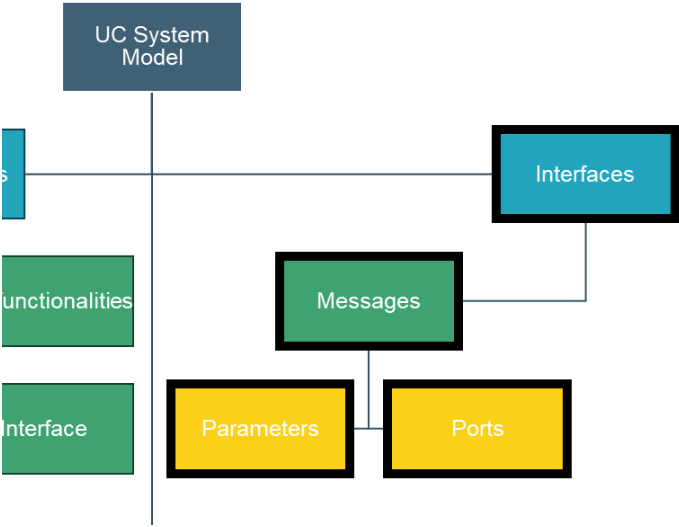
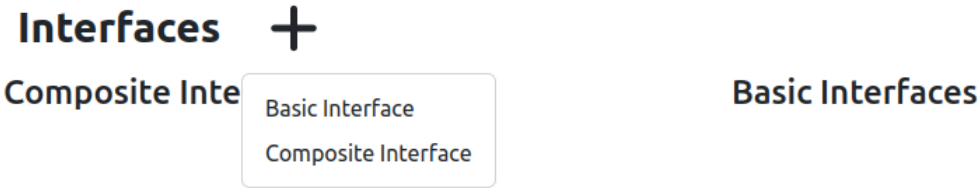
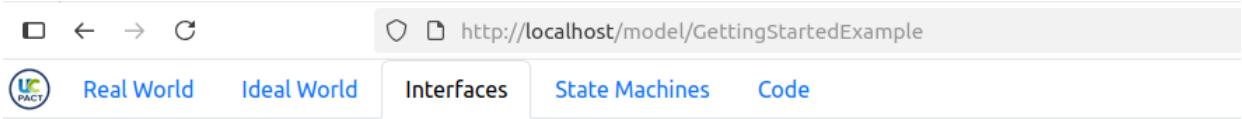
Select an Adversarial Interface...

Save Changes

Close

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

Basic Interfaces

Interface Name

Interface Name

Direct

Adversarial

Messages +

Message Name

Port Name

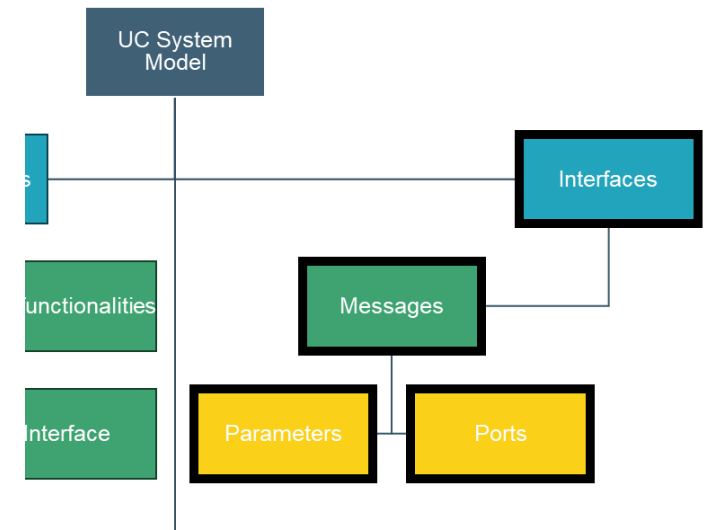
In

Out

Parameters +

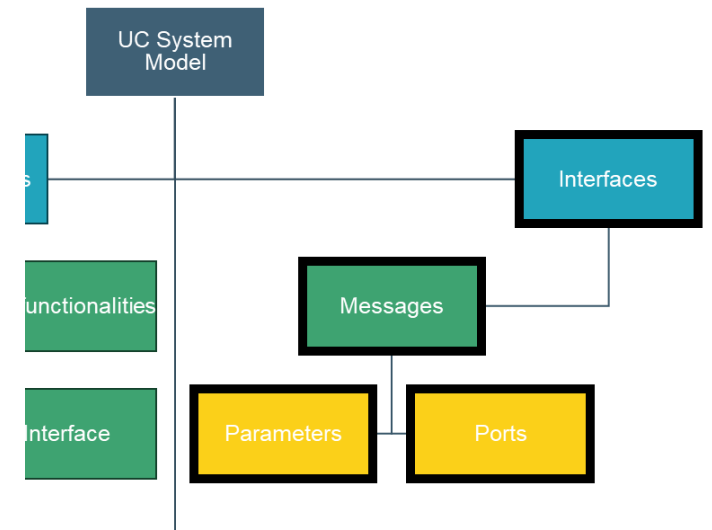
Parameter Name

Parameter Type



Interfaces Deep Dive

- › Functionalities implement composite interfaces
- › These are a collection of basic interfaces

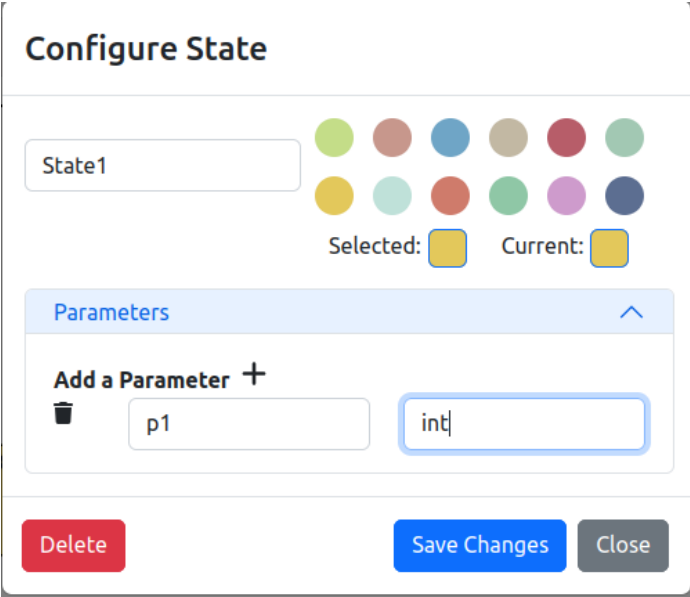
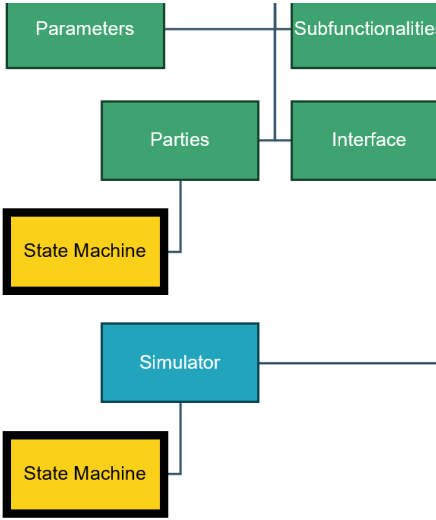
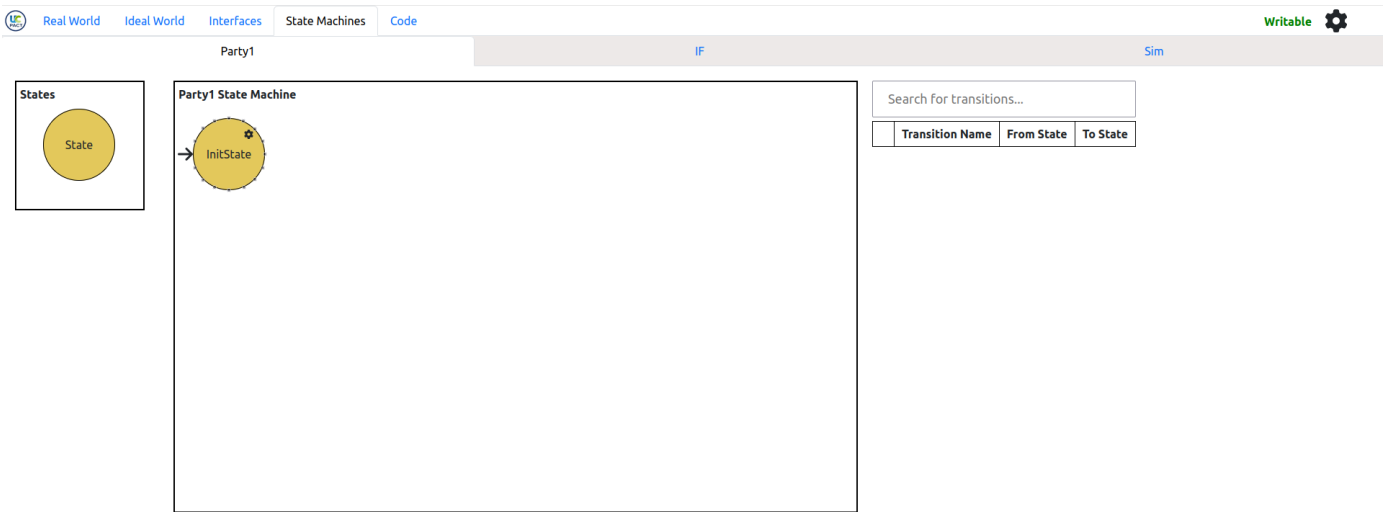


Composite Interfaces

The 'Composite Interfaces' form has a green header bar with a trash icon and the text 'Interface Name'. Below the header, there is a text input field labeled 'Interface Name'. To the right of this field are two tabs: 'Direct' (green) and 'Adversarial' (red). Below the tabs, the text 'Basic Interfaces +' is displayed. Underneath, there is another text input field labeled 'Instance Name' with a trash icon to its left. To the right of the 'Instance Name' field is a dropdown menu labeled 'Select an Interfa...' with a downward arrow.

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

Real World

Ideal World

Interfaces

State Machines

Code

Pt1

Pt2

NSNEIdeal

NSNESim

States

State

NSNEIdeal State Machine

WaitRequest

WaitSim1

WaitAccept

WaitSim2

WaitAcknowledge

WaitSim3

Final

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

Configure Transition

T1

InitState

CID.BD1.bm1

CID.BD1.bm2

State1

Target Port

Guard Description

To State Arguments

p1 (int)

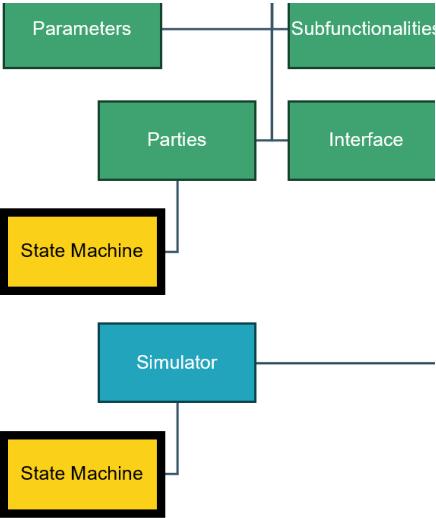
Argument Val

Out Message Arguments

Delete

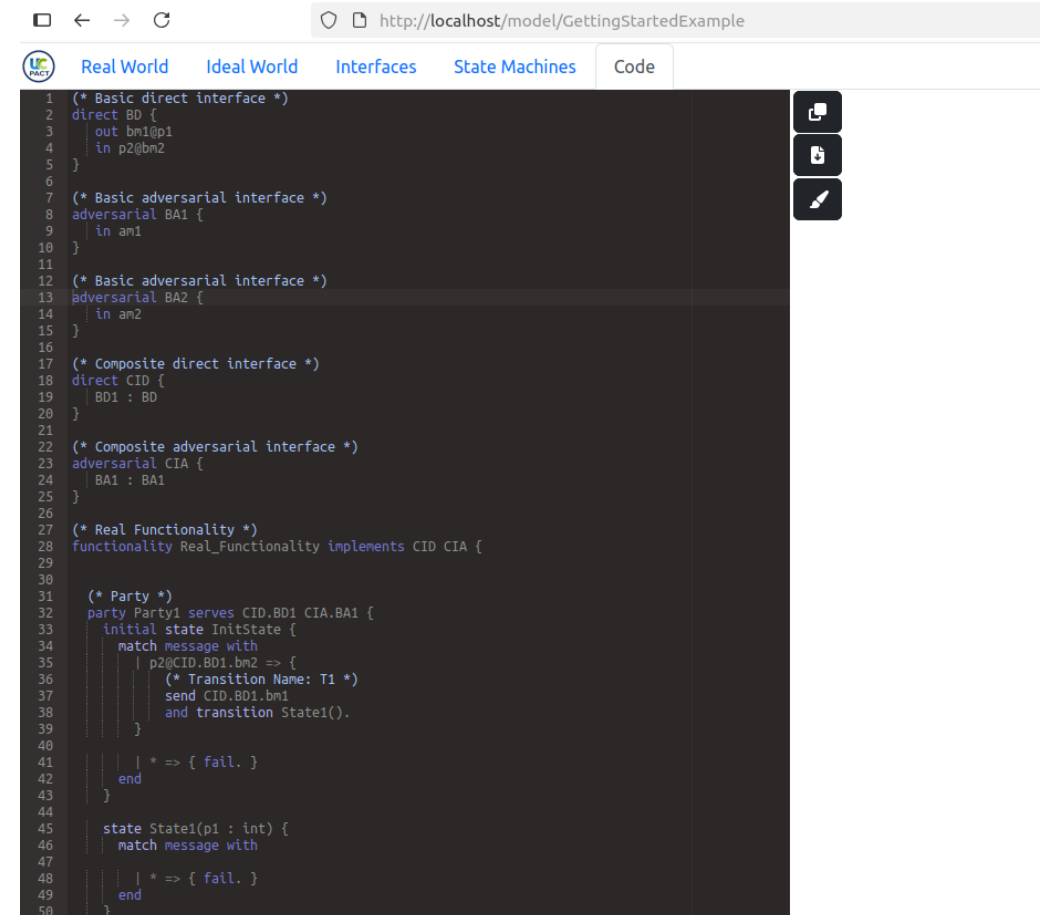
Save Changes

Close



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

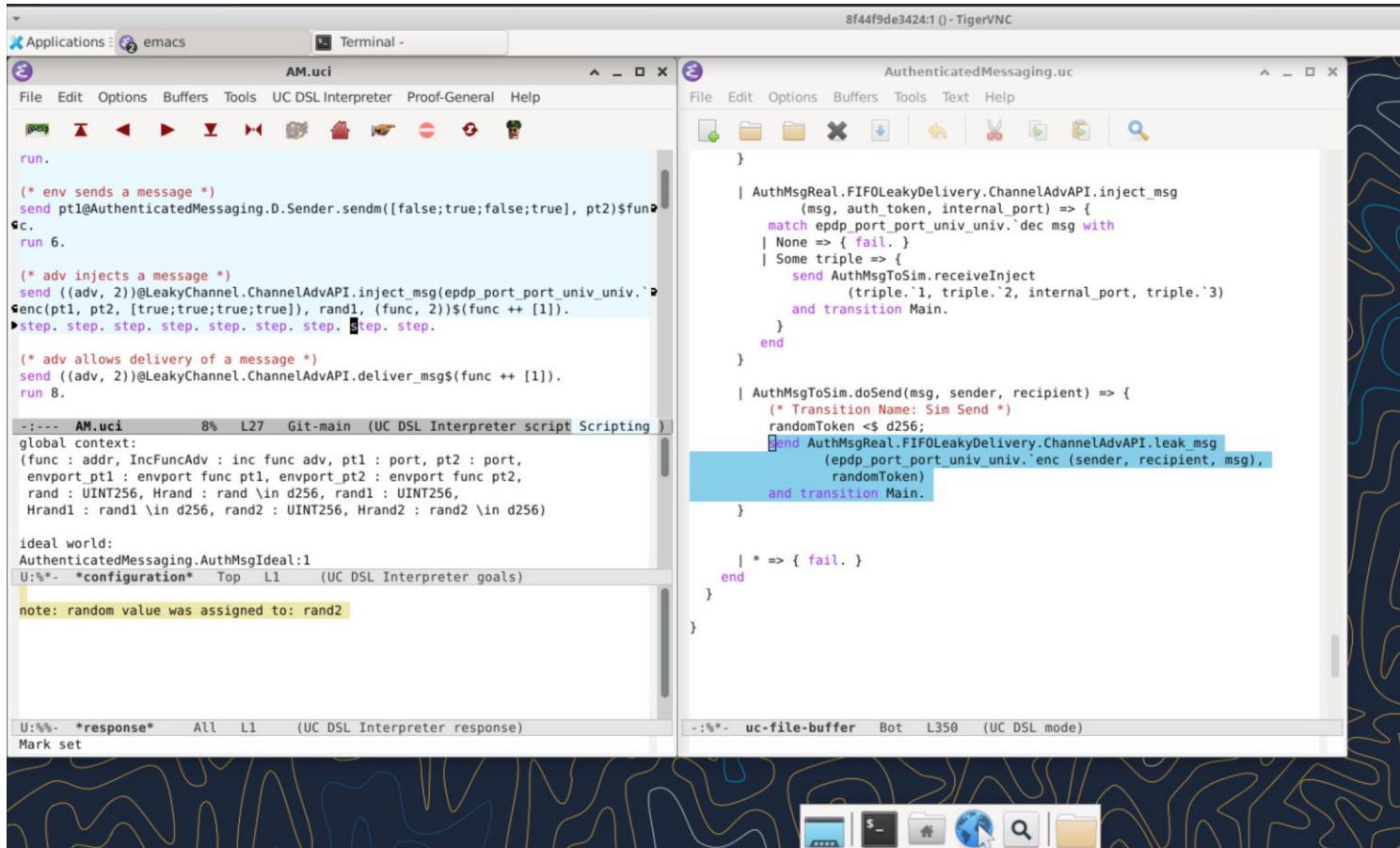


The screenshot shows a web browser window with the address bar displaying `http://localhost/model/GettingStartedExample`. The browser has several tabs: "Real World", "Ideal World", "Interfaces", "State Machines", and "Code". The "Code" tab is active, showing a code editor with generated code. The code is written in a syntax that resembles a combination of JSON and a state machine language. It defines several interfaces and a party.

```
1 (* Basic direct interface *)
2 direct BD {
3   out bm1@p1
4   in p2@bm2
5 }
6
7 (* Basic adversarial interface *)
8 adversarial BA1 {
9   in am1
10 }
11
12 (* Basic adversarial interface *)
13 adversarial BA2 {
14   in am2
15 }
16
17 (* Composite direct interface *)
18 direct CID {
19   BD1 : BD
20 }
21
22 (* Composite adversarial interface *)
23 adversarial CIA {
24   BA1 : BA1
25 }
26
27 (* Real Functionality *)
28 functionality Real_Functionality implements CID CIA {
29
30
31 (* Party *)
32 party Party1 serves CID.BD1 CIA.BA1 {
33   initial state InitState {
34     match message with
35     | p2@CID.BD1.bm2 => {
36       (* Transition Name: T1 *)
37       send CID.BD1.bm1
38       and transition State1().
39     }
40
41     | * => { fail. }
42   end
43 }
44
45 state State1(p1 : int) {
46   match message with
47
48   | * => { fail. }
49   end
50 }
```

What next?

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



```
run.

(* env sends a message *)
send pt1@AuthenticatedMessaging.D.Sender.sendm([false;true;false;true], pt2)$func
c.
run 6.

(* adv injects a message *)
send ((adv, 2))@LeakyChannel.ChannelAdvAPI.inject_msg(epdp_port_port_univ_univ.`
enc(pt1, pt2, [true;true;true]), rand1, (func, 2))$(func ++ [1]).
step. step. step. step. step. step.

(* adv allows delivery of a message *)
send ((adv, 2))@LeakyChannel.ChannelAdvAPI.deliver_msg$(func ++ [1]).
run 8.

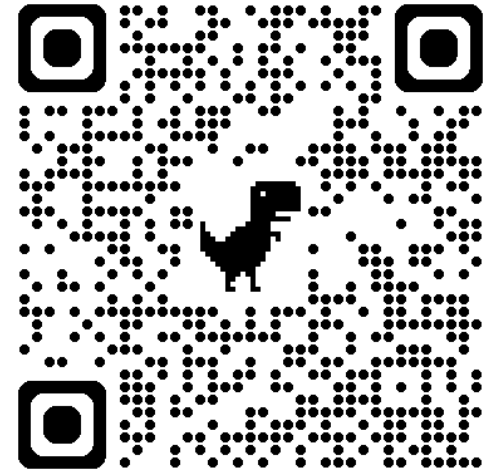
--- AM.uci 8% L27 Git-main (UC DSL Interpreter script Scripting)
global context:
(func : addr, IncFuncAdv : inc func adv, pt1 : port, pt2 : port,
envport_pt1 : envport func pt1, envport_pt2 : envport func pt2,
rand : UINT256, Hrand : rand \in d256, rand1 : UINT256,
Hrand1 : rand1 \in d256, rand2 : UINT256, Hrand2 : rand2 \in d256)

ideal world:
AuthenticatedMessaging.AuthMsgIdeal:1
U:%*- *configuration* Top L1 (UC DSL Interpreter goals)

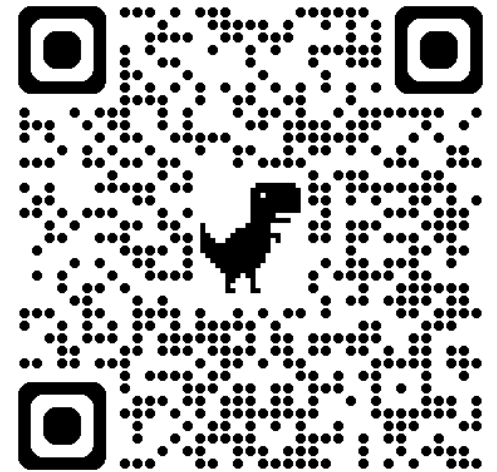
note: random value was assigned to: rand2

U:%*- *response* All L1 (UC DSL Interpreter response)
Mark set
```

To direct input to this VM, click inside or press Ctrl+G.



ucdsl-tools-docker



Easy UC User Guide

Questions?

┐

