Overview of Cadena's Architecture Definition Language and Meta-modeling Framework

SAnToS Laboratory, Kansas State University, USA

http://cadena.projects.cis.ksu.edu

Georg Jung
Adam Childs
Jesse Greenwald
John Hatcliff
Matt Hoosier
Alley Stoughton

Support

US Army Research Office (ARO)
US National Science Foundation (NSF)
US Department of Defense
Advanced Research Projects Agency (DARPA)
Air Force Office of Scientific Research (AFOSR)
IBM Eclipse
Lockheed Martin
Rockwell-Collins ATC

Distinctions of Cadena Modeling

- Architectural definition framework that matches the principle roles in *product-line development processes*
- Emphasis on *software architecture styles* as "first-class citizens" that can be
  - operated on in various ways
    - extended, transformed, combined, etc.
  - arranged in hierarchies to formally capture architecture refinement
  - used to define *domain-specific modeling environments*
- Core "calculus" of architecture *structures*
  - no built-in notion of semantics
  - semantics and interpretations of structures are added through *plug-ins*

…all of the above supported by advanced type systems

Theme

Structure according to product-line developer roles

Product-line architect

…defines component model, recognizes commonalities/variance across products, captures commonalities in infrastructure, defines process plug-ins

Component developer

…builds reusable components, refines general components to product-specific ones, contributes to component libraries

Component integrator

…connects components, configures infrastructure according set strategies, achieves global functional non-functional properties

Cadena Meta-modeling Framework

Meta-modeling structure

Type Theory

Kinds

Types

Values

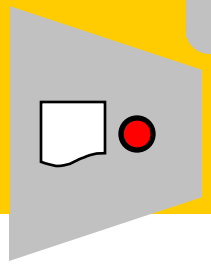
Define language for interfaces

Define language for connectors

Define language for components

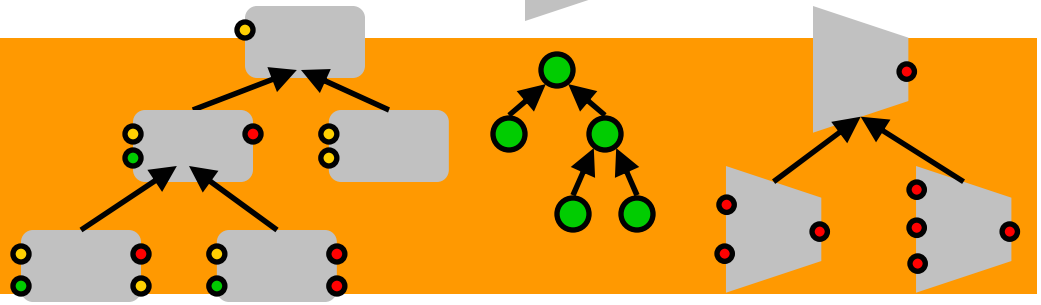
Style Tier

Define Architectures
-- domain specific languages for defining component types



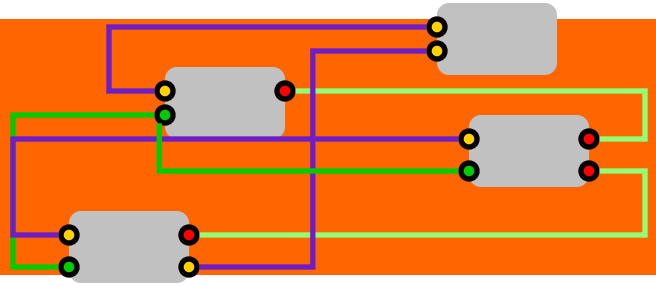
Module Tier

Define Component and Interface Types within an Architecture



Scenario Tier

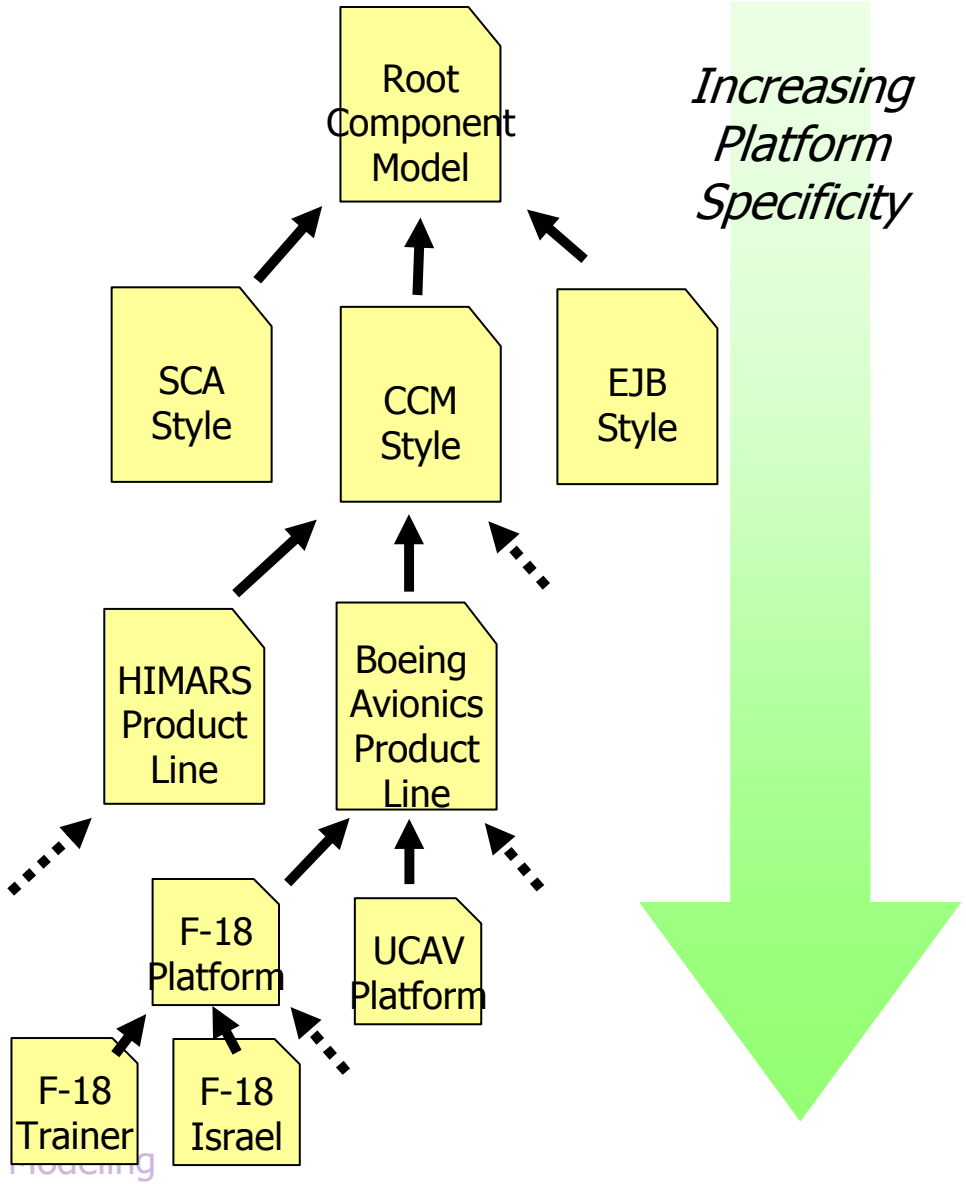
Allocate component instances, connect, model programming within an architecture



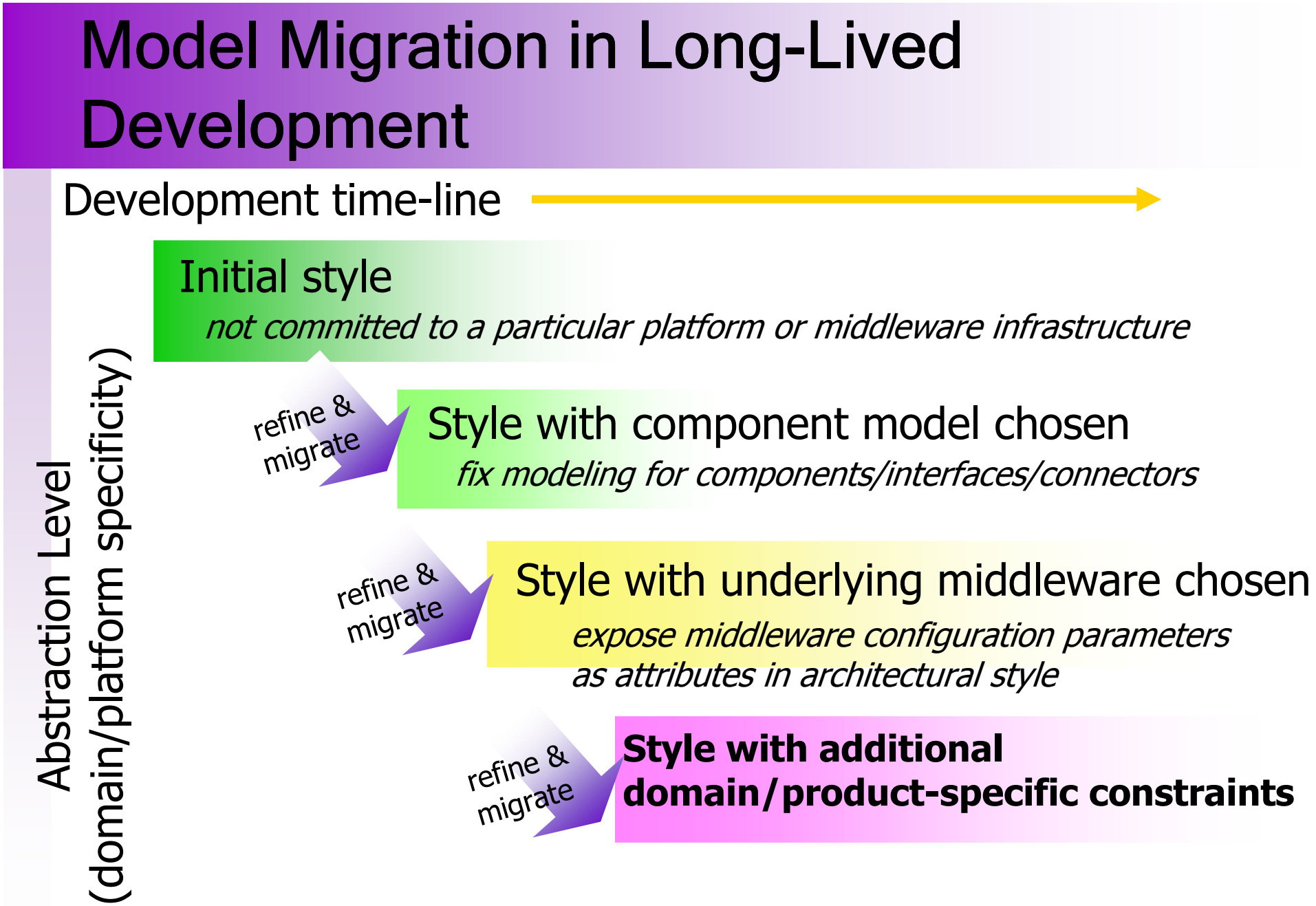
Hierarchical Arrangement of Styles

Work with styles as "first-class citizens"

- Multiple component models
- Extended, e.g., through inheritance
- Capture degrees of specificity within product-line architecture
- Incremental addition of domain characteristics
- Arranged in hierarchies to formally capture architecture refinement



Model Migration in Long-Lived Development



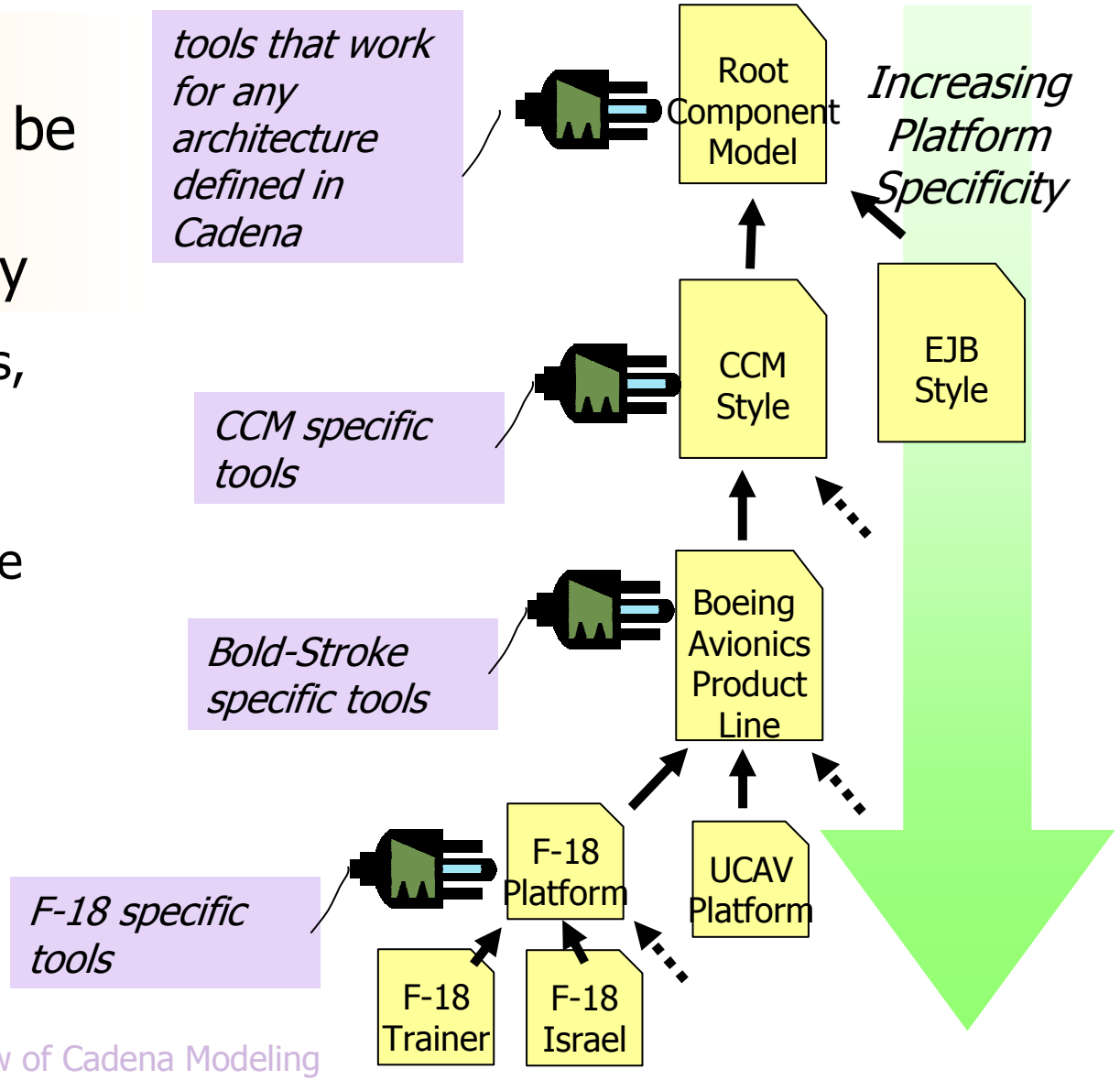
Minimal Core Architectural Framework

- It is impossible to predict all the different capabilities that might be needed in a realistic component-based development environment
- Designing a single general purpose ADL will result usually result in a bulky language that still doesn't do everything you want it to
- Instead, use a minimal meta-modeling framework that allows one to define domain-specific modeling languages where domain-specific analyses and tools can be added as plug-ins

Style-(domain)-specific Semantics & Interpretation

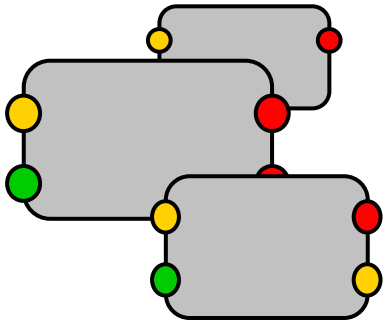
Pluggable tool components that can be re-used according to architectural hierarchy

- analyses, visualizations, design advice, state-space exploration & conformance checkers
- use Eclipse architecture as basis of clean & flexible plug-in design



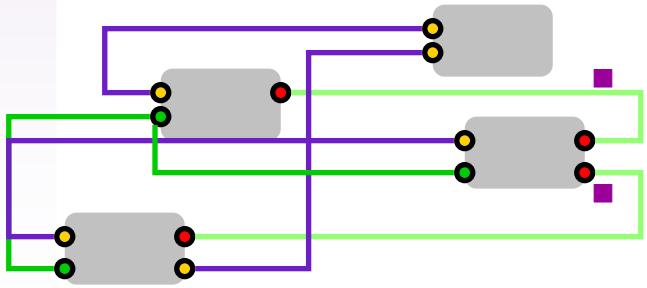
Model Interpreter Plug-ins

Examples of Module-Level Plug-ins



- Code generators that generate implementation skeletons from interface descriptions
- Editors that allow one to write pre/post-conditions and invariants at component interfaces
- Verification engines that check that implementations conform to interface specifications

Examples of Scenario-Level Plug-ins



- Code generators that generate "glue code" for realizing connections between components
- Visualizers that allow you to browse paths between components
- Editors that allow you to write end-to-end temporal specifications on system behavior
- Model checkers and simulators that check end-to-end execution behavior against specifications

Supported by Advanced Type Systems

8.2.1 Identifier list typing rules for $\rightarrow_i^a$, $\rightarrow_i^r$, and $\rightarrow_i^p$

Interface type identifier list.

$$\frac{}{id, \hat{\alpha}, \psi \vdash \epsilon \rightarrow_i^a \hat{\alpha}} \quad (50)$$

$$\frac{t_1, t_2 \in \text{Identifier-list}, \quad id, \hat{\alpha}_0, \psi \vdash t_1 \rightarrow_i^a \hat{\alpha}_1, \quad id, \hat{\alpha}_1, \psi \vdash t_2 \rightarrow_i^a \hat{\alpha}_2}{id, \hat{\alpha}_0, \psi \vdash t_1, t_2 \rightarrow_i^a \hat{\alpha}_2} \quad (51)$$

$$\frac{\psi \vdash id_0 : (id_1\ \hat{\alpha}_1), \quad \text{dom}(\hat{\alpha}_0) \cap \text{dom}(\hat{\alpha}_1) = \emptyset}{id_1, \hat{\alpha}_0, \psi \vdash id_0 \rightarrow_i^a \hat{\alpha}_0 \cup \hat{\alpha}_1} \quad (52)$$

Component type identifier list.

$$\frac{}{id, \hat{\pi}, \psi \vdash \epsilon \rightarrow_i^p \hat{\pi}} \quad (53)$$

$$\frac{t_1, t_2 \in \text{Identifier-list}, \quad id, \hat{\pi}_0, \psi \vdash t_1 \rightarrow_i^p \hat{\pi}_1, \quad id, \hat{\pi}_1, \psi \vdash t_2 \rightarrow_i^p \hat{\pi}_2}{id, \hat{\pi}_0, \psi \vdash t_1, t_2 \rightarrow_i^p \hat{\pi}_2} \quad (54)$$

$$\frac{\psi \vdash id_0 : (id_1\ \hat{\pi}_1), \quad \text{dom}(\hat{\pi}_0) \cap \text{dom}(\hat{\pi}_1) = \emptyset}{id_1, \hat{\pi}_0, \psi \vdash id_0 \rightarrow_i^p \hat{\pi}_0 \cup \hat{\pi}_1} \quad (55)$$

8.2.2 Module typing rules for $\rightarrow_t$

Interface type of kind id_0

$$\frac{\kappa \vdash id_0 : (\mathbf{ik}\ \bar{\alpha}), \quad \begin{array}{l} id_1 \in Id, \\ id_1 \notin \text{dom}(\psi), \end{array} \quad \begin{array}{l} i \in \text{Identifier-list}, \\ id_0, \emptyset, \psi \vdash i \rightarrow_i^a \hat{\alpha}_0, \end{array} \quad \begin{array}{l} t \in \text{Interface-Type-Body}, \\ \bar{\alpha}, \hat{\alpha}_0 \vdash t \rightarrow_{\hat{a}} \hat{\alpha}_1, \end{array} \quad \mathbf{C}^{\mathbf{it}}(\bar{\alpha}, \hat{\alpha}_1)}{\kappa, \psi \vdash id_0\ id_1\ \mathbf{extends}\ i\ \{\ t\ \} \rightarrow_t \psi[id_1 : (id_0\ \hat{\alpha}_1)]} \quad (56)$$

Component type of kind id_0

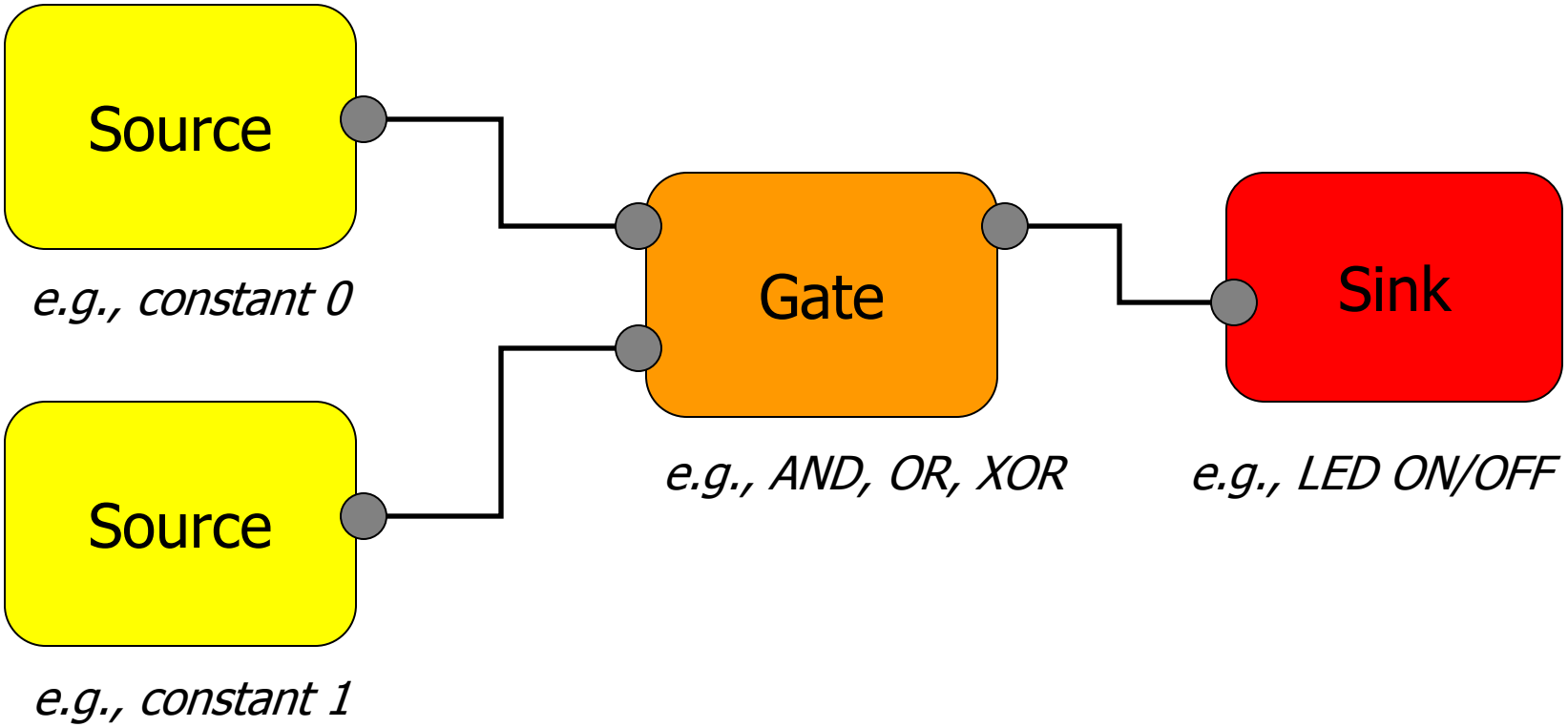
$$\frac{\kappa \vdash id_0 : (\mathbf{ck}\ \bar{\pi}), \quad \begin{array}{l} id_1 \in Id, \\ id_1 \notin \text{dom}(\psi), \end{array} \quad \begin{array}{l} i \in \text{Identifier-list}, \\ id_0, \emptyset, \psi \vdash i \rightarrow_i^p \hat{\pi}_0, \end{array} \quad \begin{array}{l} t \in \text{Connector-Type-Body}, \\ \bar{\pi}, \hat{\pi}_0 \vdash t \rightarrow_{\hat{p}} \hat{\pi}_1, \end{array} \quad \mathbf{C}^{\mathbf{ct}}(\bar{\pi}, \hat{\pi}_1)}{\kappa, \psi \vdash id_0\ id_1\ \mathbf{extends}\ i\ \{\ t\ \} \rightarrow_t \psi[id_1 : (id_0\ \hat{\pi}_1)]} \quad (57)$$

...rigorous foundations for Cadena's three-tiered modeling framework

Outline

- A three-tiered approach to define architectures and domain specific modeling environments
- Using Cadena's three modeling layers
  - Defining architectural styles
  - Defining component types and interface types
  - Allocating and connecting component instances
- Advanced Concepts (lab sessions)
  - Style inheritance
  - Component Nesting
  - Attribute Attachment

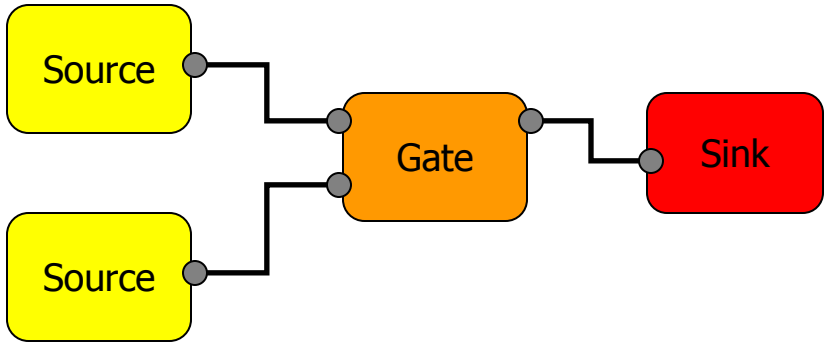
Example Domain: Logical Gates



Domain Analysis/Engineering

Product-line architect

Analyze the domain for the purpose of defining a domain-specific modeling framework



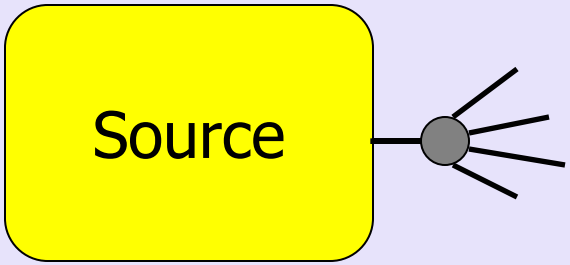
Example Analysis

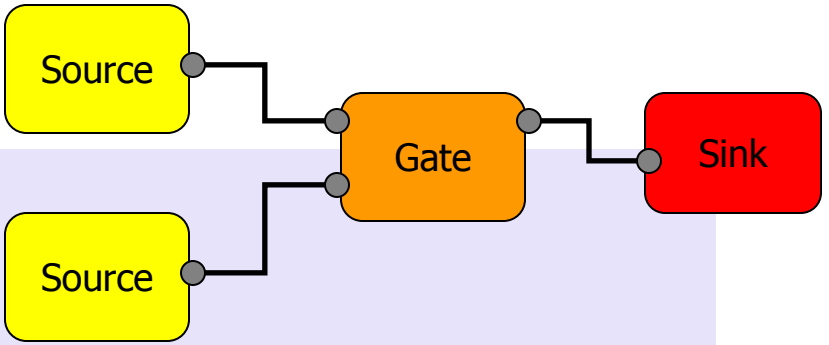
- Determine commonalities/variances in domain
  - what common shapes of components exist?
  - what are possible structures of interfaces between components
- Determine the constraints on the uses of each class of shapes?
  - how many ports can occur on components (multiplicity)?
  - how many connections can be made on a single port (multiplexity)?

Domain Analysis/Engineering

Domain Analysis

Product-line architect





No input

One output → **Multiplicity = 1**

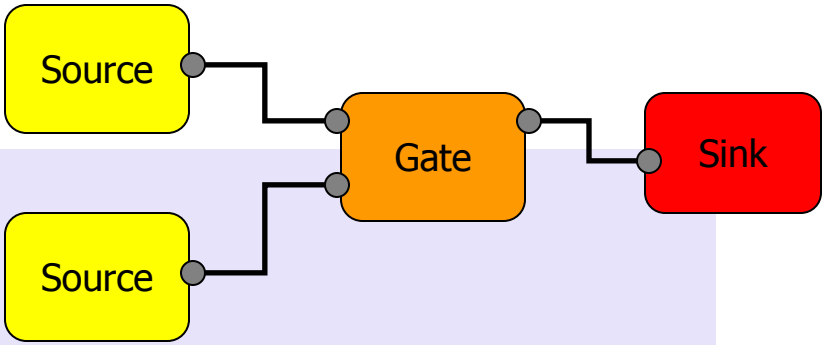
Arbitrary Connections → **Multiplexity = [0..*]**

Shapes of a Bit-Source-Component

Domain Analysis/Engineering

Domain Analysis

Product-line architect



Multiple inputs → **Multiplicity = [1..*]**

One Connection each → **Multiplexity = 1**

One Output → **Multiplicity = 1**

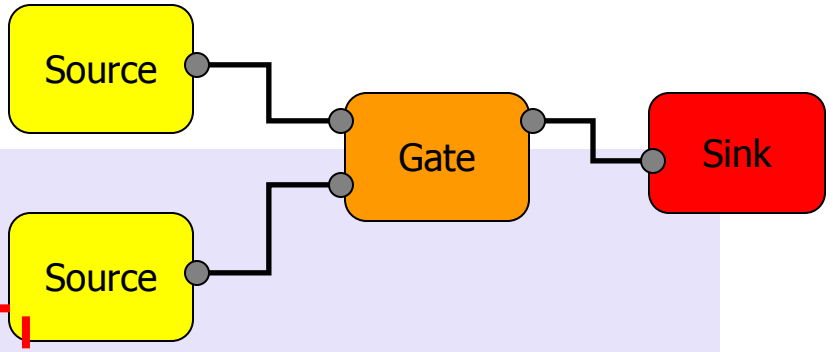
Arbitrary Connections → **Multiplexity = [0..*]**

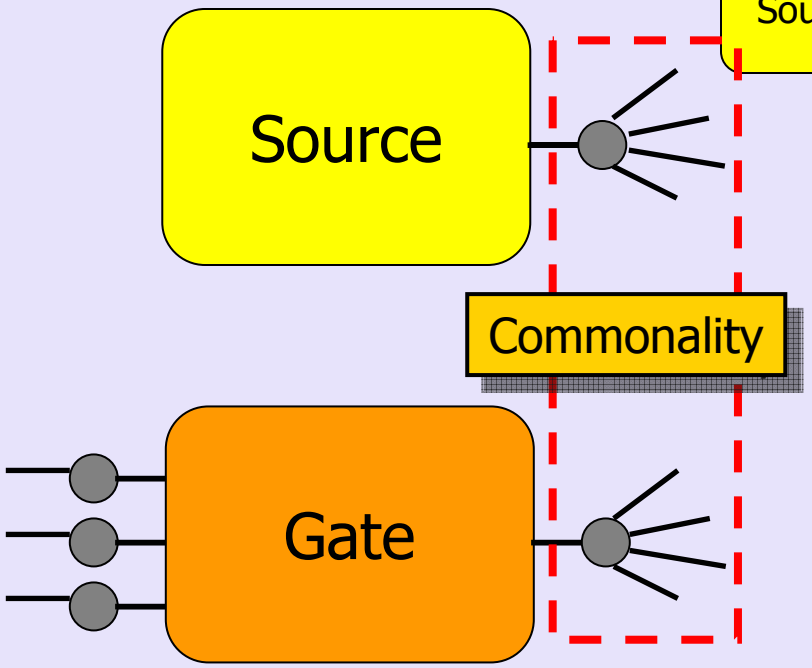
Shapes of a Gate-Component

Domain Analysis/Engineering

Domain Analysis

Product-line architect





Observation:

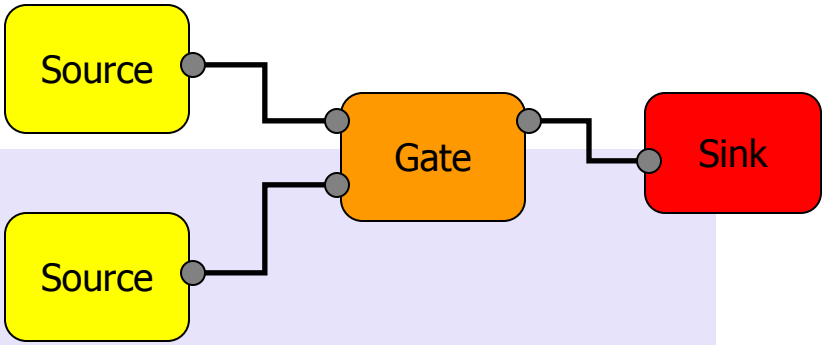
A Gate's properties on output ports are in *common* with Source but *vary* with the additional capacity to have multiple inputs

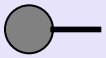
Relationship between Source and Gate

Domain Analysis/Engineering

Domain Analysis

Product-line architect





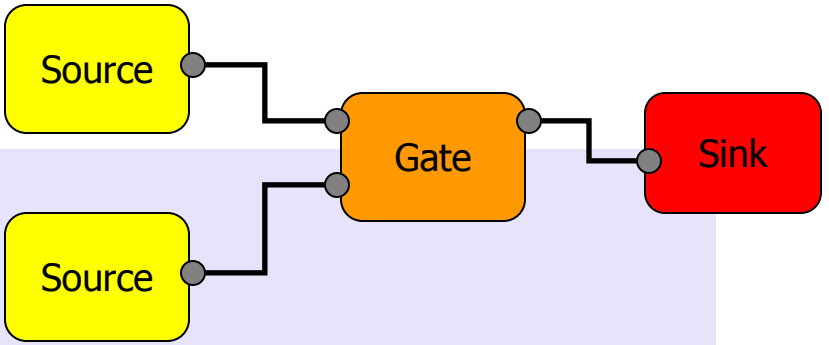
There is no distinction between different interfaces (plugs) that represent connection points on components. All connection points represent communication of 0/1 bit-levels.

Shapes of Interfaces

Domain Analysis/Engineering

Domain Analysis

Product-line architect



There is no distinction between different connectors (wires) that represent interactions between components. All connectors represent communication of 0/1 bit-levels.

Shapes of Connectors

Formalizing Domain Analysis Results as Cadena Styles

Product-line architect

The product-line architect will use Cadena Styles to define a domain-specific modeling language.

- The style tier is the *meta-modeling* layer in the sense that
  - it is used to define modeling languages that developers use to build component and interface types.
- A style describes the possible shapes/forms that component types, connectors, and interfaces can have.
- A style defines abstractions that represent the basic kinds of software entities and services available on a particular platform.

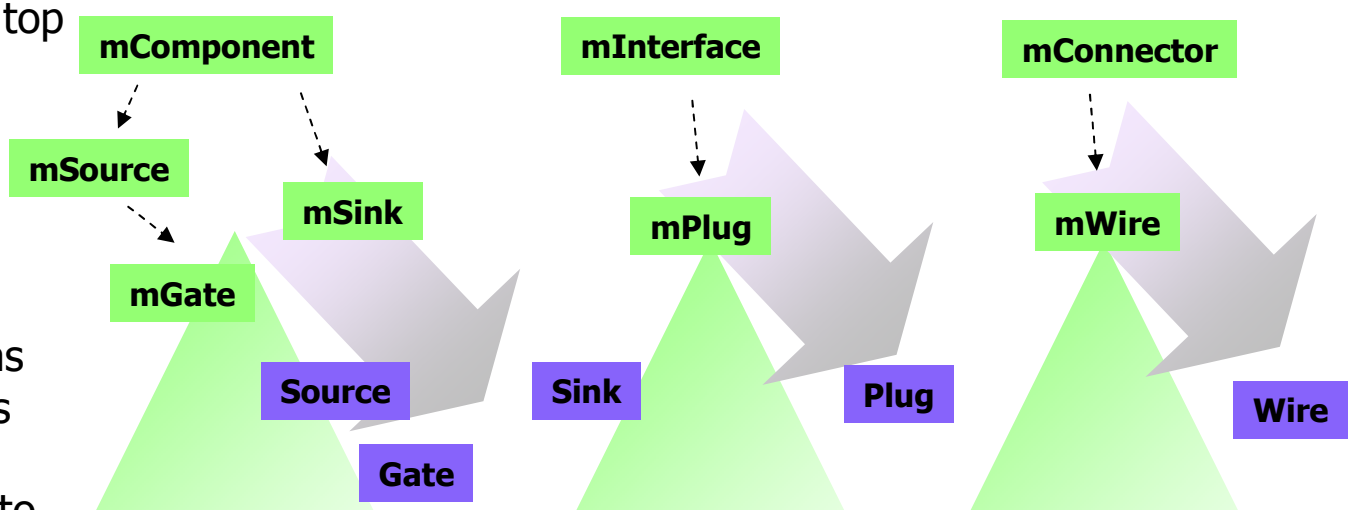
Inheritance-based Meta-modeling

Designing Kinds – Languages for building types

Cadena has three basic architecture entities (kinds)

Components Interfaces Connectors

Three root meta-kinds at top of inheritance hierarchy



Derive meta-kinds for current domain

"Export" **meta-kinds** as **kinds** which developers use as languages/frameworks to create types

Continue to derive meta-kinds into type languages for model platforms with greater specificity

We now illustrate how we use Cadena meta-kinds and kinds to build languages for defining interface, component, and connector types in the Logical Gates domain.

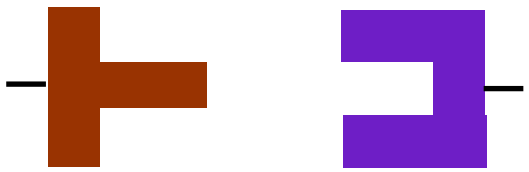
Modeling: Plug Interfaces

Components of the Logical Gates example have one sort of interaction point or ***interface kind****, the* Plug*. To model this, we derive a meta kind from mInterface and export* Plug*.*

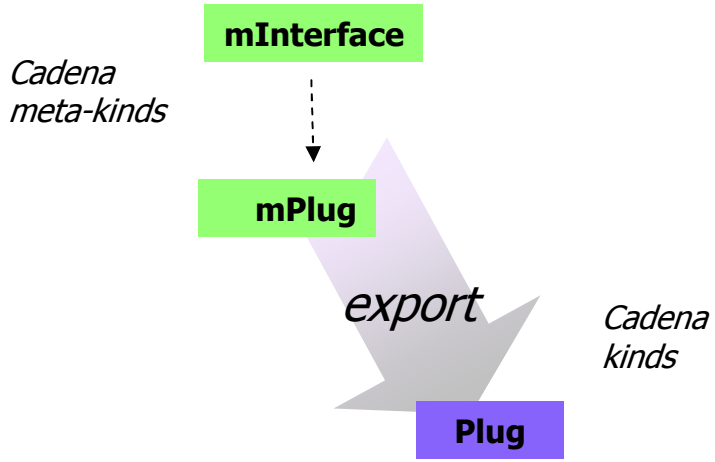
Logical Gates

The Plug interface

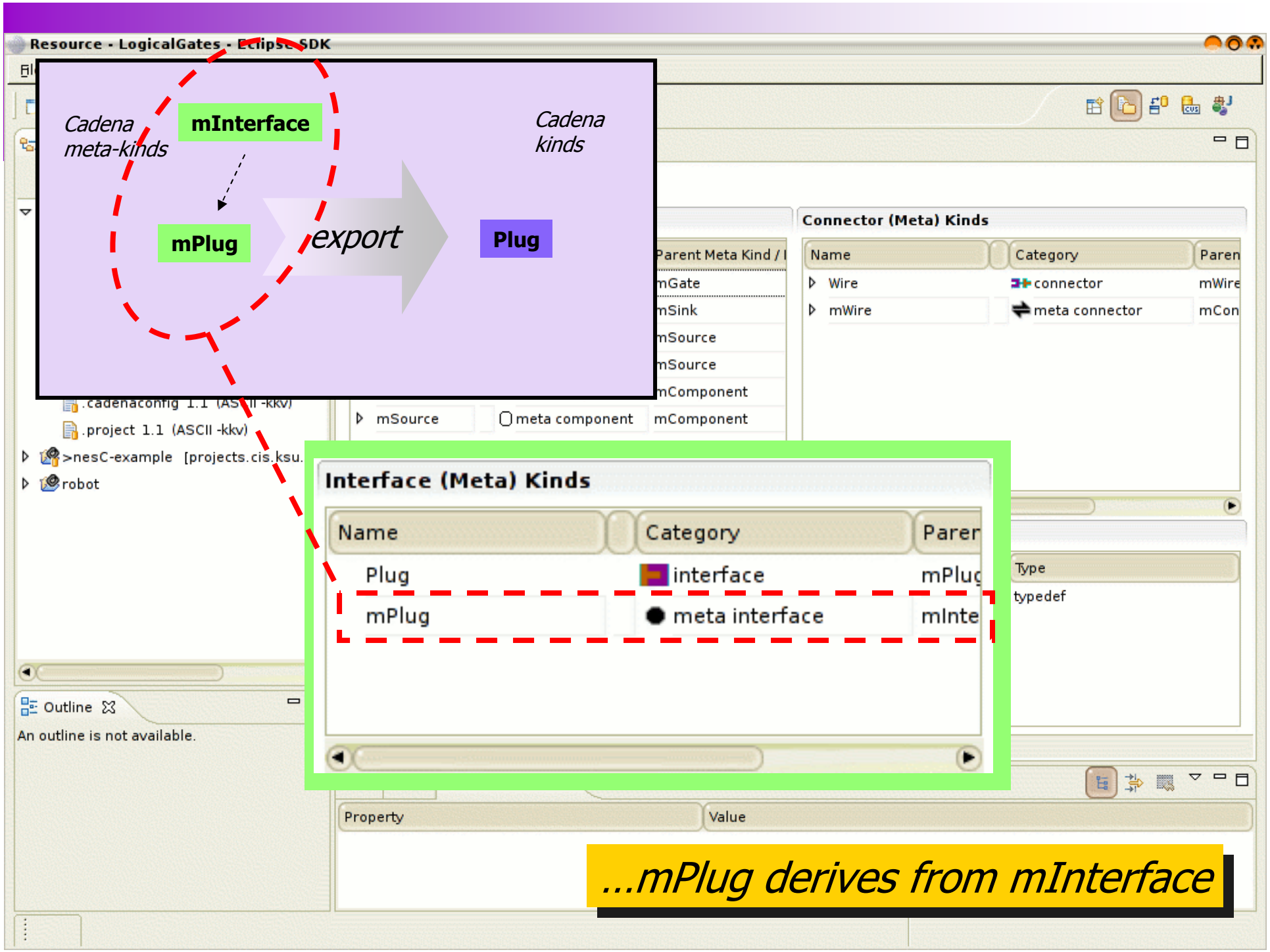
+ Side - Side

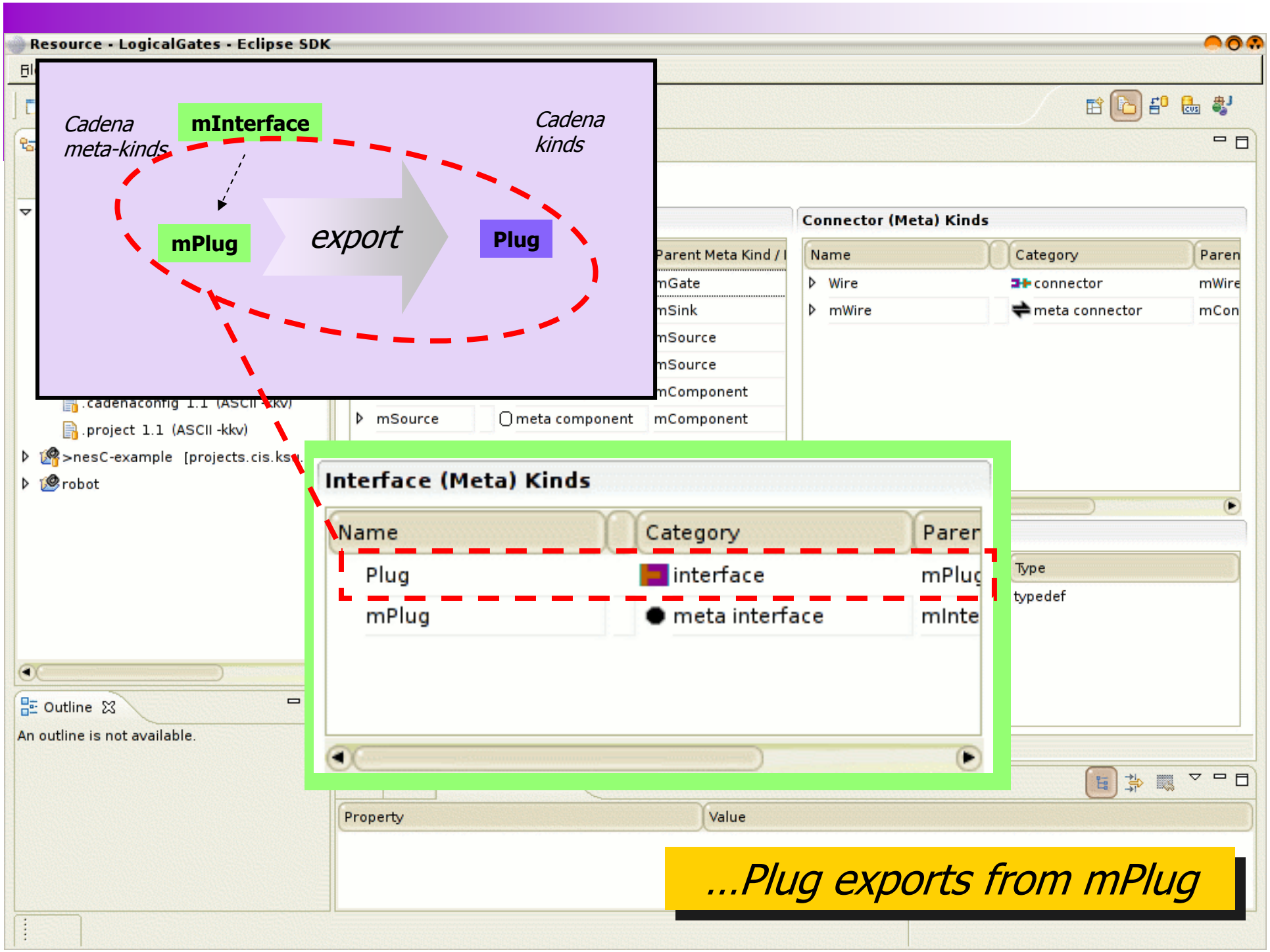


Logical Gates Style in Cadena



...illustrate this within the Cadena editors

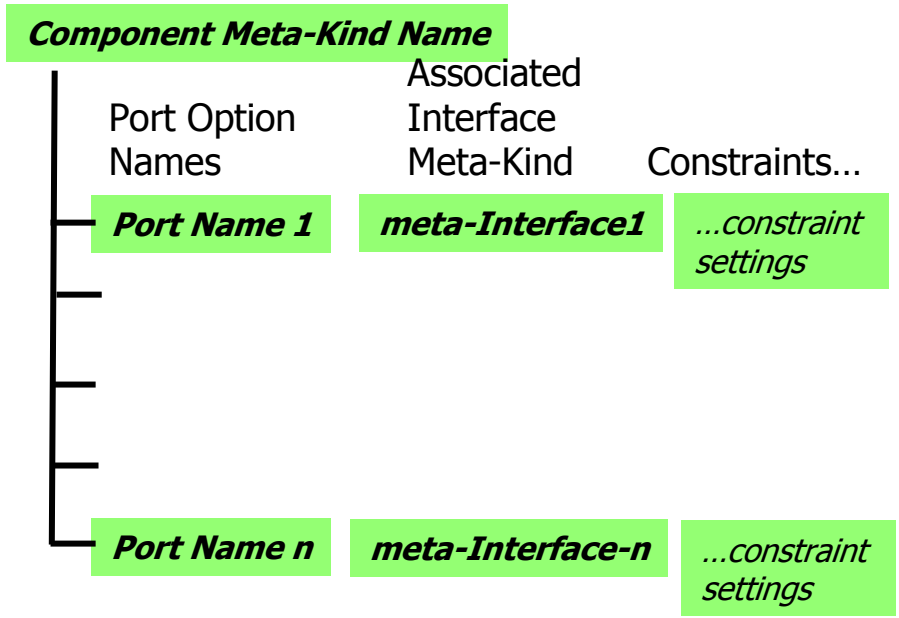

Cadena meta-kinds
mInterface
mPlug
export
Cadena kinds
Plug
Connector (Meta) Kinds
Name
Category
Wire
connector
mWire
meta connector
Interface (Meta) Kinds
Plug
interface
mPlug
meta interface
Outline
An outline is not available.
Property
Value
...mPlug derives from mInterface


Resource - LogicalGates - Eclipse SDK
Cadena meta-kinds
mInterface
Cadena kinds
mPlug
export
Plug
Connector (Meta) Kinds
Name
Category
Wire
connector
mWire
meta connector
Interface (Meta) Kinds
Plug
interface
mPlug
meta interface
Outline
An outline is not available.
Property
Value
...Plug exports from mPlug

Modeling: Components

Cadena *component* (meta)-kinds are used to define languages for expressing possible component types within the domain.

Structure of Cadena Component (meta)-kind



PORT OPTIONS -- options for the kinds of ports that can be attached to the component (with constraints for each option)

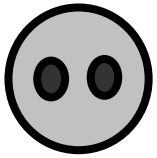
Modeling: Port Options

A *Port Option* in a Cadena component style declaration represents a capacity for declaring ports that adhere to certain constraints. Note that Cadena ports need to specify a **parity**:

"Real Life"

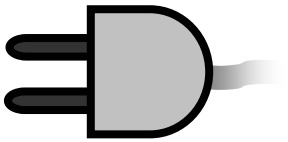
Interaction of two matching parts

Socket



Server-side protocol

Plug

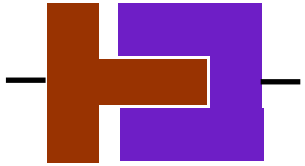


Client-side protocol

Modeling in Cadena

Interaction through shared Interface

One side *PROVIDES* the Interface

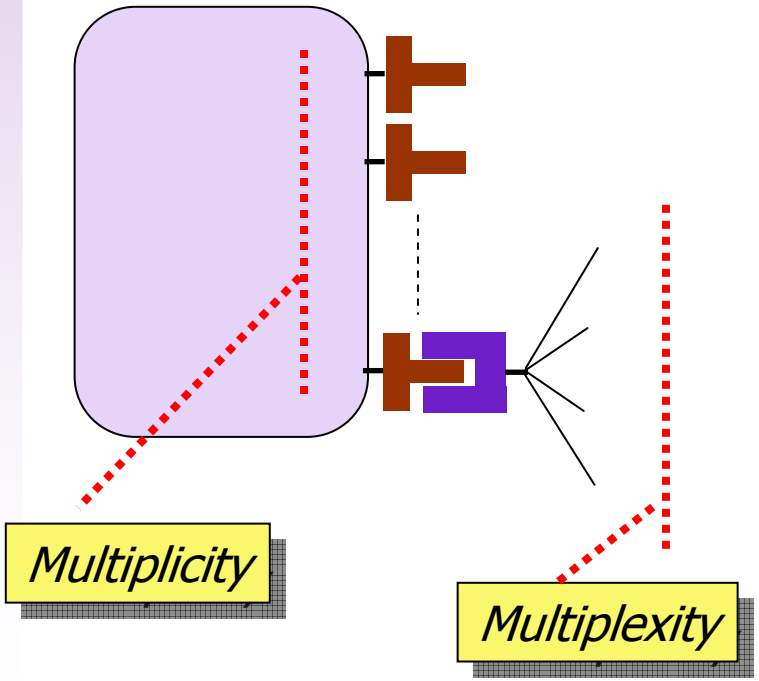


The other side *USES* the Interface

Modeling: Port Options

A *Port Option* in a Cadena component style declaration represents a capacity for declaring ports that adhere to certain constraints

Logical Gates



Logical Gates Style in Cadena

Port Option

Name: *<name of port option>*
e.g., output

Interface Meta Kind: *<what kind of interfaces can be bound>*
e.g., mPlug

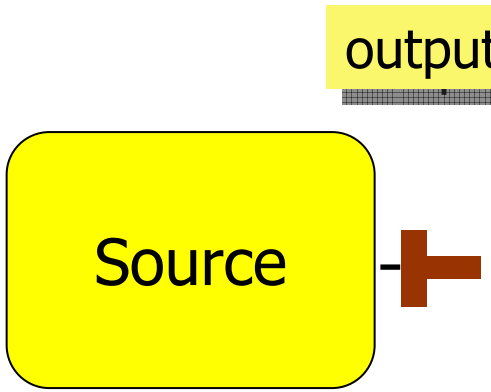
Parity: *<PROVIDES or USES>*
e.g., PROVIDES

Multiplicity: *<# ports allowed>*
e.g., 0..*, for Source and Gate: 1..1

Multiplexity: *<# connections allowed on a particular port>*
e.g., 0..1

Modeling: Components & Ports

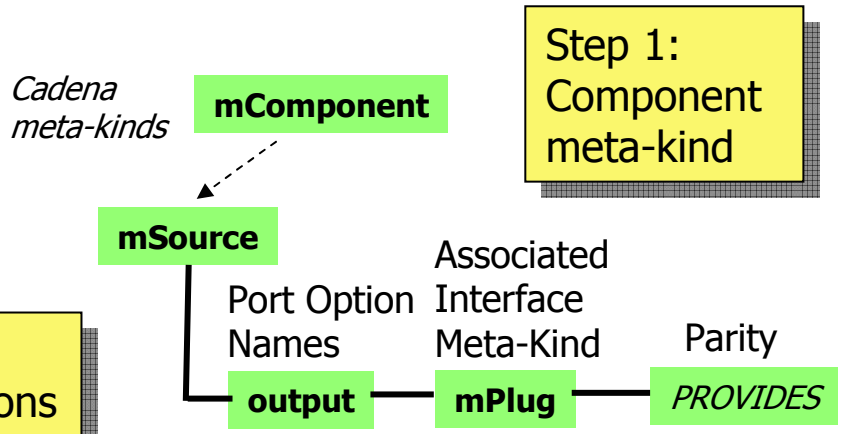
Logical Gates



Step 2:
Port Options
(excerpts)

"output" is a *architect-defined keyword* that names that particular port option.

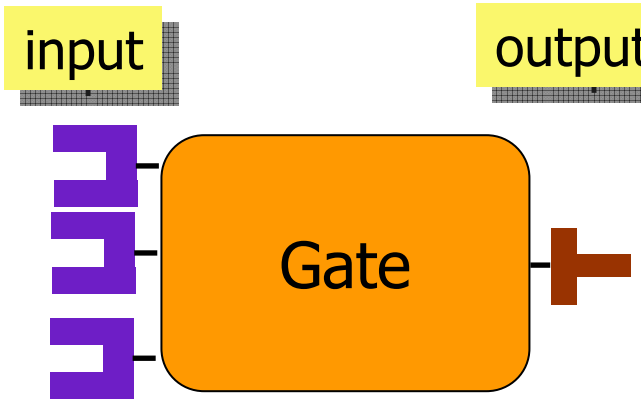
Logical Gates Modeled in Cadena



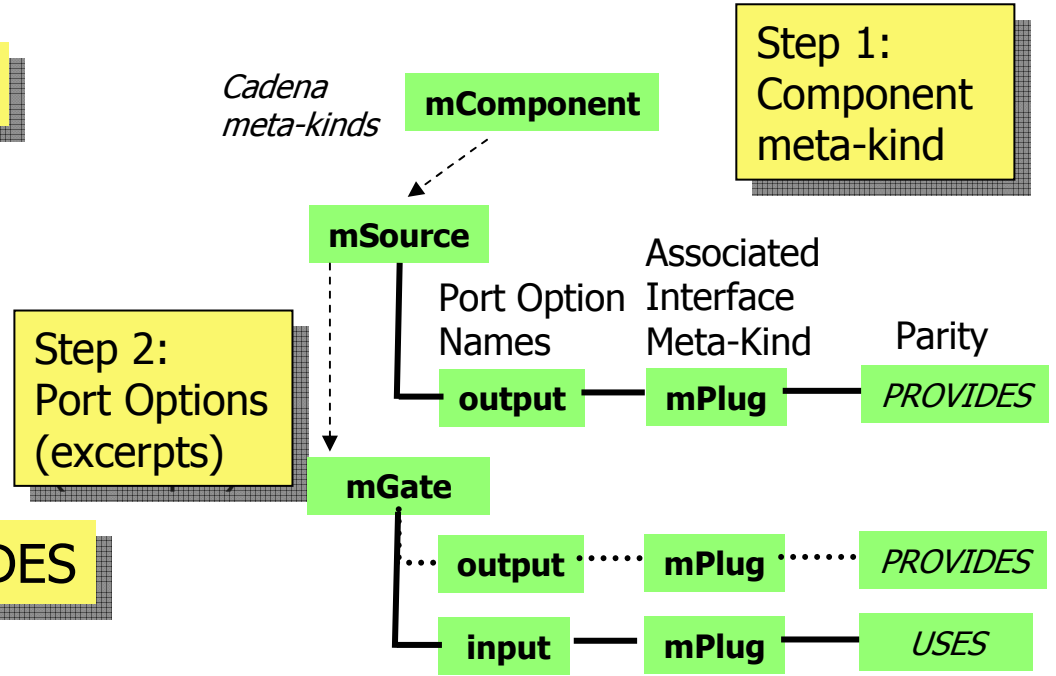
Shapes of a Logical Gate Component Interface

Modeling: Components & Ports

Logical Gates

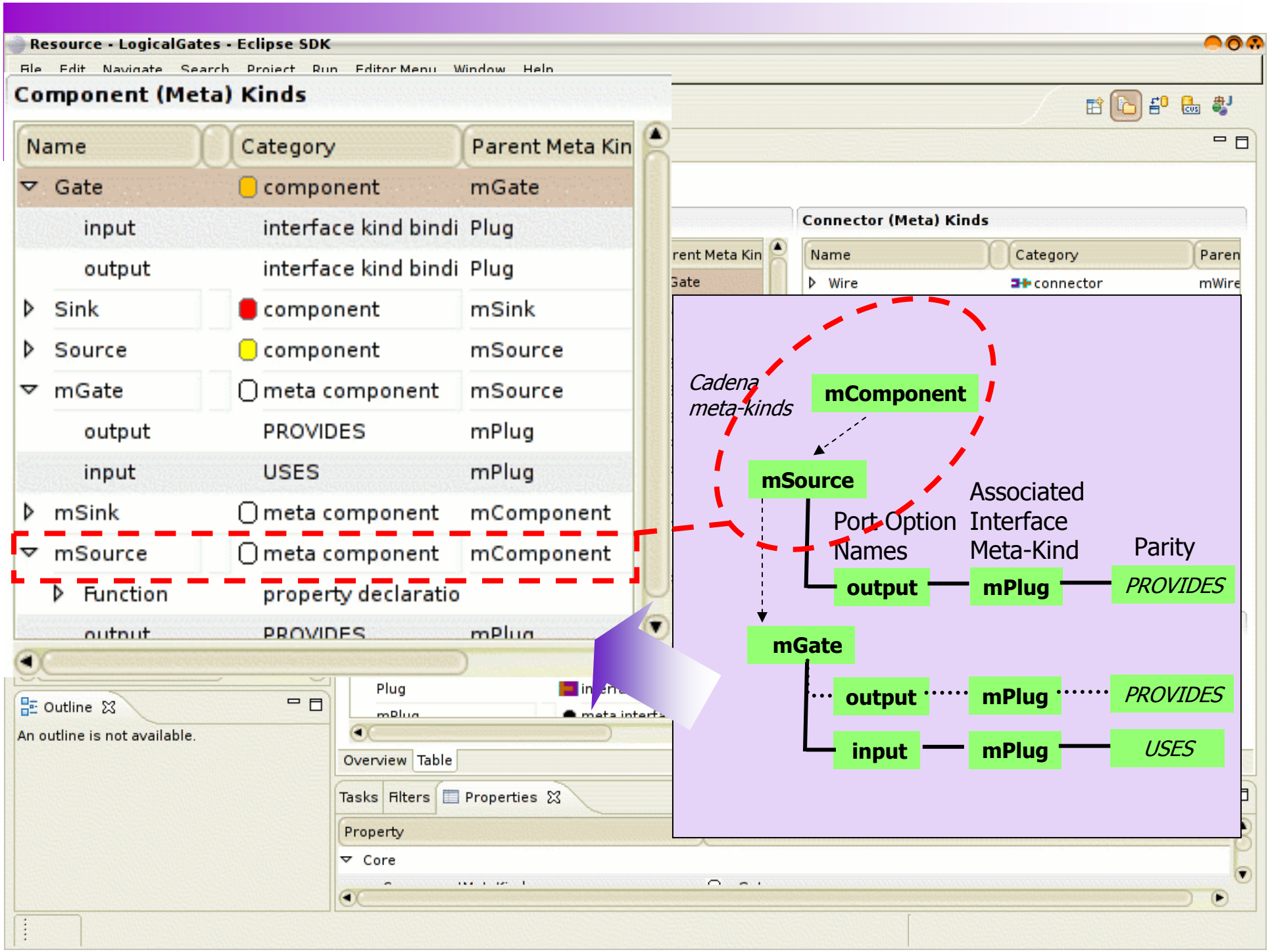


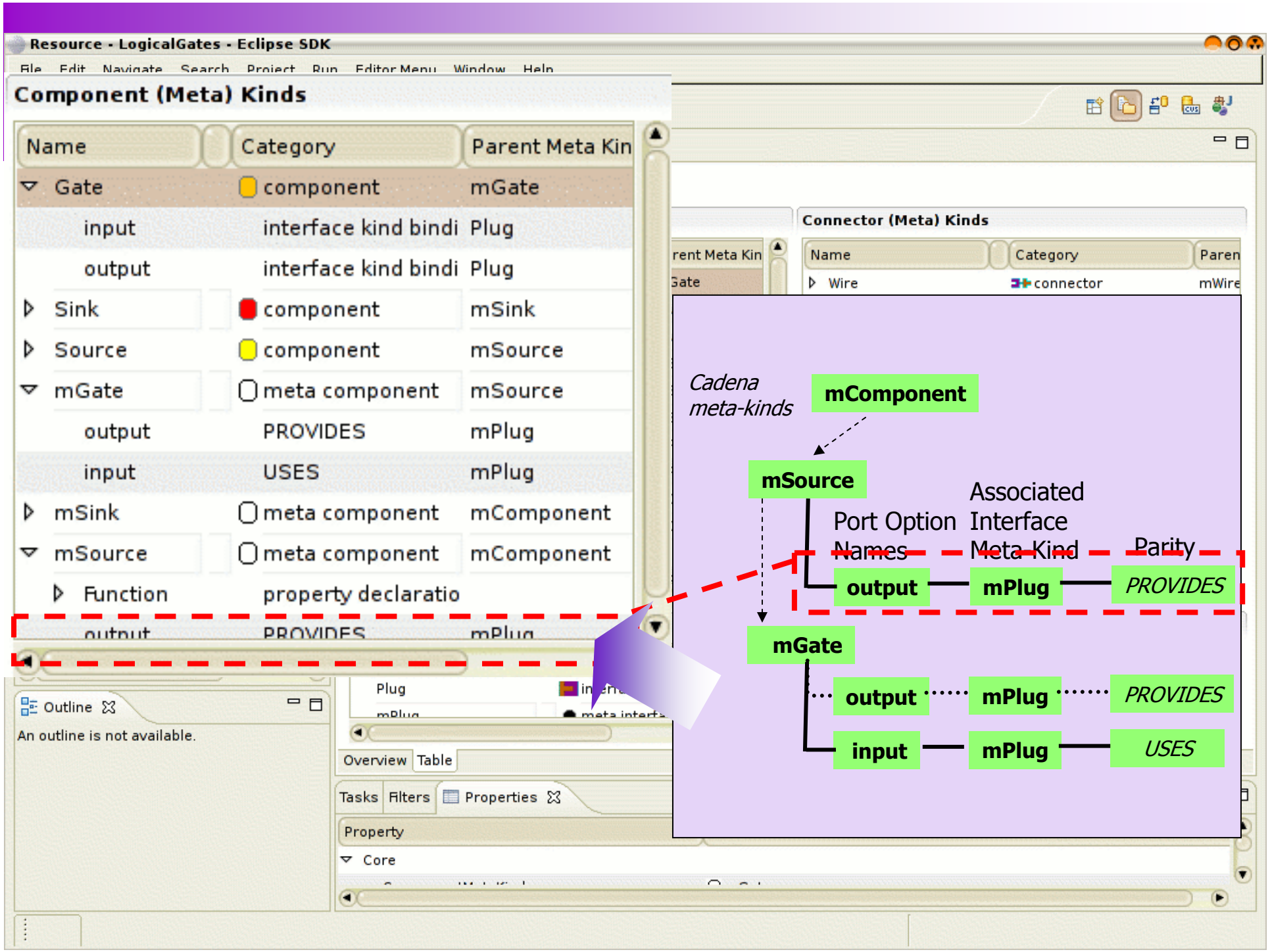
Logical Gates Modeled in Cadena

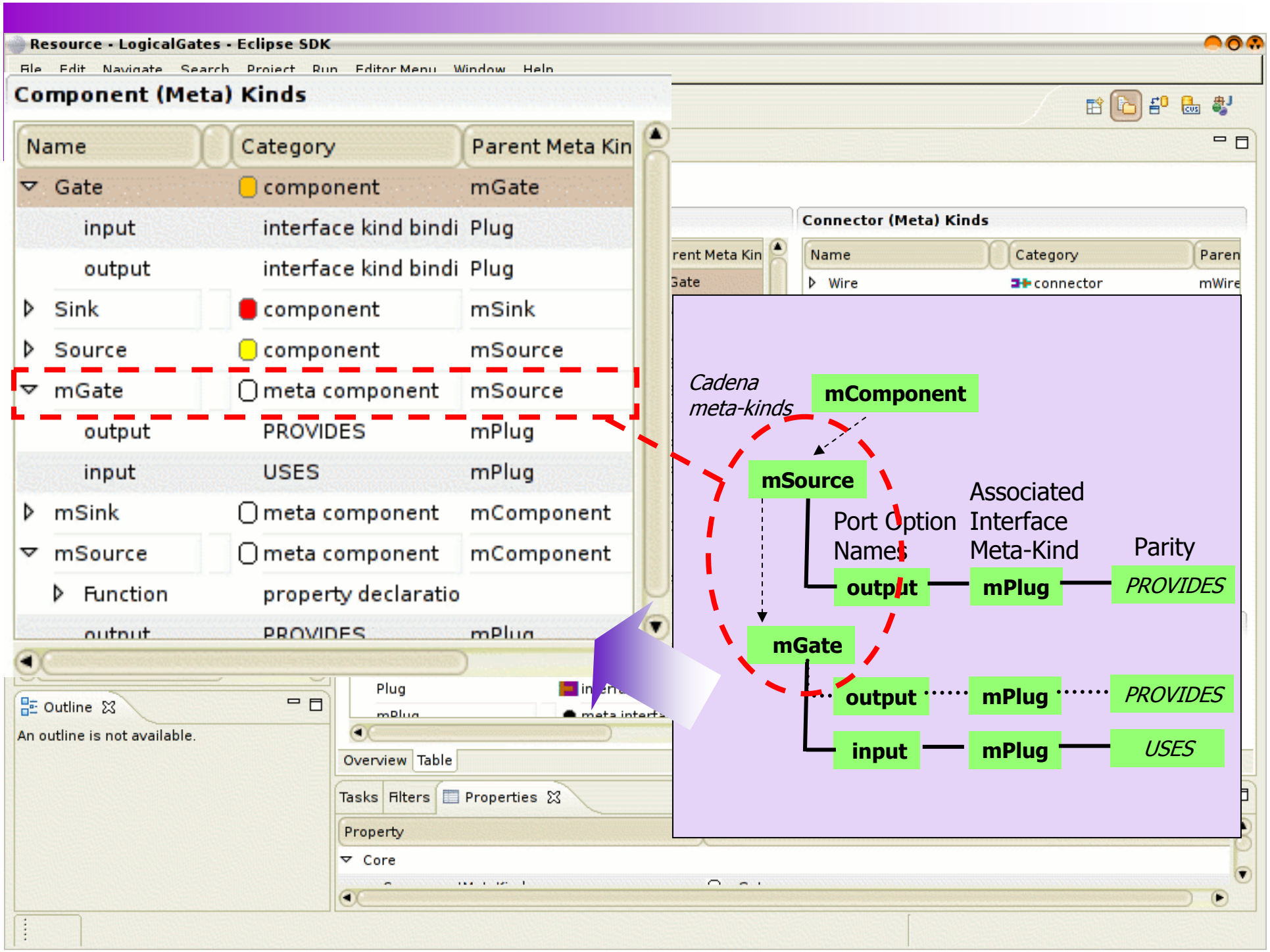


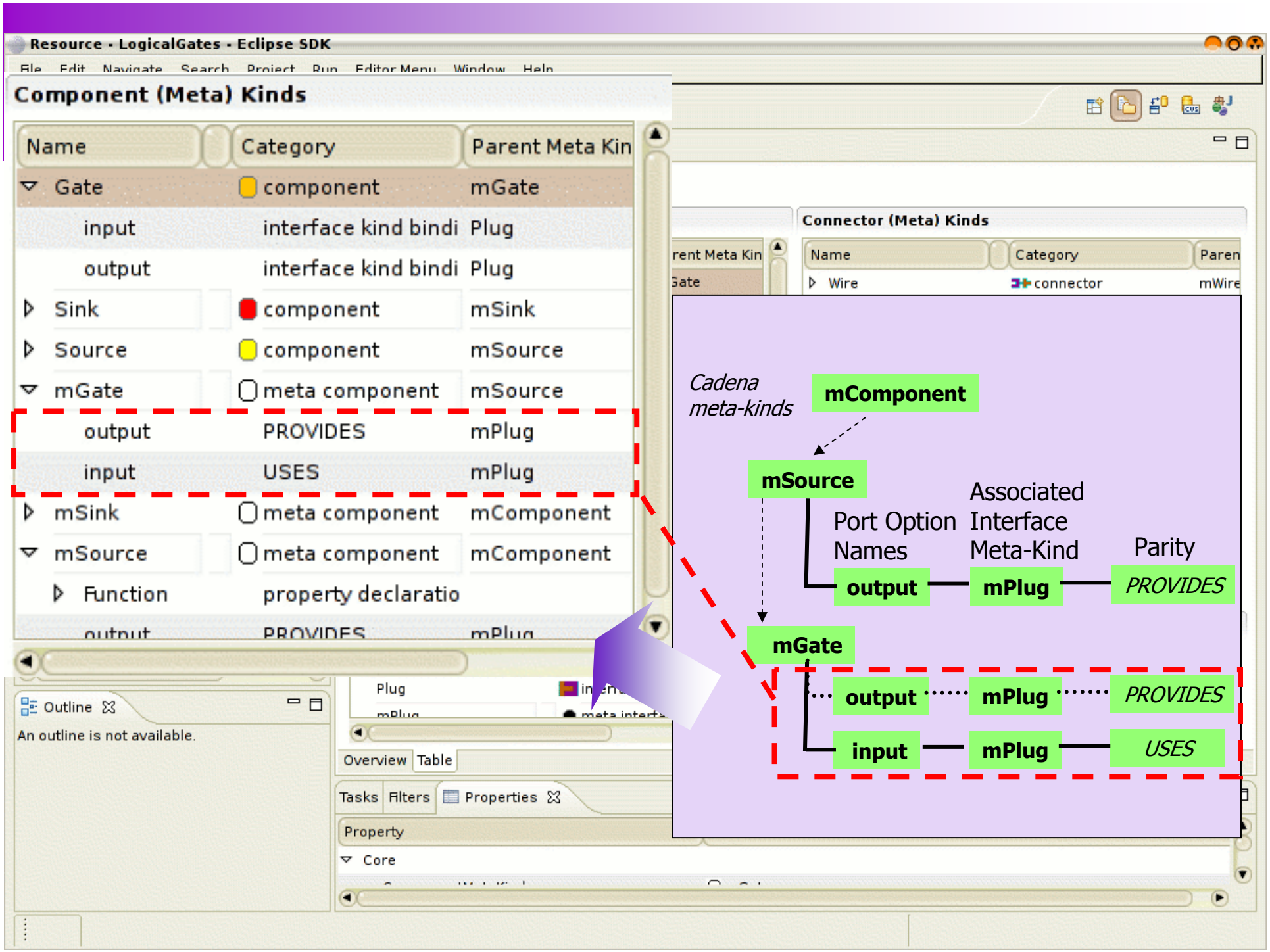
"input" and "output" are *user-defined keywords*, defined at the style tier for the model

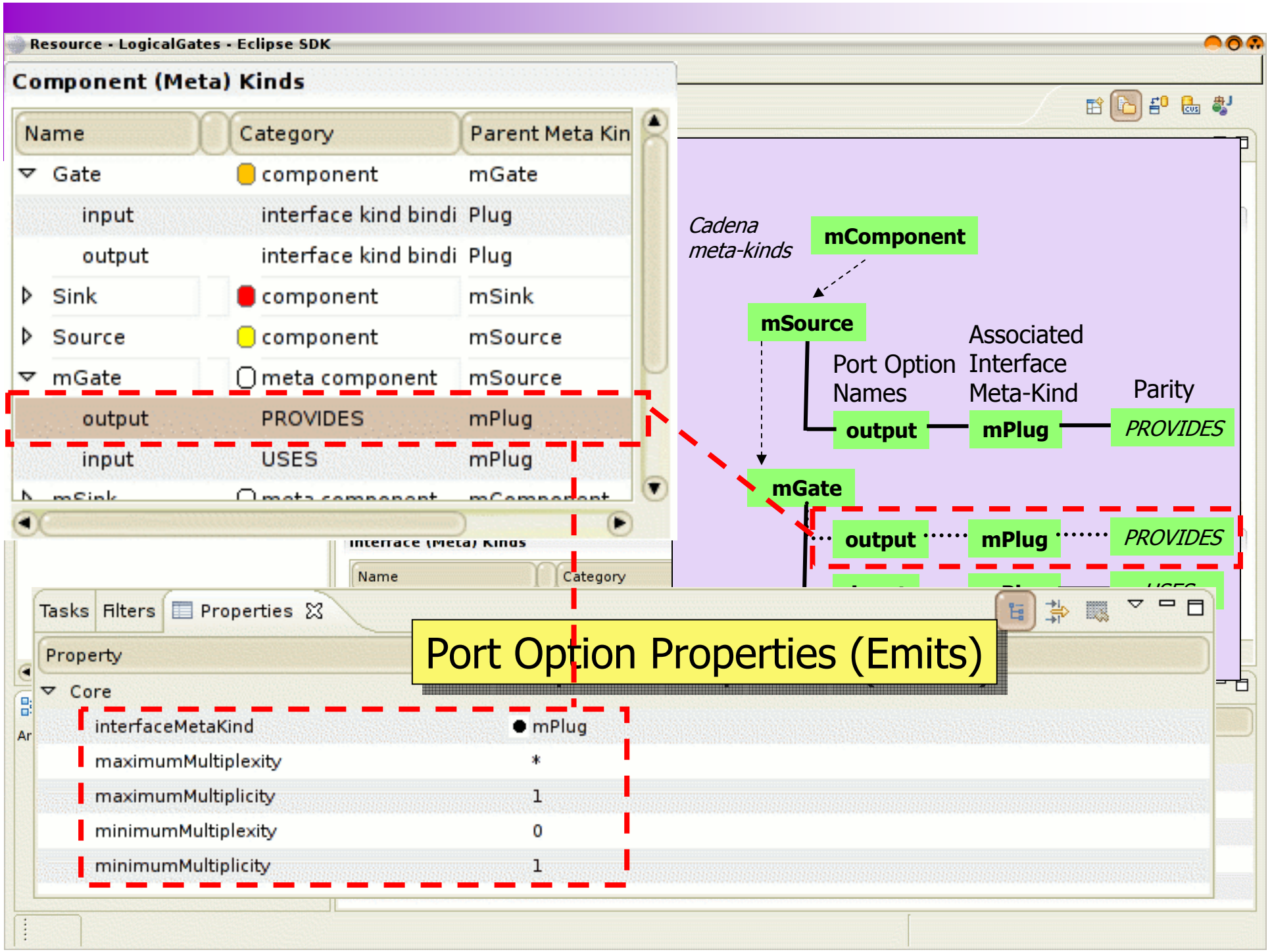
Shapes of a Logical Gate Component Interface


Resource - LogicalGates - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu Window Help
Component (Meta) Kinds
Name
Category
Parent Meta Kin
Gate
component
mGate
input
interface kind bindi
Plug
output
interface kind bindi
Plug
Sink
component
mSink
Source
component
mSource
mGate
meta component
mSource
output
PROVIDES
mPlug
input
USES
mPlug
mSink
meta component
mComponent
mSource
meta component
mComponent
Function
property declaratio
Connector (Meta) Kinds
Name
Category
Paren
Wire
connector
mWire
Cadena
meta-kinds
mComponent
mSource
Port Option
Names
Associated
Interface
Meta-Kind
Parity
output
mPlug
PROVIDES
mGate
output
mPlug
PROVIDES
input
mPlug
USES
Outline
An outline is not available.
Plug
Overview
Table
Tasks
Filters
Properties
Property
Core


Resource - LogicalGates - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu Window Help
Component (Meta) Kinds
Name
Category
Parent Meta Kin
Gate
component
mGate
input
interface kind bindi
Plug
output
interface kind bindi
Plug
Sink
component
mSink
Source
component
mSource
mGate
meta component
mSource
output
PROVIDES
mPlug
input
USES
mPlug
mSink
meta component
mComponent
mSource
meta component
mComponent
Function
property declaratio
Connector (Meta) Kinds
Name
Category
Paren
Wire
connector
mWire
Cadena meta-kinds
mComponent
mSource
Port Option Names
Associated Interface Meta-Kind
Parity
output
mPlug
PROVIDES
mGate
output
mPlug
PROVIDES
input
mPlug
USES
Outline
An outline is not available.
Plug
Overview
Table
Tasks
Filters
Properties
Property
Core

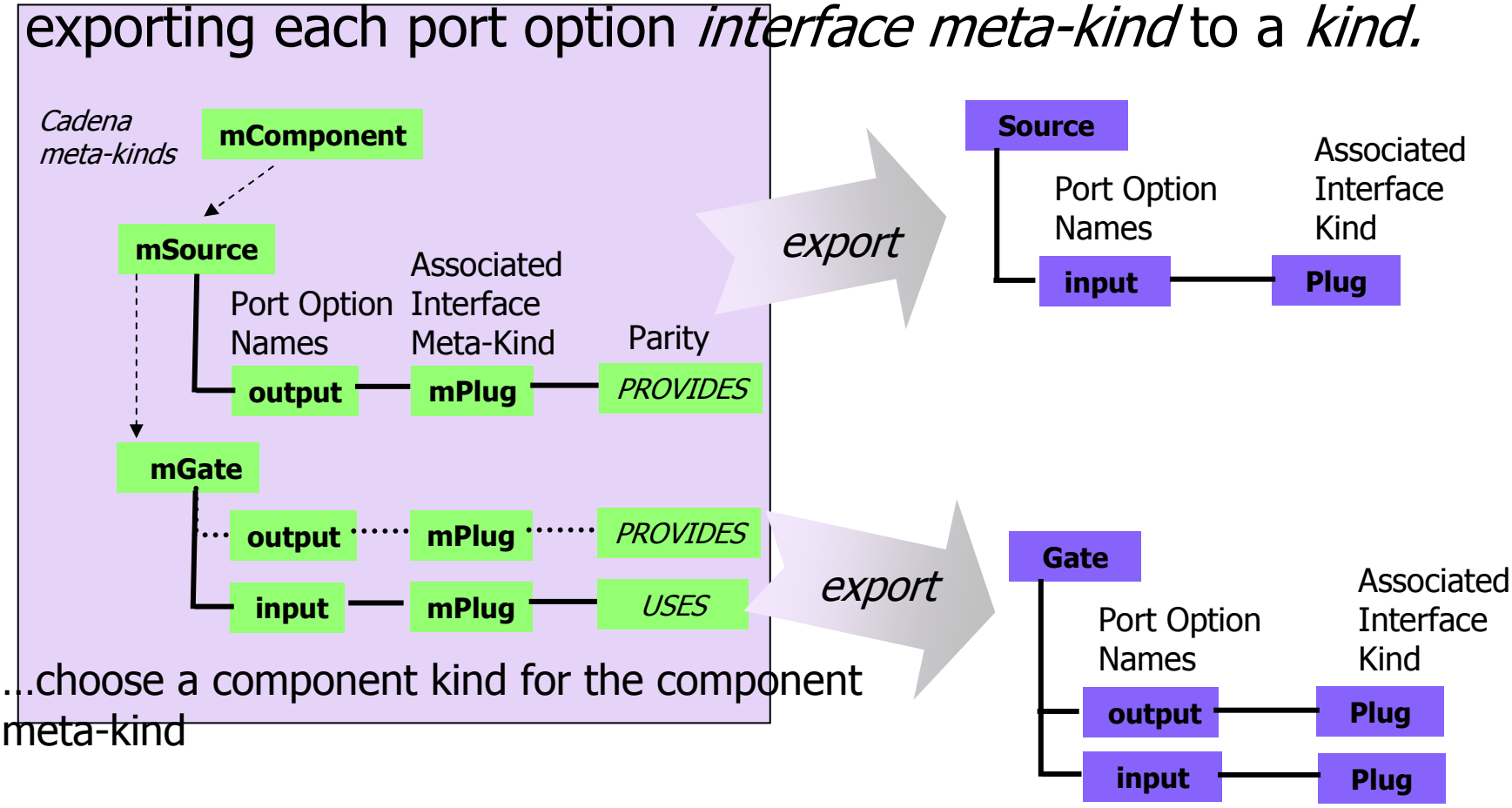

Resource - LogicalGates - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu Window Help
Component (Meta) Kinds
Name
Category
Parent Meta Kin
Gate
component
mGate
input
interface kind bindi
Plug
output
interface kind bindi
Plug
Sink
component
mSink
Source
component
mSource
mGate
meta component
mSource
output
PROVIDES
mPlug
input
USES
mPlug
mSink
meta component
mComponent
mSource
meta component
mComponent
Function
property declaratio
Connector (Meta) Kinds
Name
Category
Paren
Wire
connector
mWire
Cadena meta-kinds
mComponent
mSource
Port Option Names
Associated Interface Meta-Kind
Parity
output
mPlug
PROVIDES
mGate
output
mPlug
PROVIDES
input
mPlug
USES
Outline
An outline is not available.
Plug
Overview
Table
Tasks
Filters
Properties
Property
Core


Resource - LogicalGates - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu Window Help
Component (Meta) Kinds
Name
Category
Parent Meta Kin
Gate
component
mGate
input
interface kind bindi
Plug
output
interface kind bindi
Plug
Sink
component
mSink
Source
component
mSource
mGate
meta component
mSource
output
PROVIDES
mPlug
input
USES
mPlug
mSink
meta component
mComponent
mSource
meta component
mComponent
Function
property declaratio
Connector (Meta) Kinds
Name
Category
Paren
Wire
connector
mWire
Cadena meta-kinds
mComponent
mSource
Port Option Names
Associated Interface Meta-Kind
Parity
output
mPlug
PROVIDES
mGate
output
mPlug
PROVIDES
input
mPlug
USES
Outline
An outline is not available.
Plug
Overview
Table
Tasks
Filters
Properties
Property
Core


Resource - LogicalGates - Eclipse SDK
Component (Meta) Kinds
Name
Category
Parent Meta Kin
Gate
component
mGate
input
interface kind bindi
Plug
output
interface kind bindi
Plug
Sink
component
mSink
Source
component
mSource
mGate
meta component
mSource
output
PROVIDES
mPlug
input
USES
mPlug
Cadena meta-kinds
mComponent
mSource
Port Option Names
Associated Interface Meta-Kind
Parity
output
mPlug
PROVIDES
mGate
output
mPlug
PROVIDES
Tasks
Filters
Properties
Property
Core
interfaceMetaKind
mPlug
maximumMultiplexity
*
maximumMultiplicity
1
minimumMultiplexity
0
minimumMultiplicity
1
Port Option Properties (Emits)

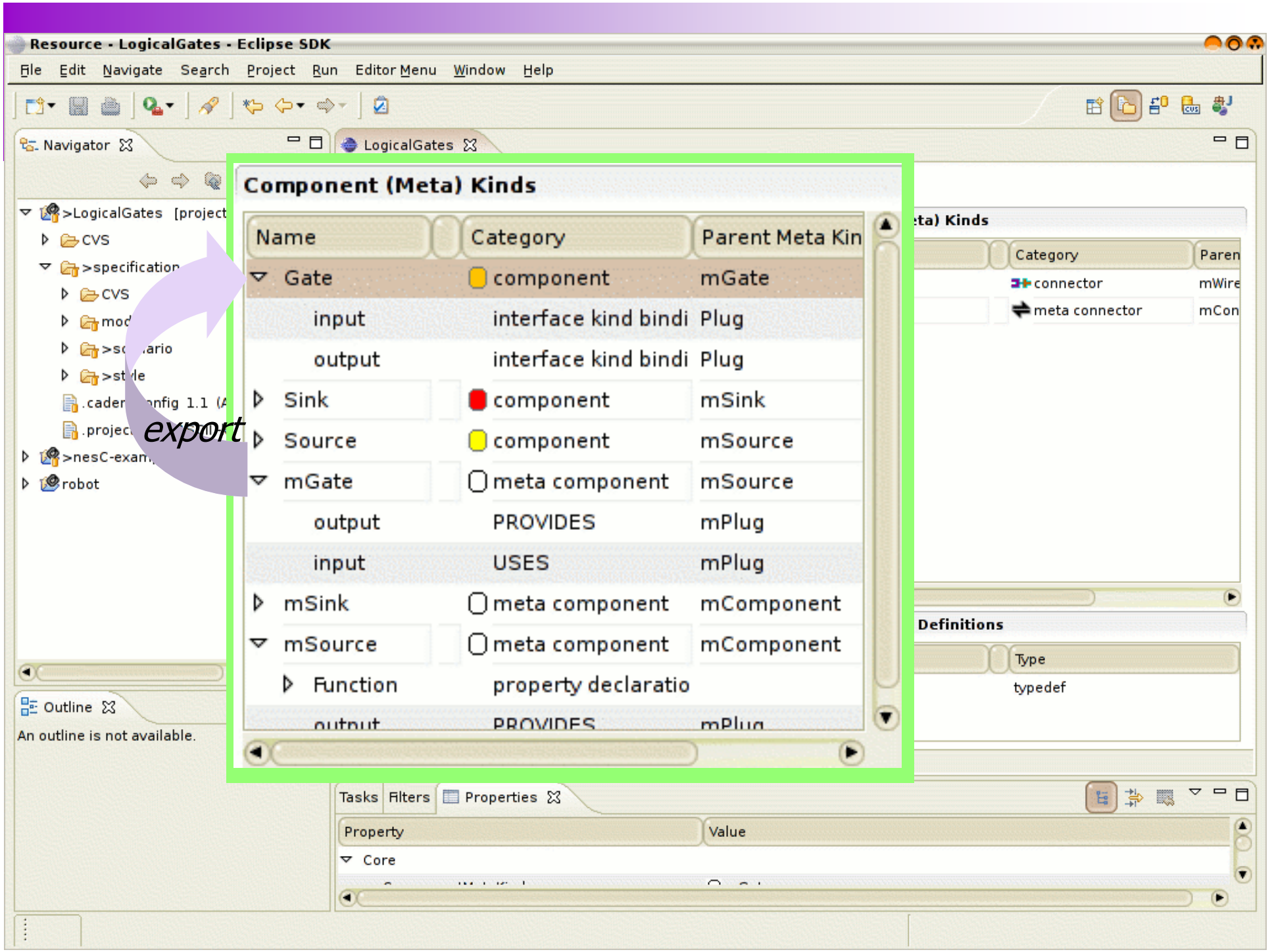
Modeling: Exporting Components

Exporting a *component meta-kind* to a *kind* involves exporting each port option *interface meta-kind* to a *kind.*



...choose a component kind for the component meta-kind

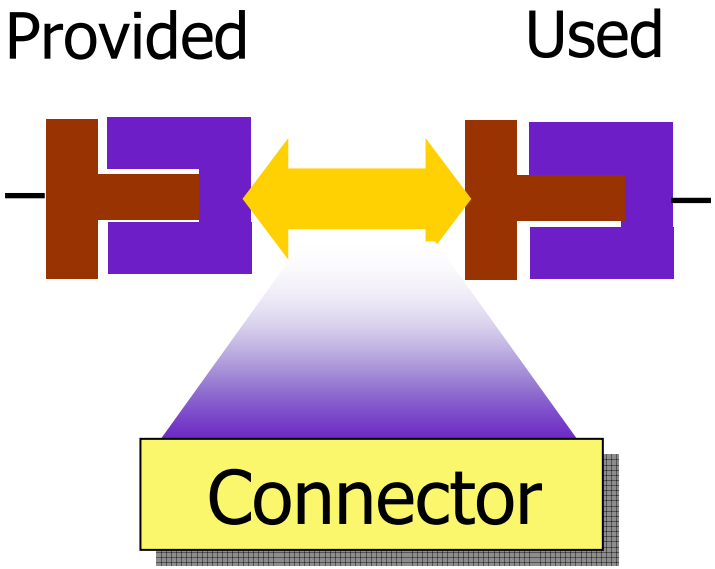
...choose an interface kind for each port option interface meta-kind


Resource - LogicalGates - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu Window Help
Navigator
LogicalGates
>LogicalGates [project
CVS
>specification
CVS
>nesC-exam
robot
export
Component (Meta) Kinds
Name
Category
Parent Meta Kin
Gate
component
mGate
input
interface kind bindi
Plug
output
interface kind bindi
Plug
Sink
component
mSink
Source
component
mSource
mGate
meta component
mSource
output
PROVIDES
mPlug
input
USES
mPlug
mSink
meta component
mComponent
mSource
meta component
mComponent
Function
property declaratio
connector
mWire
meta connector
mCon
Definitions
Type
typedef
Outline
An outline is not available.
Tasks
Filters
Properties
Property
Value
Core

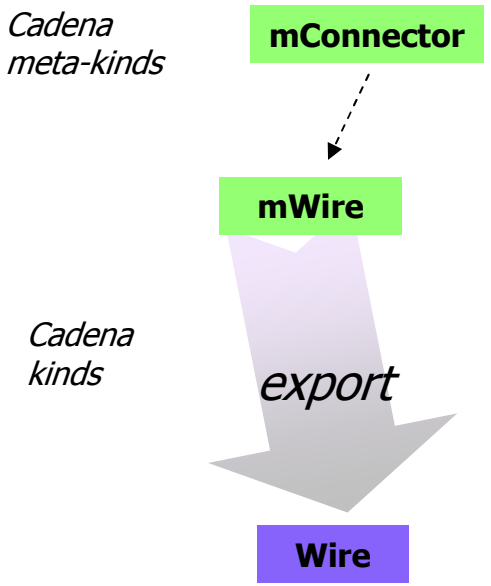
Modeling: Connectors

Distinct component ports cannot associate with the same interface instance. Between the ports of components in Cadena there has to be a connector.

Logical Gates

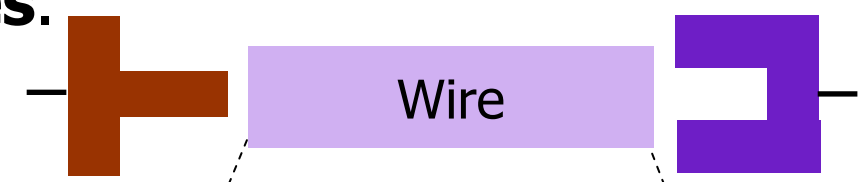


Logical Gates Cadena Model

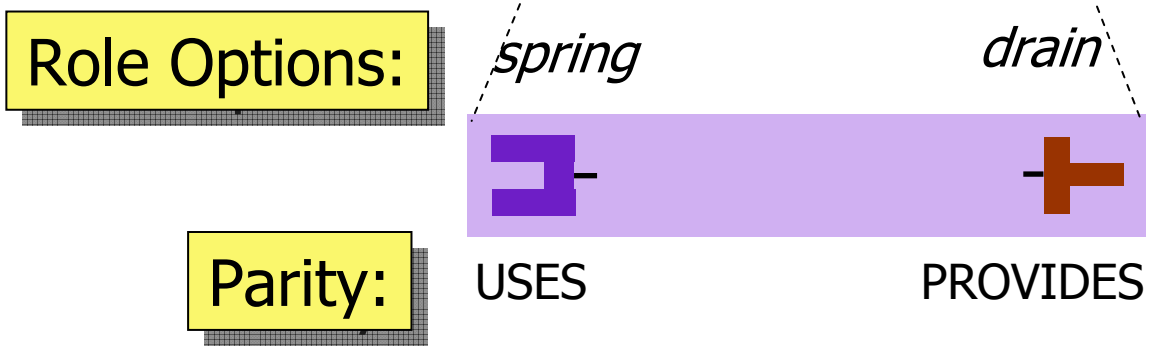


Modeling: Connector with Roles

The interfaces on each end of a connector are specific actors, fulfilling task within the connection. One might be the server, one the client. These end-points of connectors are called **roles**.



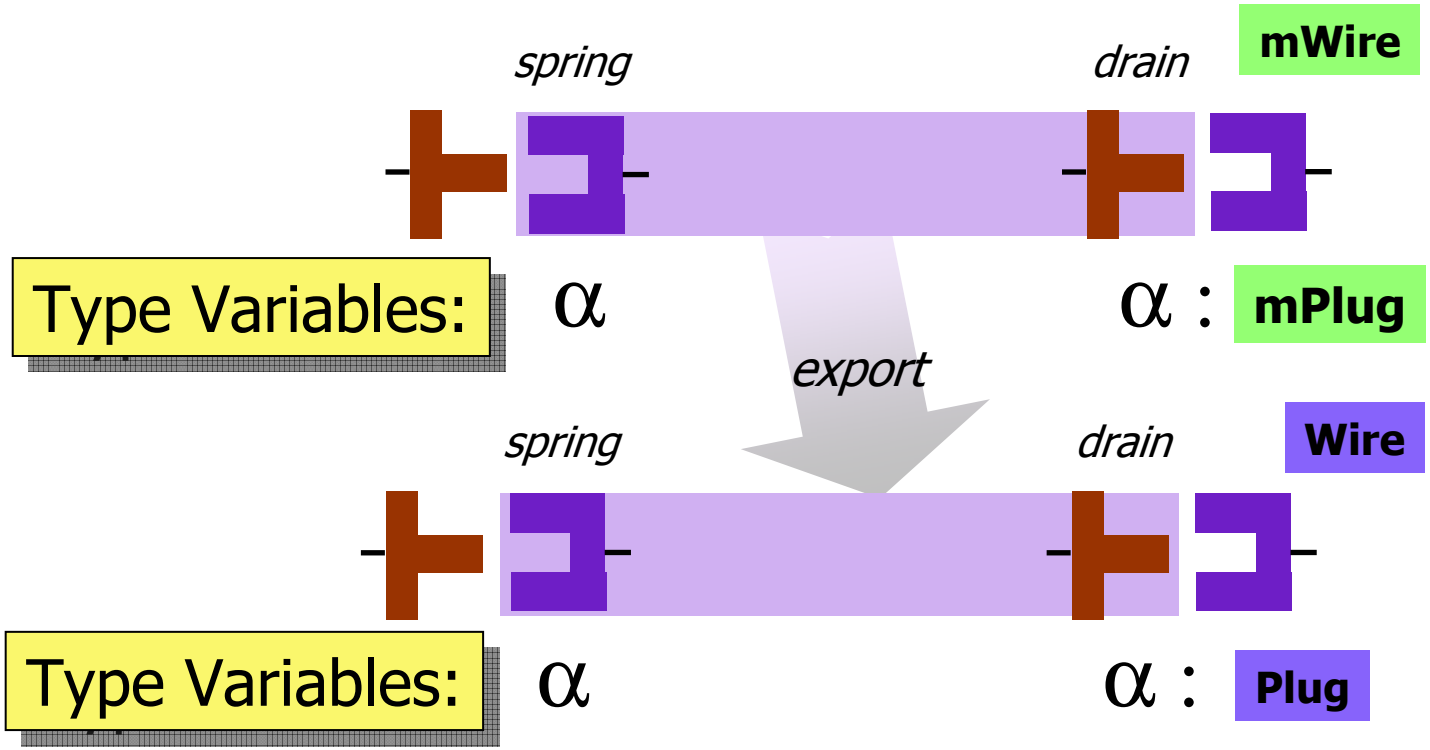
The wire-connector e.g. knows a spring and a drain of electricity



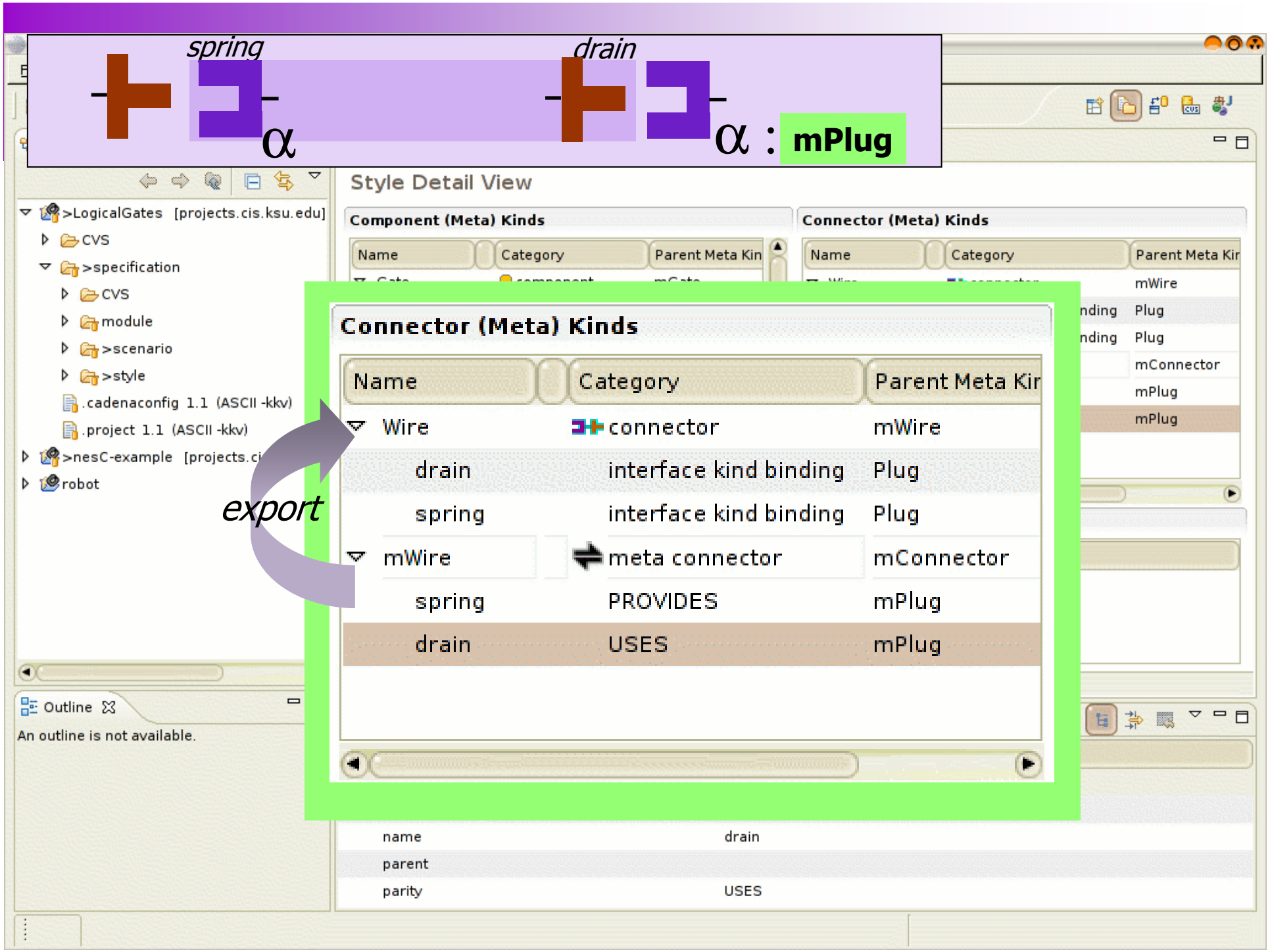
A Cadena connector has connection points called *roles* (analogous to component *ports*)

Modeling: Connection Constraints

Use *typing capability* to ensure that connections on either side of the connector are compatible.



For Wire, the typing tells us that the connector can bind on both sides to some Interface type T.


spring
drain
α
α : mPlug
Style Detail View
Component (Meta) Kinds
Connector (Meta) Kinds
Name
Category
Parent Meta Kin
>LogicalGates [projects.cis.ksu.edu]
CVS
>specification
CVS
module
>scenario
>style
.cadenaconfig 1.1 (ASCII -kkv)
.project 1.1 (ASCII -kkv)
>nesC-example [projects.ci
robot
export
Wire
connector
mWire
drain
interface kind binding
Plug
spring
interface kind binding
Plug
mWire
meta connector
mConnector
spring
PROVIDES
mPlug
drain
USES
mPlug
Outline
An outline is not available.
name
drain
parent
parity
USES

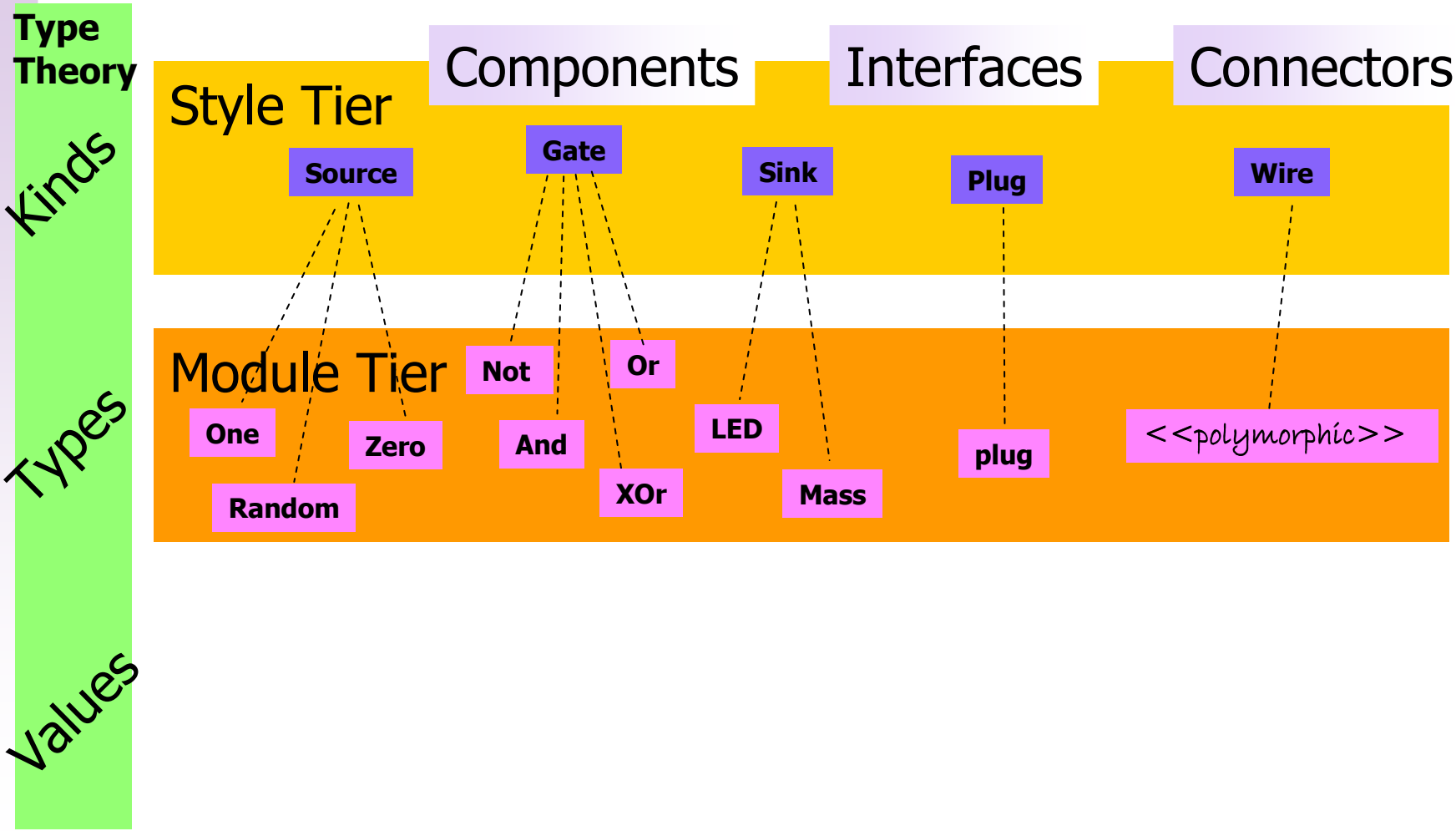
Development Environment

Product-Line Architect also defines plug-ins to aid process and guide workflow.

- Example Module Tier Plug-ins
  - Generation of gate implementations
- Example Scenario Tier Plug-ins
  - Circuit simulators
  - Layout planner

Types from Kinds

Declaring Types use vocabulary defined by Kinds

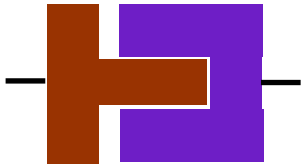


Modeling: Interface Types

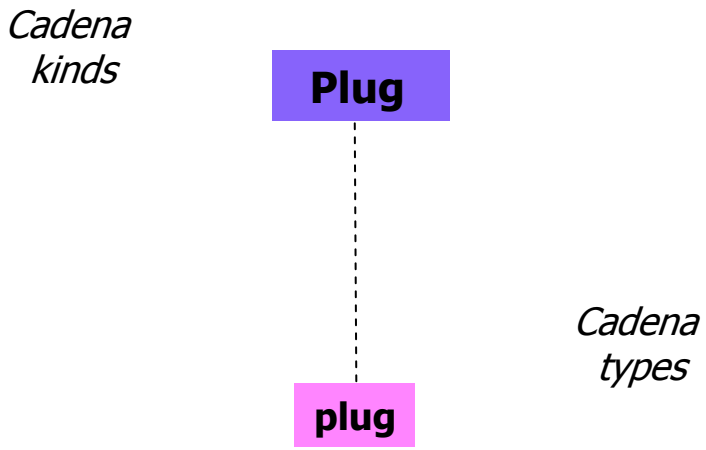
Within the interface kind Plug we define the single type plug.

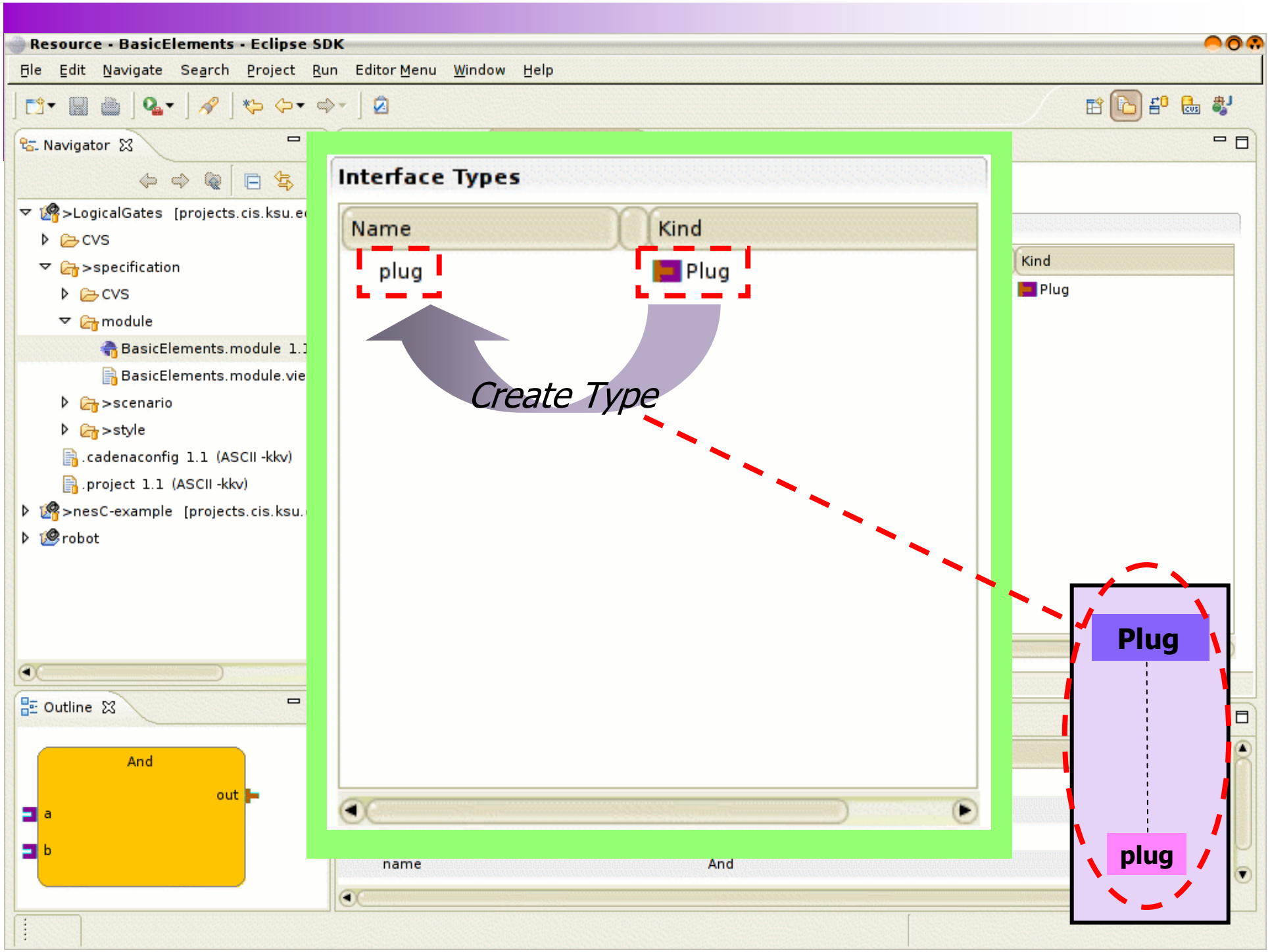
Logical Gates

The Plug interface



Logical Gates Cadena Model



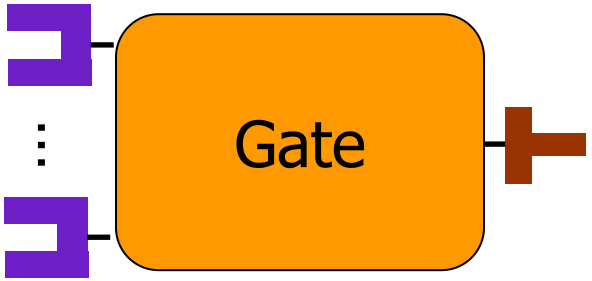

Resource - BasicElements - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu Window Help
Navigator
>LogicalGates [projects.cis.ksu.e
CVS
>specification
CVS
module
BasicElements.module 1.1
BasicElements.module.vie
>scenario
>style
.cadenaconfig 1.1 (ASCII -kkv)
.project 1.1 (ASCII -kkv)
>nesC-example [projects.cis.ksu.
robot
Outline
And
out
a
b
Interface Types
Name
Kind
plug
Plug
Create Type
Kind
Plug
name
And
Plug
plug

Modeling: Component Types

Within the interface kind Gate we define the types Not, And, Or, XOr, etc.

Logical Gates

The Gate Component



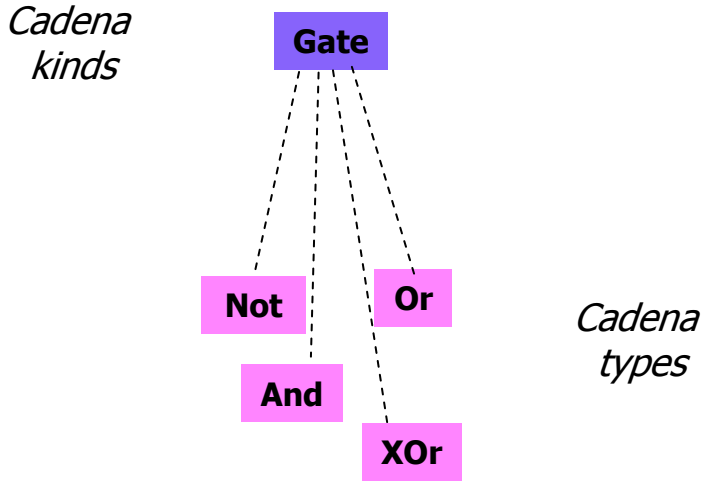
Not

Or

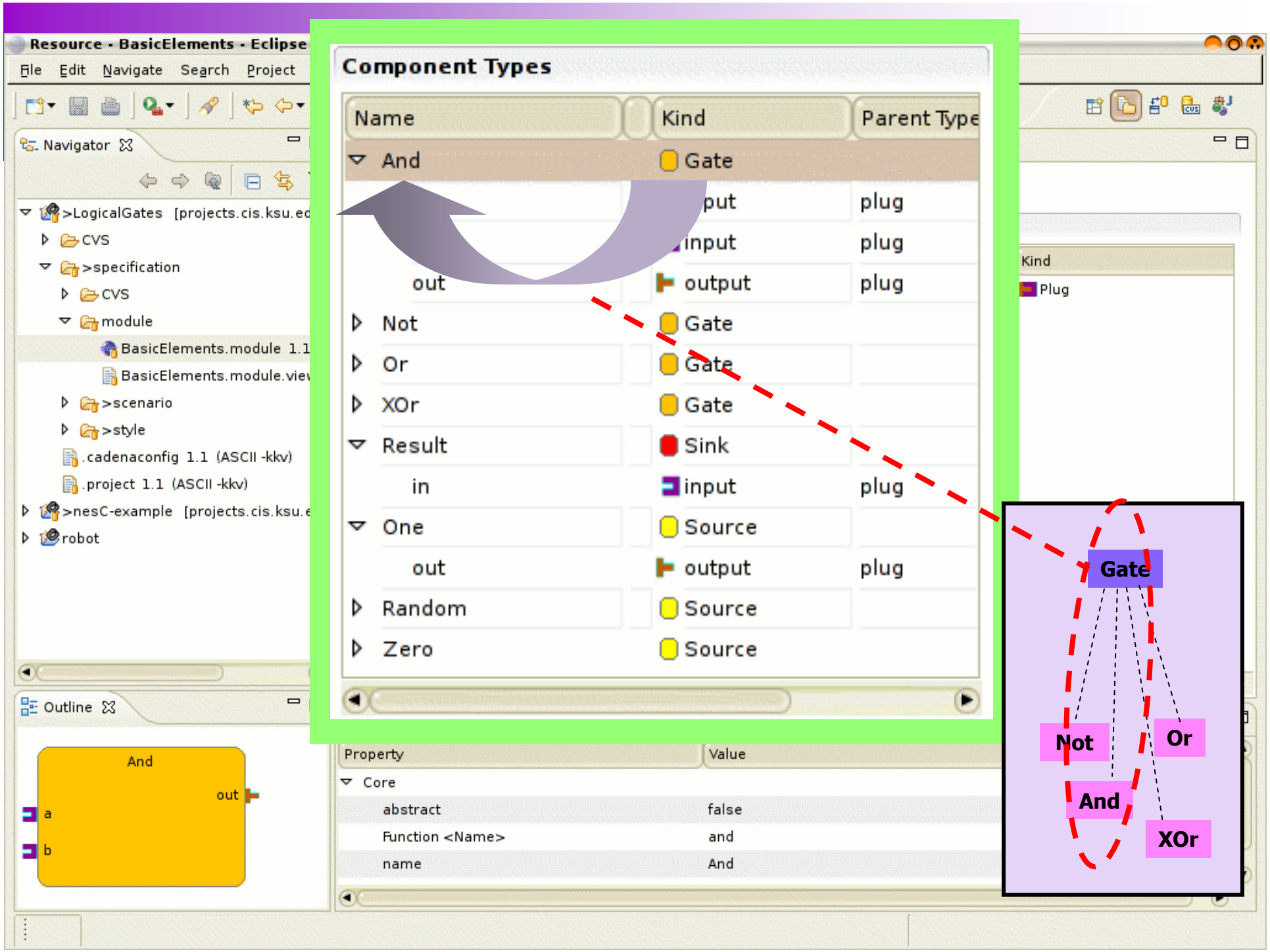
And

XOr

Logical Gates Cadena Model



Specify Types of "Gate"

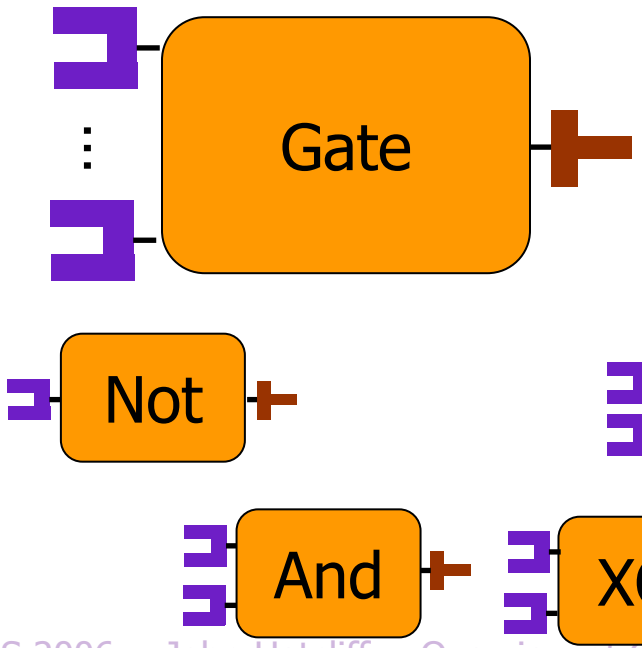

Resource - BasicElements - Eclipse
File Edit Navigate Search Project
Navigator
>LogicalGates [projects.cis.ksu.ed
CVS
>specification
CVS
module
BasicElements.module 1.1
BasicElements.module.view
>scenario
>style
.cadenaconfig 1.1 (ASCII -kkv)
.project 1.1 (ASCII -kkv)
>nesC-example [projects.cis.ksu.e
robot
Outline
And
out
a
b
Component Types
Name
Kind
Parent Type
And
Gate
put
plug
input
plug
out
output
plug
Not
Gate
Or
Gate
XOr
Gate
Result
Sink
in
input
plug
One
Source
out
output
plug
Random
Source
Zero
Source
Kind
Plug
Property
Value
Core
abstract
false
Function <Name>
and
name
And
Gate
Not
Or
And
XOr

Modeling: Component Ports

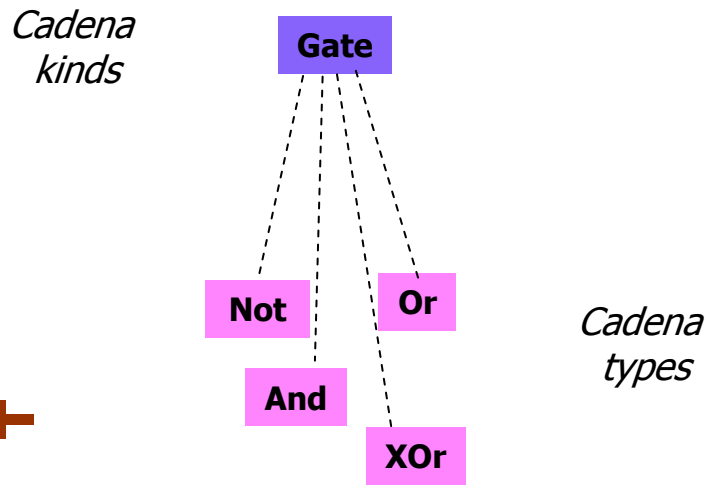
Within the interface kind Gate we define the types Not, And, Or, XOr, etc.

Logical Gates

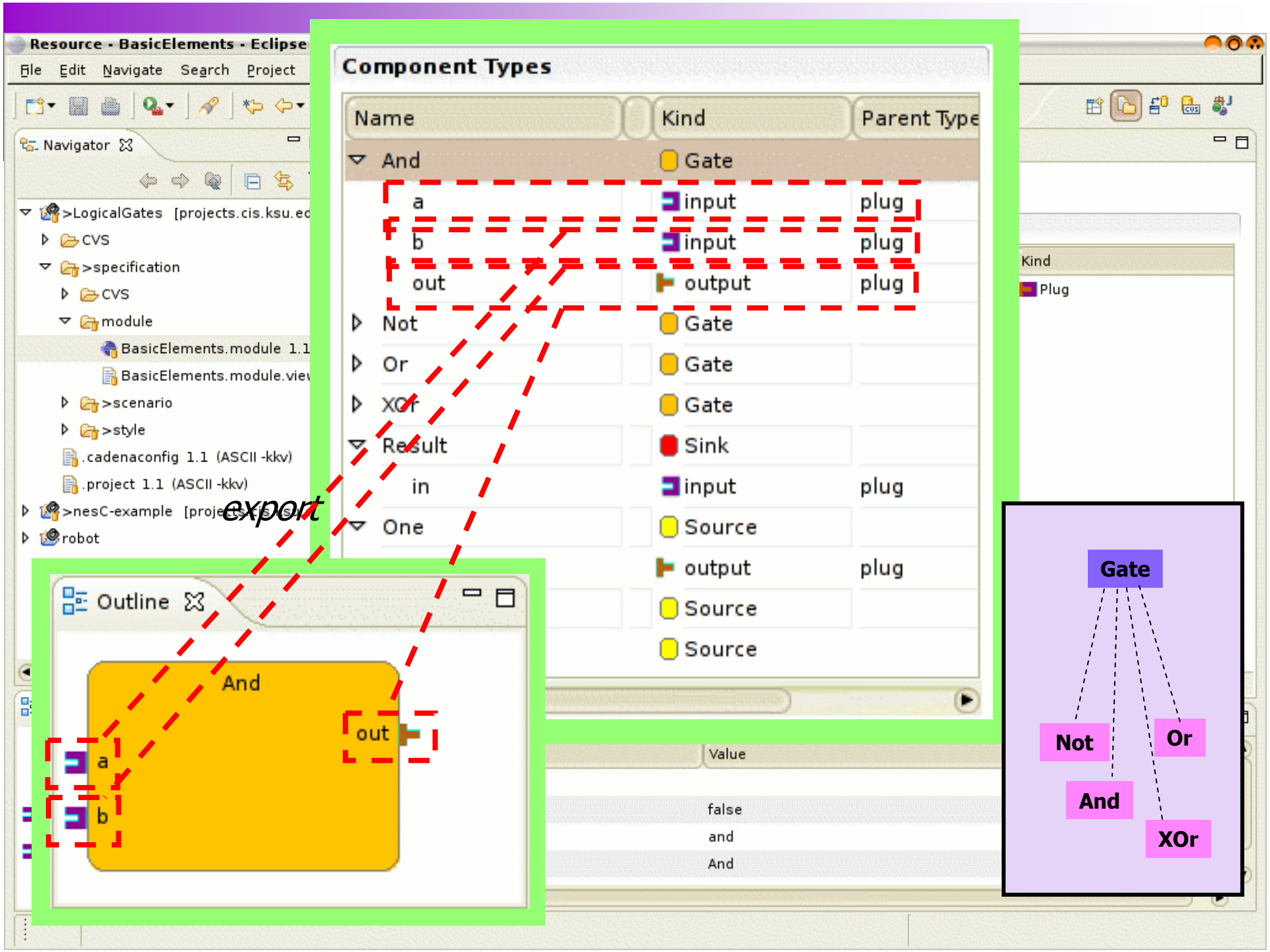
The Gate Component



Logical Gates Cadena Model

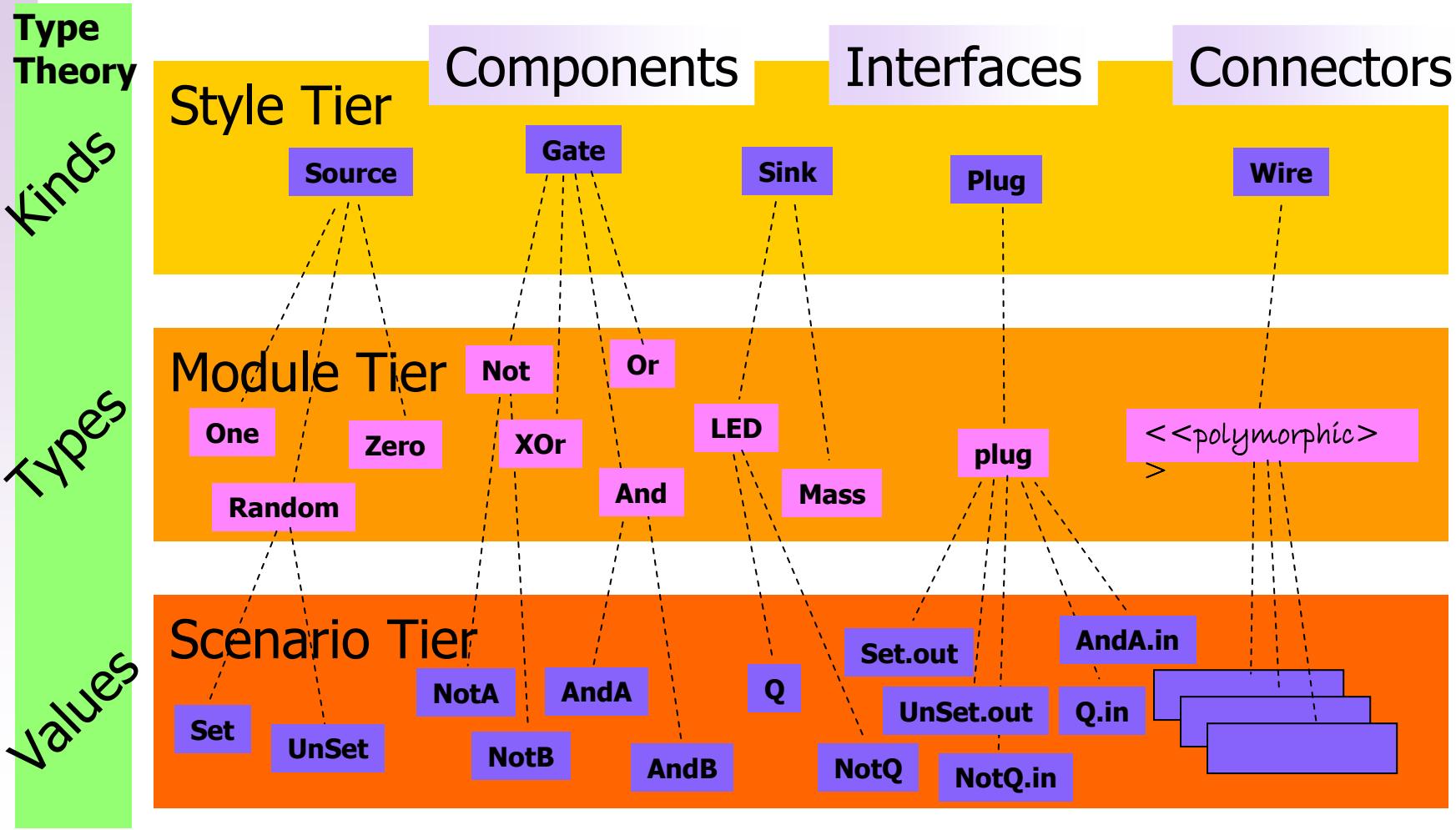


Populate the Port Options


Resource - BasicElements - Eclipse
File Edit Navigate Search Project
Navigator
>LogicalGates [projects.cis.ksu.ec
CVS
>specification
CVS
module
BasicElements.module 1.1
BasicElements.module.view
>scenario
>style
.cadenaconfig 1.1 (ASCII -kkv)
.project 1.1 (ASCII -kkv)
>nesC-example [projects.cis.ksu.
robot
Component Types
Name
Kind
Parent Type
And
Gate
a
input
plug
b
input
plug
out
output
plug
Not
Gate
Or
Gate
XOr
Gate
Result
Sink
in
input
plug
One
Source
output
plug
Source
Source
export
Outline
And
out
a
b
Value
false
and
And
Kind
Plug
Gate
Not
Or
And
XOr

Types from Kinds

Declaring Types use vocabulary defined by Kinds

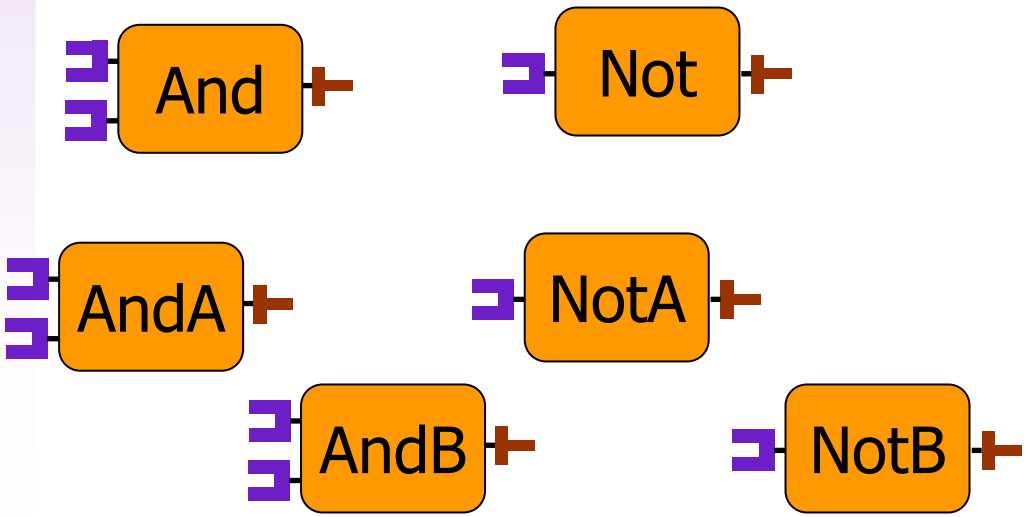


Modeling: Component Instances

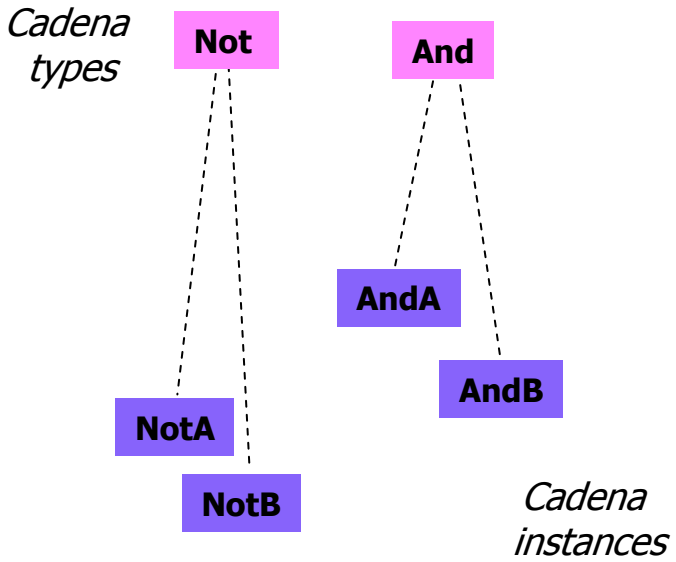
Types can be instantiated in specific assemblies or scenarios

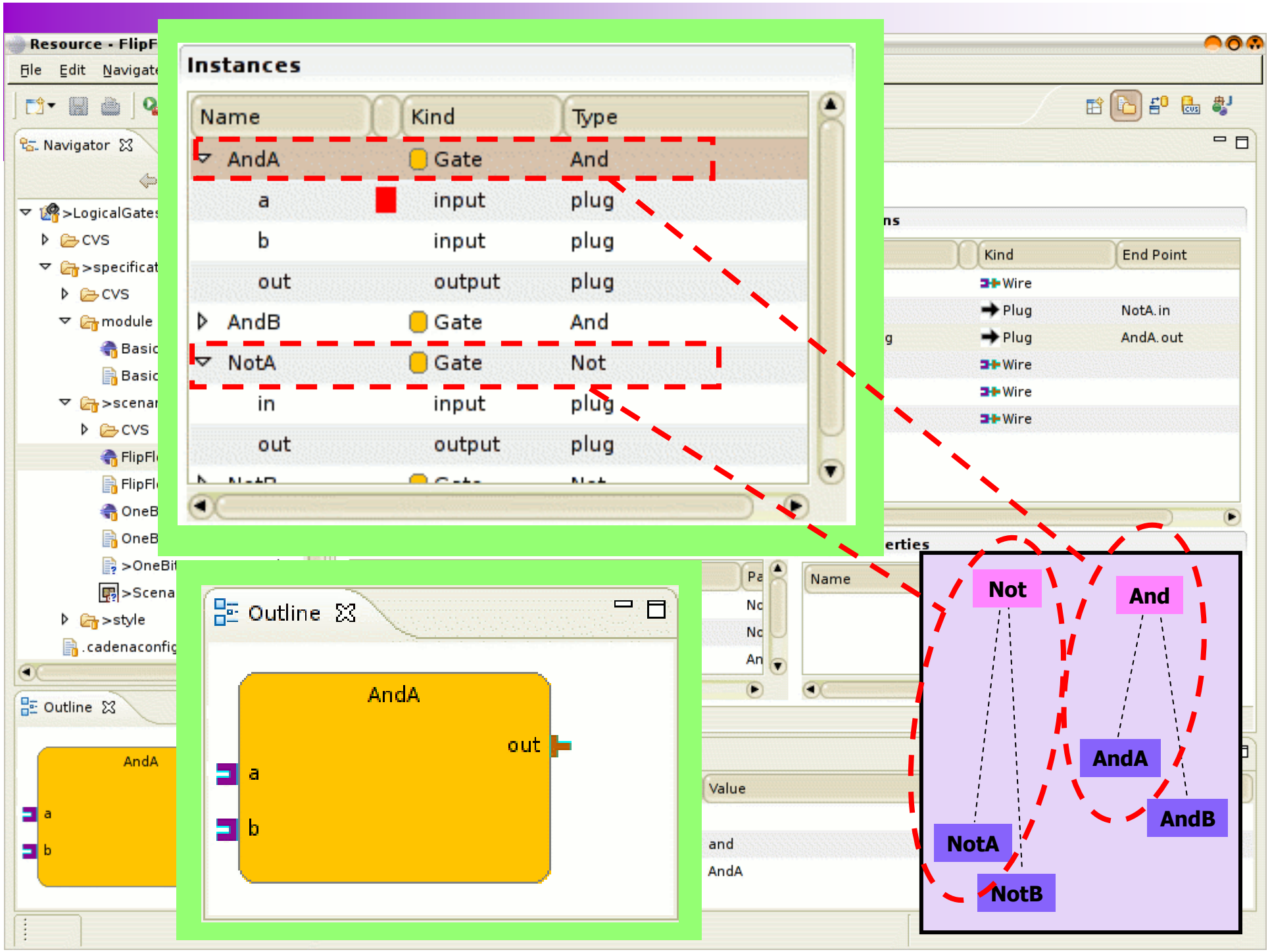
Logical Gates

The And and Not Component types are used twice



Logical Gates Cadena Model



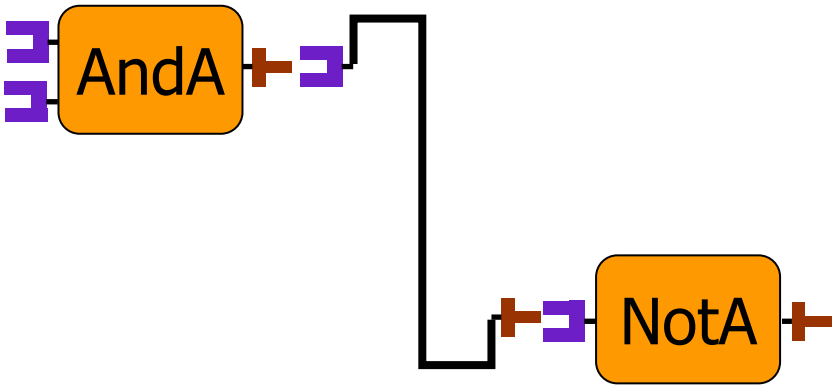

Instances
Name
Kind
Type
AndA
Gate
And
a
input
plug
b
input
plug
out
output
plug
AndB
Gate
And
NotA
Gate
Not
in
input
plug
out
output
plug
Outline
AndA
a
b
out
Not
And
AndA
AndB
NotA
NotB

Modeling: Connecting Components

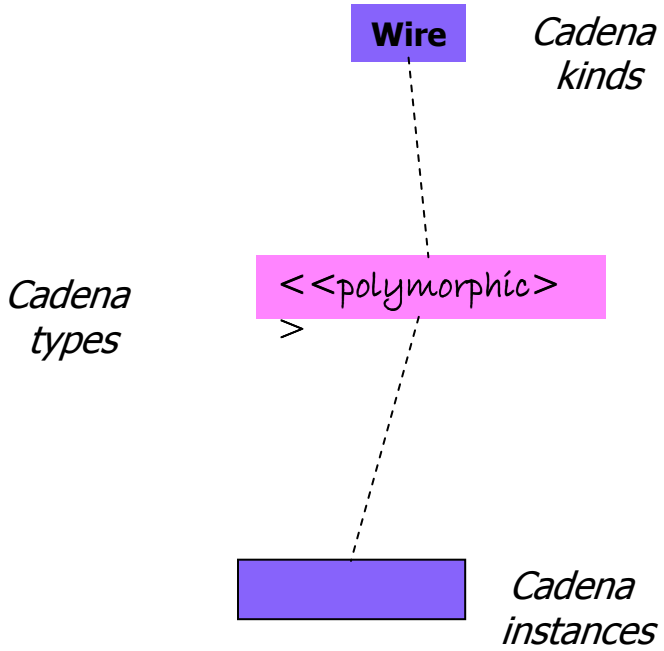
Connectors, being a model of the infrastructure, are instantiated out of the kind specification with polymorphic typing.

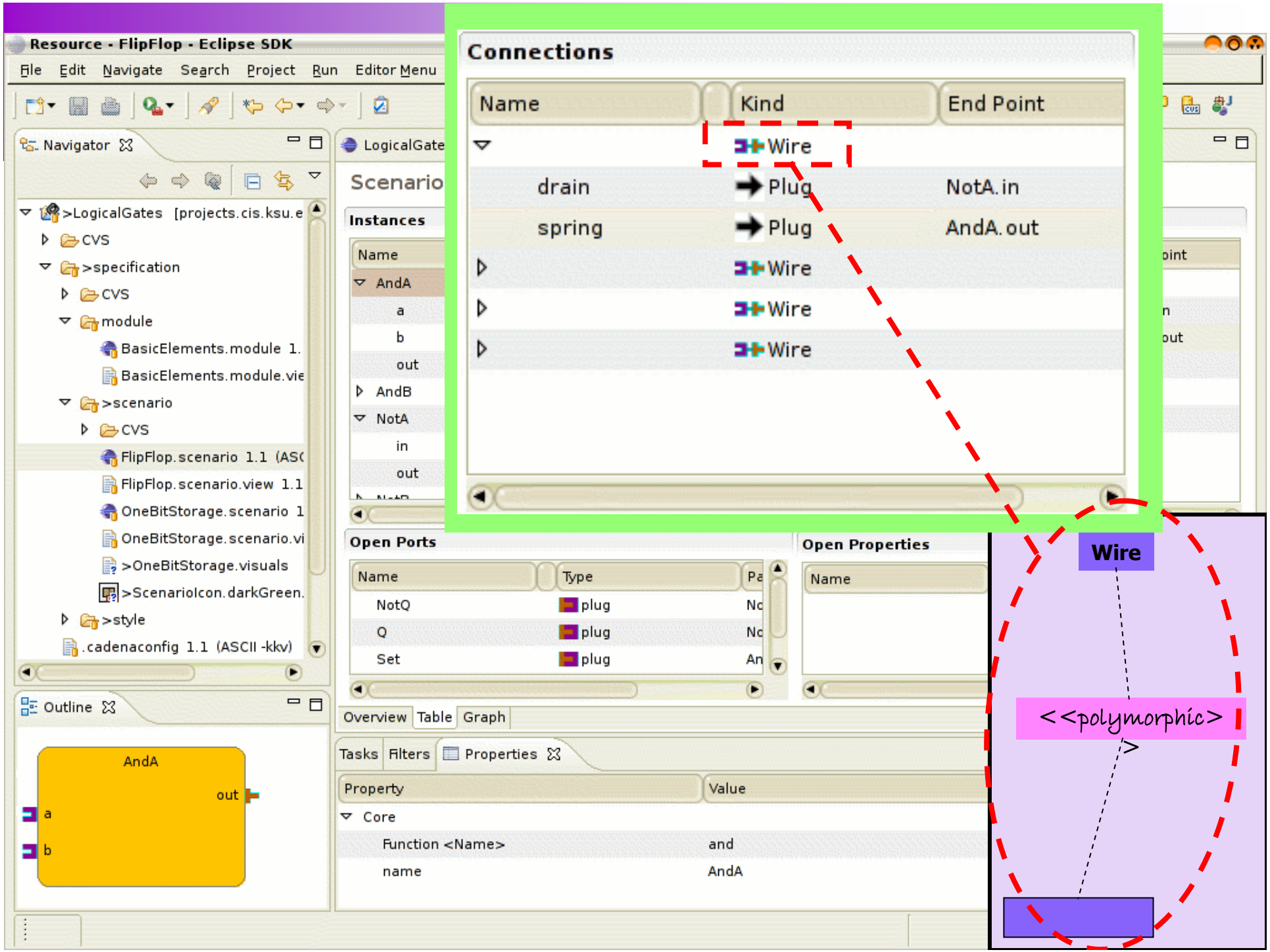
Logical Gates

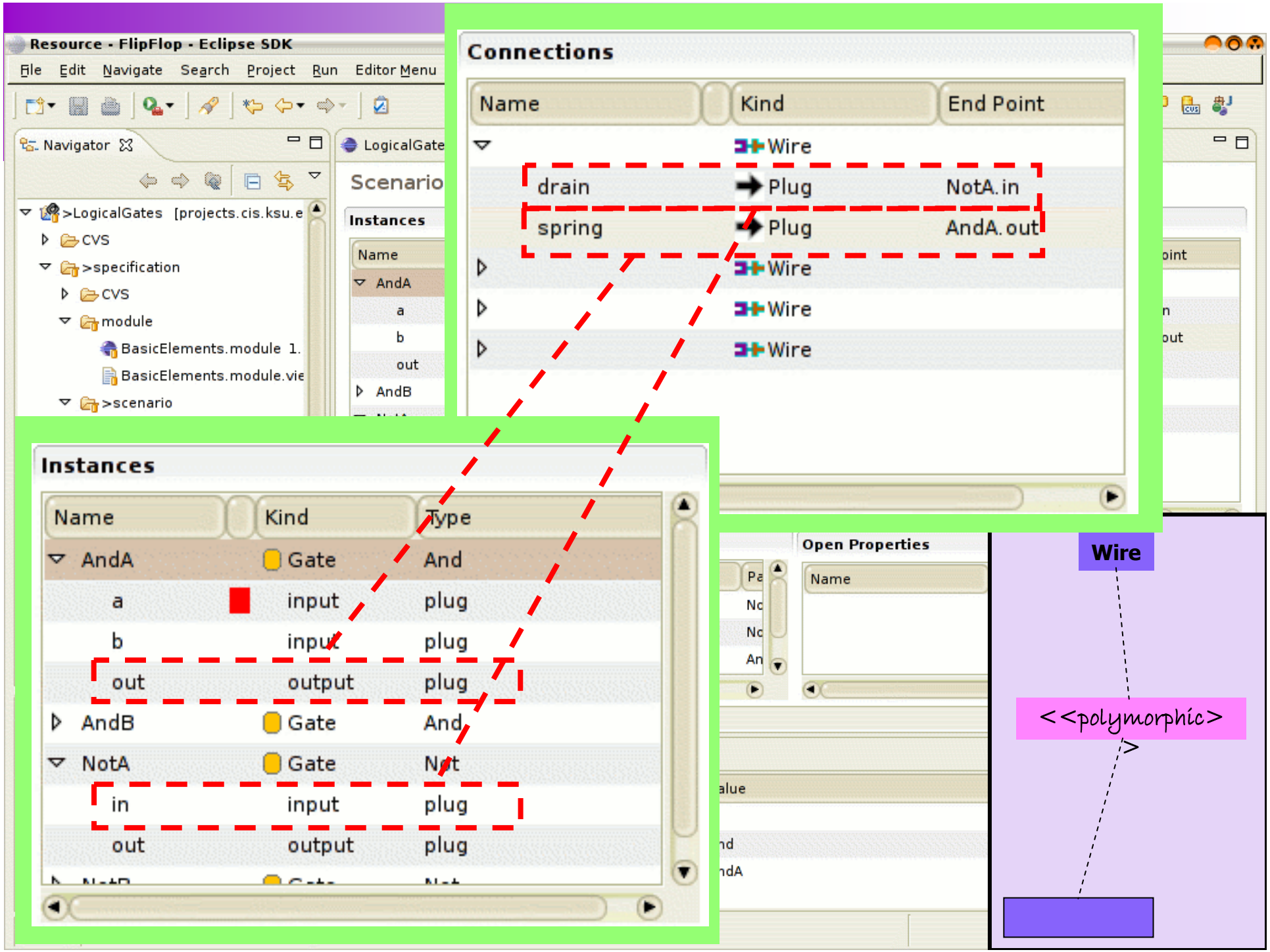
Connecting the instances "AndA" and "NotA"



Logical Gates Cadena Model




Resource - FlipFlop - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu
Navigator
>LogicalGates [projects.cis.ksu.e
CVS
>specification
CVS
module
BasicElements.module 1.
BasicElements.module.vie
>scenario
CVS
FlipFlop.scenario 1.1 (AS(
FlipFlop.scenario.view 1.1
OneBitStorage.scenario 1
OneBitStorage.scenario.vi
>OneBitStorage.visuals
>ScenarioIcon.darkGreen.
>style
.cadenaconfig 1.1 (ASCII -kkv)
Outline
AndA
out
a
b
LogicalGate
Scenario
Instances
Name
AndA
a
b
out
AndB
NotA
in
out
Open Ports
Name
Type
NotQ
plug
Q
plug
Set
plug
Open Properties
Name
Overview Table Graph
Tasks Filters Properties
Property
Value
Core
Function <Name>
and
name
AndA
Connections
Name
Kind
End Point
Wire
drain
Plug
NotA.in
spring
Plug
AndA.out
Wire
Wire
Wire
Wire
<<polymorphic>>
>

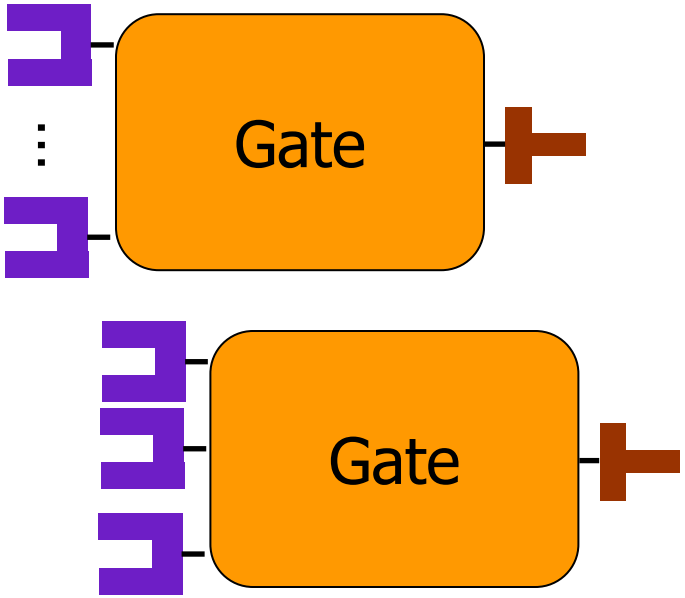

Resource - FlipFlop - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu
Navigator
>LogicalGates [projects.cis.ksu.e
CVS
>specification
CVS
module
BasicElements.module 1.
BasicElements.module.vie
>scenario
LogicalGate
Scenario
Instances
Name
AndA
a
b
out
AndB
Connections
Name
Kind
End Point
Wire
drain
Plug
NotA.in
spring
Plug
AndA.out
Wire
Wire
Wire
Instances
Name
Kind
Type
AndA
Gate
And
a
input
plug
b
input
plug
out
output
plug
AndB
Gate
And
NotA
Gate
Not
in
input
plug
out
output
plug
Open Properties
Name
Wire
<<polymorphic>>

Modeling: Component Types

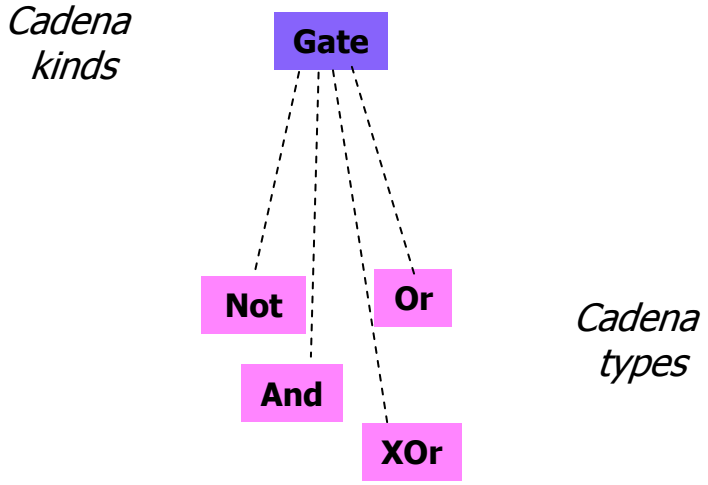
Within the interface kind Gate we define the types Not, And, Or, XOr, etc.

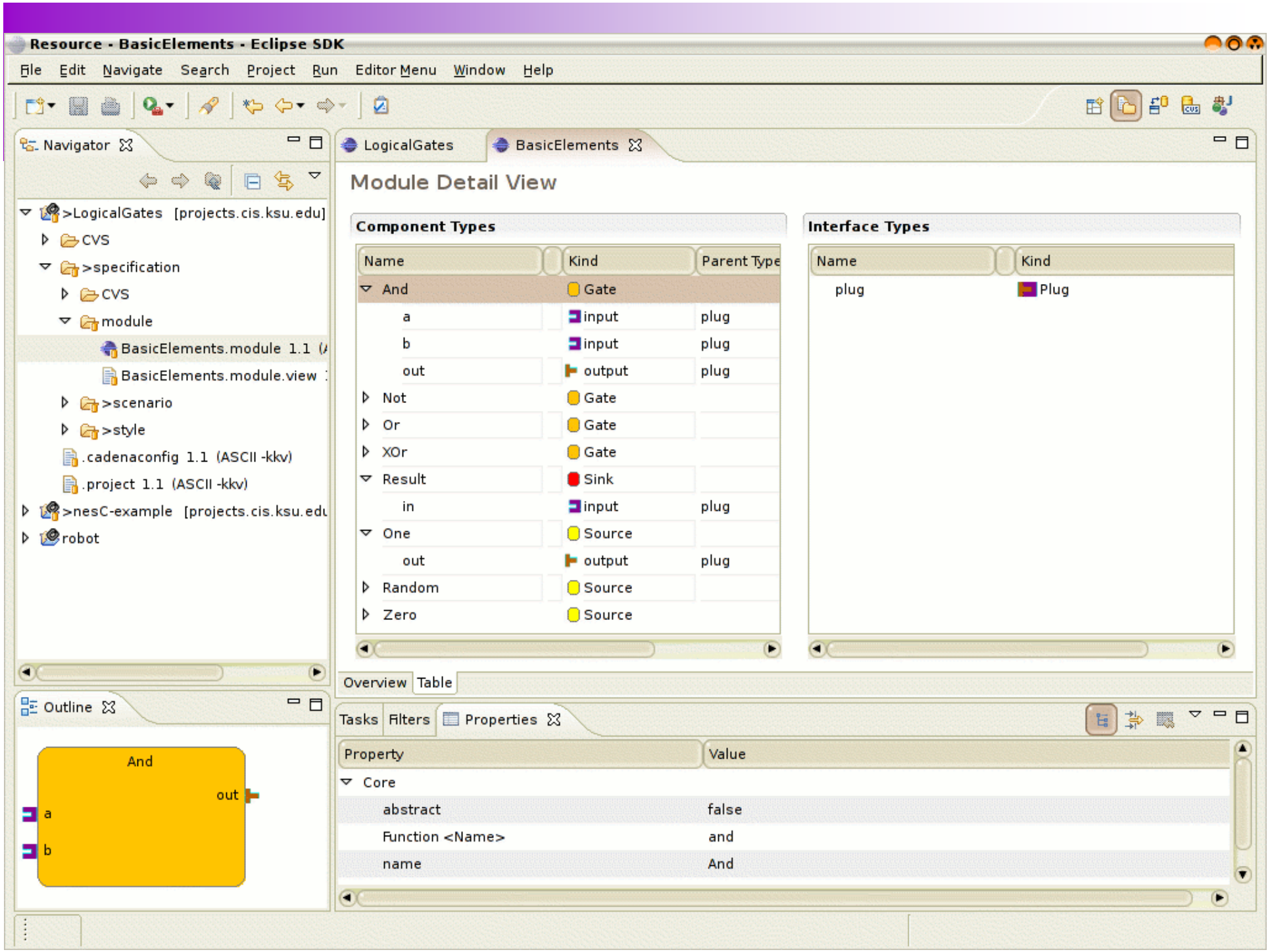
Logical Gates

The Gate Component



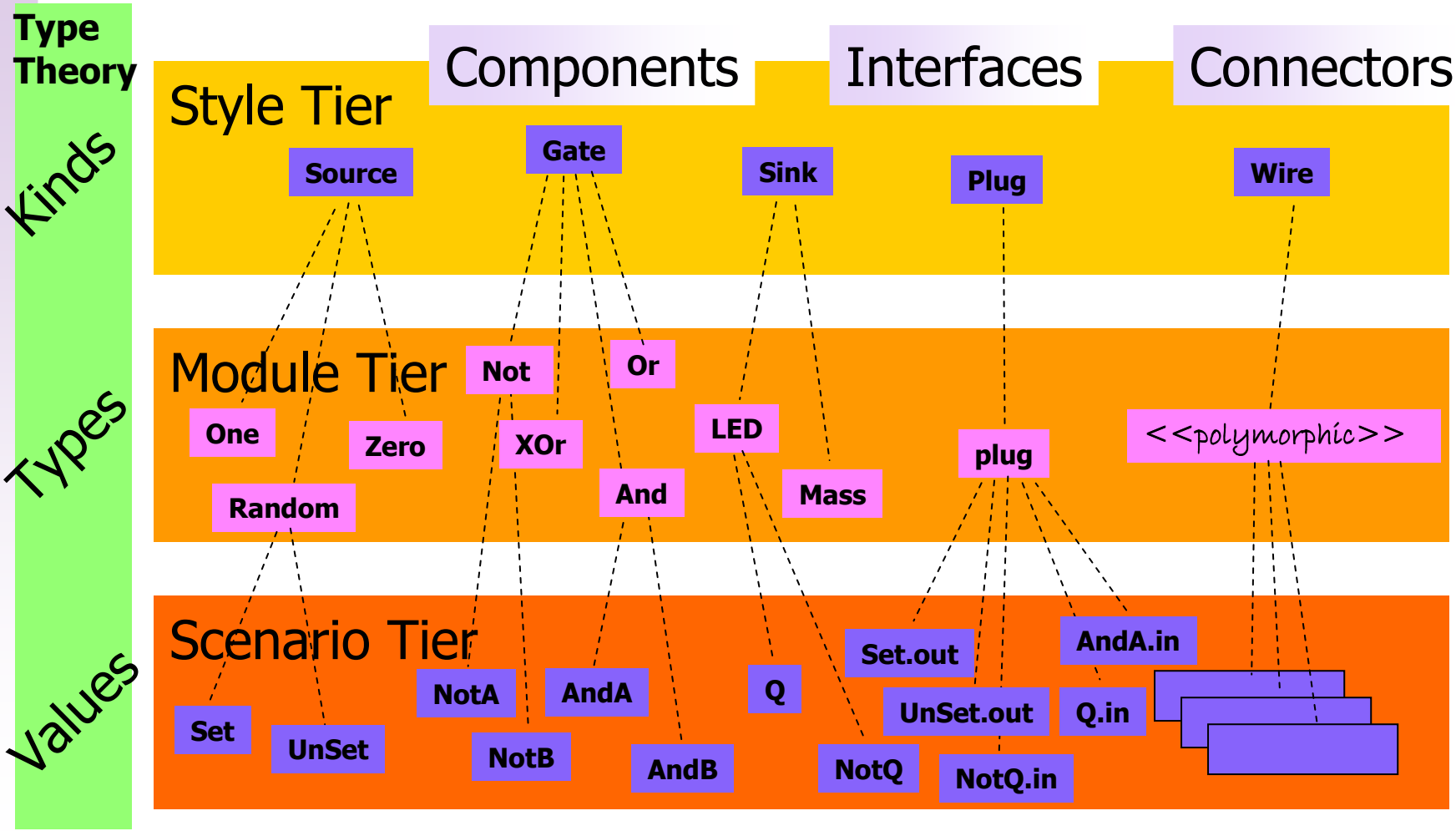
Logical Gates Cadena Model




Resource - BasicElements - Eclipse SDK
File Edit Navigate Search Project Run Editor Menu Window Help
Navigator
>LogicalGates [projects.cis.ksu.edu]
CVS
>specification
CVS
module
BasicElements.module 1.1
BasicElements.module.view
>scenario
>style
.cadenaconfig 1.1 (ASCII -kkv)
.project 1.1 (ASCII -kkv)
>nesC-example [projects.cis.ksu.edu
robot
LogicalGates
BasicElements
Module Detail View
Component Types
Name Kind Parent Type
And Gate
a input plug
b input plug
out output plug
Not Gate
Or Gate
XOr Gate
Result Sink
in input plug
One Source
out output plug
Random Source
Zero Source
Interface Types
Name Kind
plug Plug
Overview Table
Outline
And
out
a
b
Tasks Filters Properties
Property Value
Core
abstract false
Function <Name> and
name And

Types from Kinds

Declaring Types use vocabulary defined by Kinds



Summary

- Cadena provides a 3-tiered modeling approach that emphasize the notion of an architectural style a mechanism for defining domain-specific component modeling languages (DSCML)
- Plug-ins can be associated with styles to give various interpretations or meanings to types and values within a particular DSCML
- Styles can be arranged in hierarchies to capture (to some degree) architectural refinements, PIM to PSM mappings, changes in levels of abstraction