An Object-Z based Metamodel for Wright

Joint work with M. Maouche[2], and M. Mosteghanemi[1]

Presented by M. Bettaz[1]

[1]MESRS/ESI Algeria
[2]Philadelphia University Jordan

Motivation

- Wright:
    - a component model designed for formal description of software architecture.
    - defined by an ADL (Architecture Description Language).
- Our interest in metamodeling of Wright is motivated by:
    - The regain of interest in software architectural models supporting connectors (S. Kell, Rethinking Software Connectors, 2007),
    - Wright is considered as a reference for formal architectural models,
    - Wright provides support for connectors,
    - Many component systems are leaving ADL-based definitions for metamodel based definitions (PALLADIO, PRISMA, SOFA 2, etc.).
- Benefits:
    - Semi-automated creation of the development supporting tools.

Using of Object-Z

- On one hand
  - OMG has defined the MOF (Meta-Object Facilities) as a standard,
  - MOF 2.x may be seen as a subset of UML 2.x,
  - To get more precise descriptions, an association of MOF and OCL (Object Constraint Language) is used,
    - OCL is based on first-order logic.
- On the other hand
  - Transformation approaches from UML to Object-Z exist,
  - Object-Z is based on set and first-order logic.
- This precisely motivates our use of Object-Z.

Objective

- Build an Object-Z metamodel for Wright.
- Show, through a simplified client-server architecture example, how to derive a Wright model.

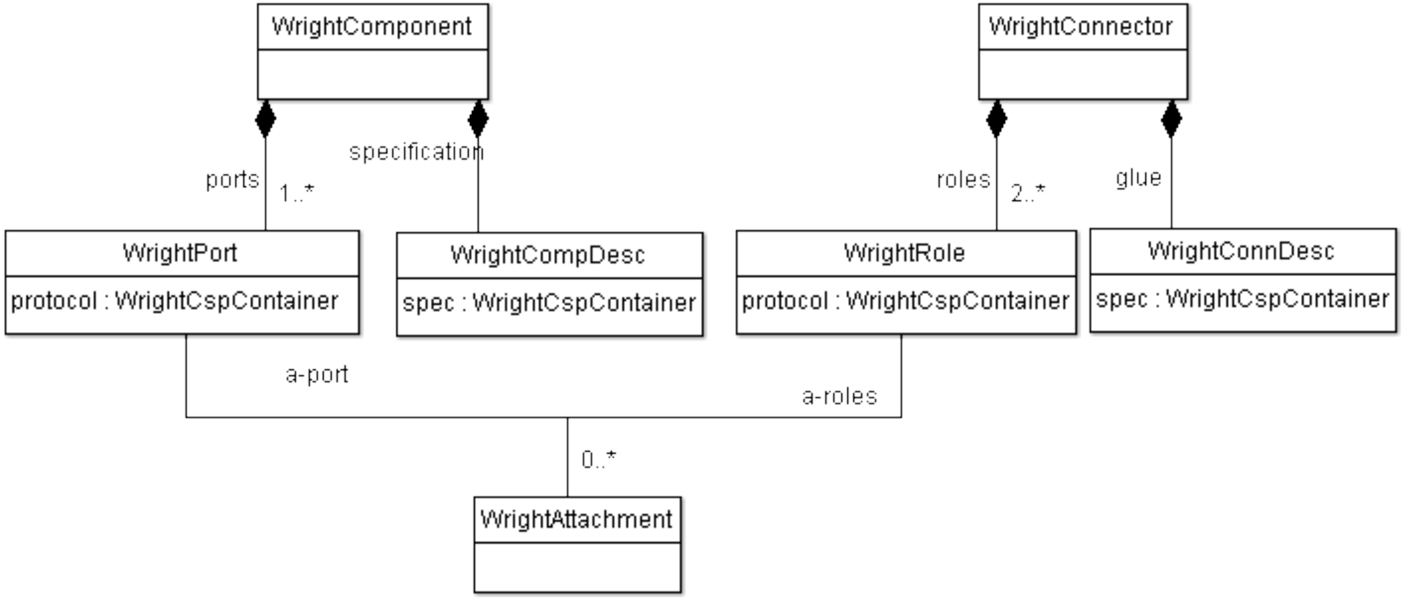
The Approach

- Use of MOF - UML (*without OCL*) as an intermediary notation *(conformity with the standards, reuse of results of works based on MOF).*
- Transform UML metamodels into Object-Z notation to get more formal metamodels, which may be rigorously checked, and formally analysed (*adapting of existing transformation techniques*).

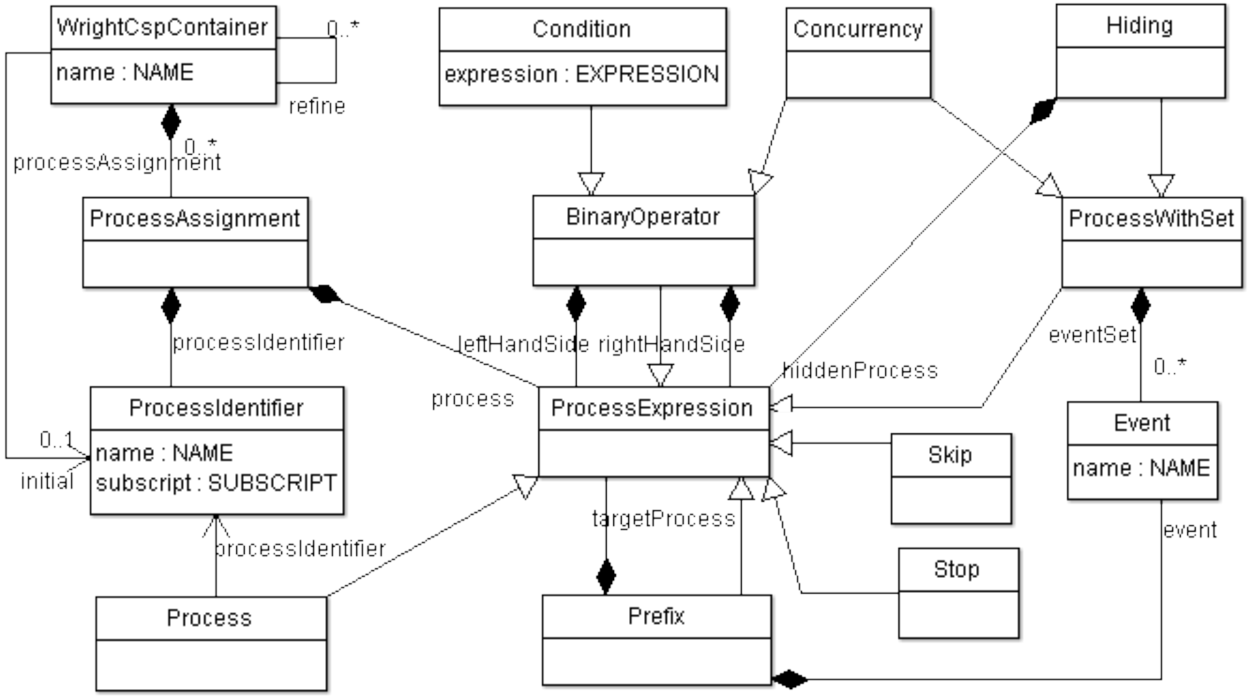
Wright

- The Architectural abstractions:
  - components,
  - connectors,
  - configurations.

A UML Metamodel of Wright Structural Aspects



A UML Metamodel of Wright Behavioral Aspects



From *[D. Bisztray, K. Ehrig, and R. Heckel, Case Study: UML to CSP transformation, 2007]* with slight modifications.

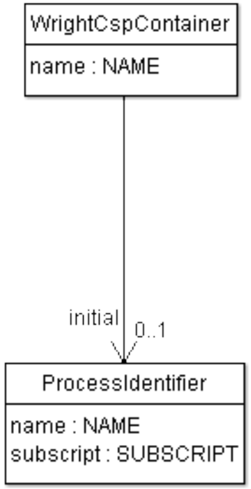
Transformation into Object-Z

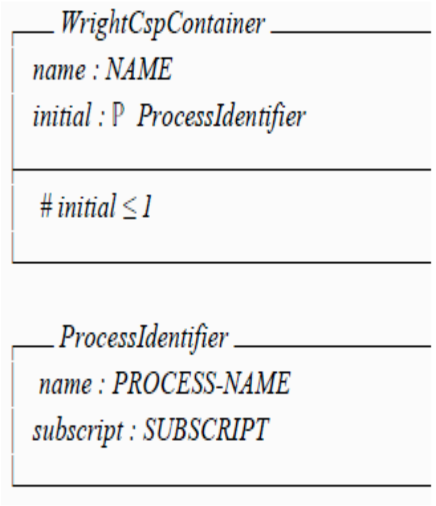
- We use rules defined by Kim - Carrington, and Amalio – Polack (*with some modifications*).
- The UML definitions are based on theUML 1.4 specification.

Classes, attributes and associations

UML	**Object-Z**
UML class	Objet-Z class schema
multi-valued attribute	power-set
multiplicity constraint	predicate
association	class attribute, powerset or simple set *(according to the multiplicity)*
linking of objects from different classes via roles	predicates using the built-in self constant *(holding the implicit identity of the object)*

Classes, attributes and associations Illustration





WrightCspContainer

$name : NAME$

$initial : \mathbb{P}\ ProcessIdentifier$

$\#\, initial \leq 1$

ProcessIdentifier

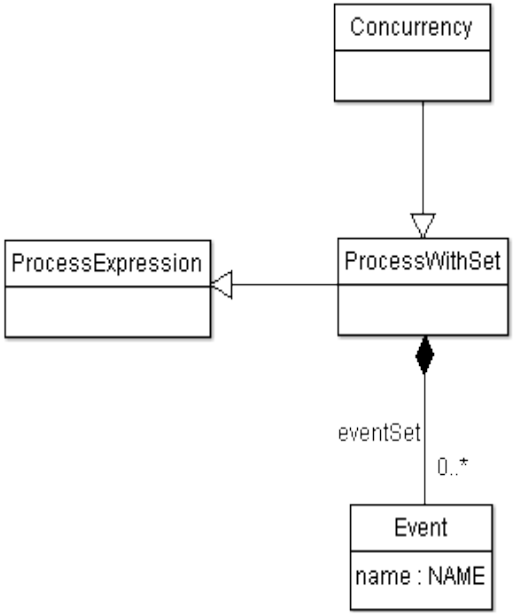
$name : PROCESS\text{-}NAME$

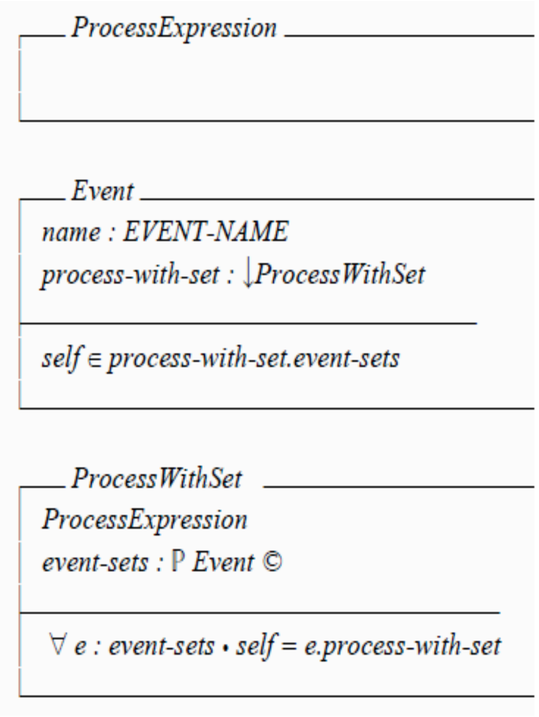
$subscript : SUBSCRIPT$

Generalisation / Specialisations

UML	**Object-Z**
inheritance	Schema inclusion
subtyping	‘enforced’ by polymorphism (*Object-Z inheritance does not imply subtyping*).

Generalisation / Specialisations: Illustration




ProcessExpression

Event
name : EVENT-NAME
process-with-set : ↓ProcessWithSet

self ∈ process-with-set.event-sets

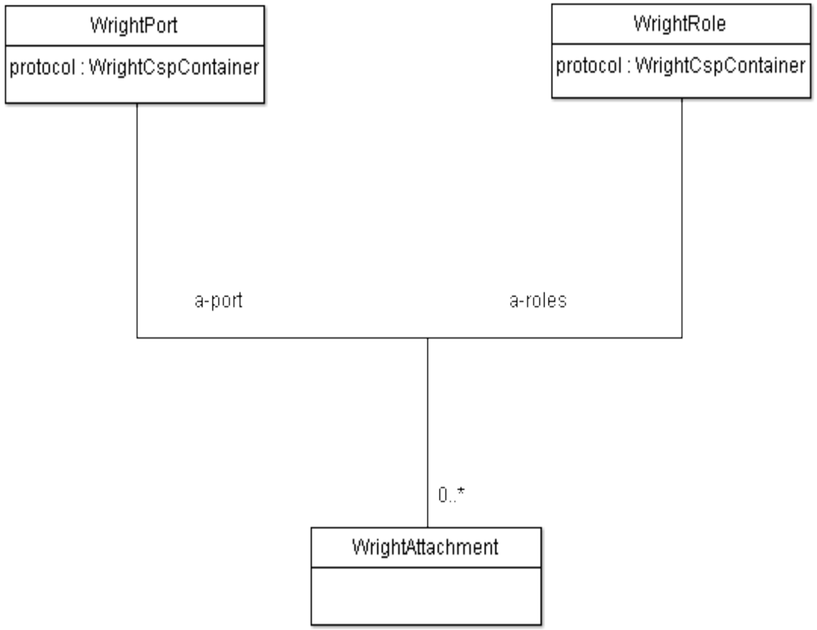
ProcessWithSet
ProcessExpression
event-sets : ℙ Event ©

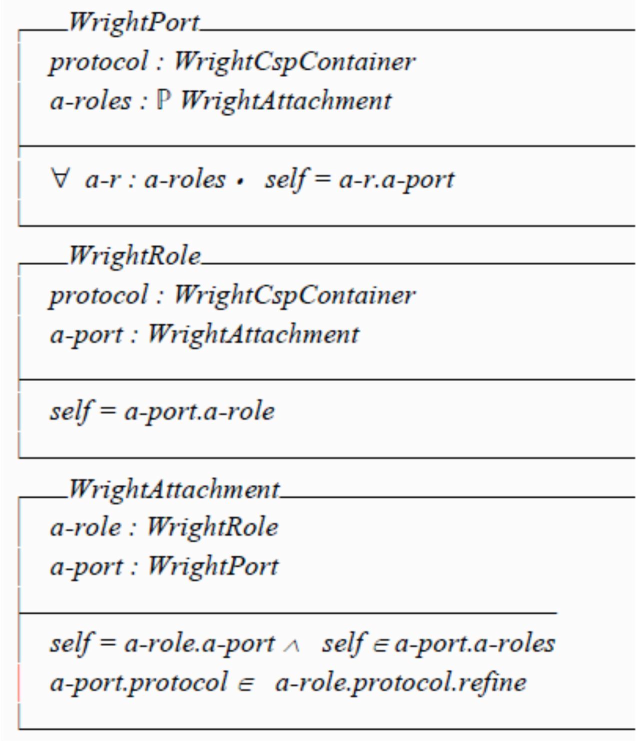
∀ e : event-sets • self = e.process-with-set

Association Classes

UML	**Object-Z**
association-class	a class with two attributes (*representing the ends of the association*)
Association multiplicity	In relation with roles
	Predicates to enforce the semantics (eventuelly)

Association Classes: Illustration





WrightPort

$protocol : WrightCspContainer$

$a\text{-}roles : \mathbb{P}\ WrightAttachment$

$\forall\ a\text{-}r : a\text{-}roles \bullet\ self = a\text{-}r.a\text{-}port$

WrightRole

$protocol : WrightCspContainer$

$a\text{-}port : WrightAttachment$

$self = a\text{-}port.a\text{-}role$

WrightAttachment

$a\text{-}role : WrightRole$

$a\text{-}port : WrightPort$

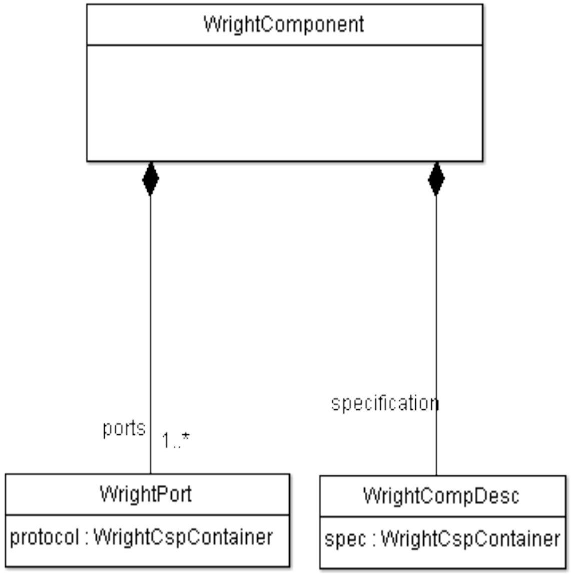
$self = a\text{-}role.a\text{-}port \wedge\ self \in a\text{-}port.a\text{-}roles$

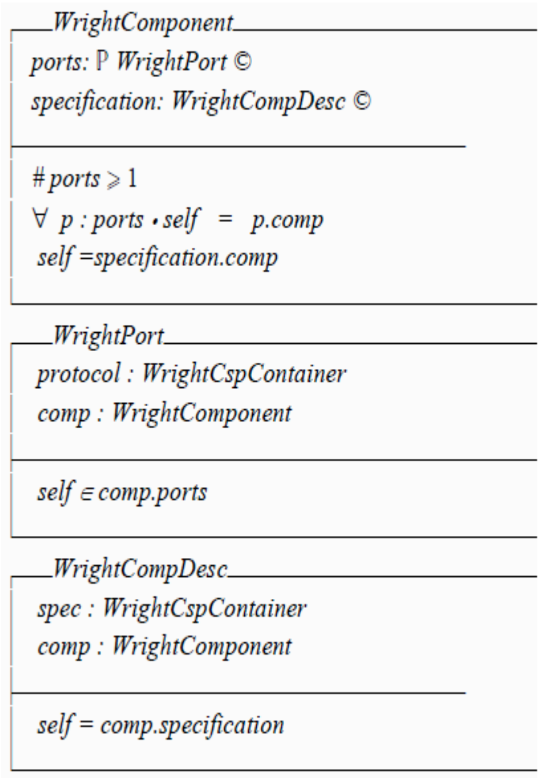
$a\text{-}port.protocol \in\ a\text{-}role.protocol.refine$

Composition

UML	Object-Z
component class, composite class	according to the rules of classes and associations
Containment relationships	Via a ©, attached to the types of attributes and operations.

Composition: Illustration





WrightComponent

ports: $\mathbb{P}$ *WrightPort* ©
specification: *WrightCompDesc* ©

$\# ports \geqslant 1$
$\forall\ p : ports \bullet self = p.comp$
$self = specification.comp$

WrightPort

protocol : *WrightCspContainer*
comp : *WrightComponent*

$self \in comp.ports$

WrightCompDesc

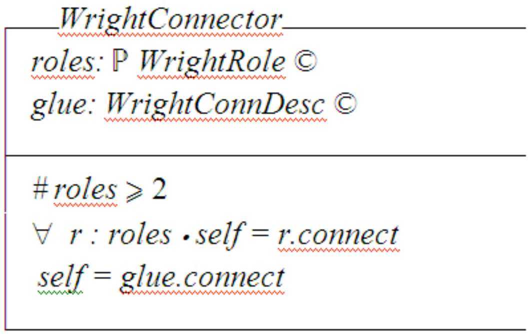
spec : *WrightCspContainer*
comp : *WrightComponent*

$self = comp.specification$

Example: Deriving a Wright client-server model.

- Client-server connector
- Client-server components
- Client-server configuration

1. Client-server connector



WrightConnector

$roles: \mathbb{P}\ WrightRole$ ©

$glue: WrightConnDesc$ ©

$\#roles \geqslant 2$

$\forall\ r : roles \bullet self = r.connect$

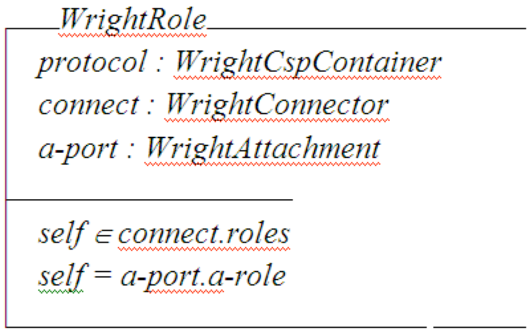
$self = glue.connect$

cs_con: instance of the class <u>WrightConnector</u>

cs_con.roles = {c_role, s_role}

cs_con.glue = cs_glue_desc

The client role



WrightRole

protocol : WrightCspContainer
connect : WrightConnector
a-port : WrightAttachment

self ∈ *connect.roles*
self = *a-port.a-role*

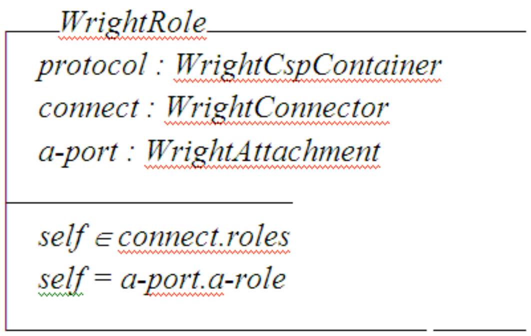
c_role: instance of the class WrightRole

c_role.protocol = crl_proc_cont

c_role.connect = cs_con

c_role.a-port = att_cl_p_cs_con

The server role



WrightRole

protocol : WrightCspContainer
connect : WrightConnector
a-port : WrightAttachment

self $\in$ *connect.roles*
self = *a-port.a-role*

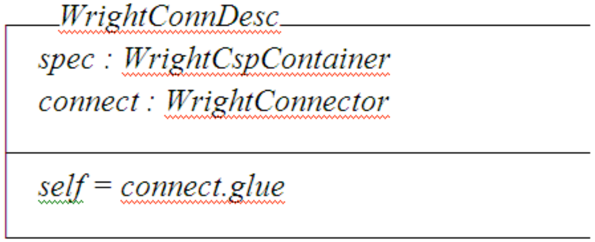
s_role: instance of the class WrightRole

s_role.protocol = srl_proc_cont

s_role.connect = cs_con

s_role.a-port= att_sv_p_cs_cont

The client-server glue



WrightConnDesc

spec : WrightCspContainer
connect : WrightConnector

self = connect.glue

cs_glue_desc : instance of the class WrightConnDesc

cs_glue_desc.spec =

cs_glue_proc_cont

cs_glue_desc.connect = cs_con

Roles and glue protocols

srl_proc_cont: protocol of the server role

events associated to server role : srl_proc_cont : {srl_request,
srl_reply}

process expression: srl_proc_id = srl_request → srl_reply →
srl_proc_id □ §

crl_proc_cont: protocol of the client role

events associated to client role: crl_proc_cont : {crl_request, crl_rep□y}

process expression : crl_proc_id = crl_request → crl_reply →
crl_proc_id §

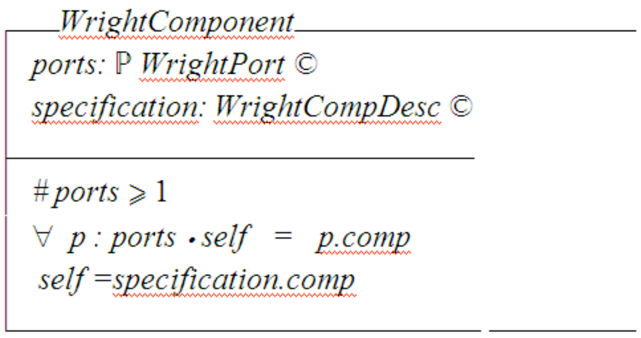
cs_glue_proc_cont: protocol of the client-server glue

events associated to glue : cs_glue_proc_cont : {srl_request, srl_reply, crl_request, crl_reply }

process expression:

cs_glue_proc_id = crl_request → srl_request → cs_glue_proc_id
□
srl_reply → crl_reply →
cs_glue_proc_id □ §

2. Client-server components:
2.1 The Client



WrightComponent

ports: $\mathbb{P}$ *WrightPort* ©

specification: *WrightCompDesc* ©

$\# ports \geqslant 1$

$\forall\ p : ports \bullet self = p.comp$

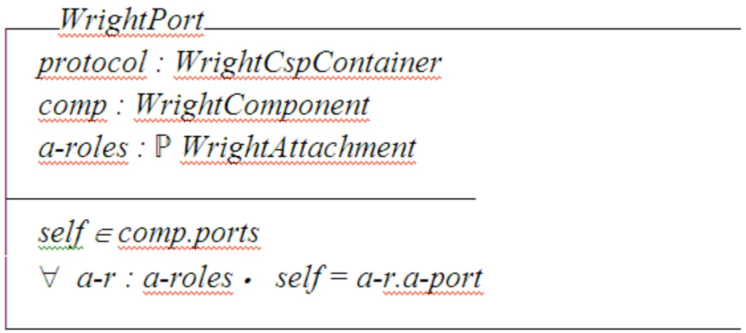
$self = specification.comp$

client : instance of the class WrigthComponent

client.ports ={cl_p}

client.specification = cl_desc

Client port



WrightPort

protocol : *WrightCspContainer*
comp : *WrightComponent*
a-roles : $\mathbb{P}$ *WrightAttachment*

self $\in$ *comp.ports*
$\forall$ *a-r* : *a-roles* • *self* = *a-r.a-port*

cl_p : instance of the class WrithPort

cl_p.protocol = cl_p_proc_cont

cl_p.comp = client

cl_p.a-roles = {att_cl_p_cs_con}

Client port protocol

cl_p_proc_cont: instance of the class WrightCSpContainer

associated events:

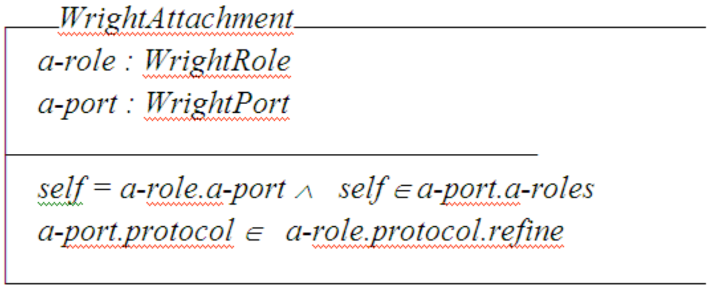
cl_p_request, cl_p_reply

process identifier:

cl_p_proc_id = cl_p_request → cl_p_reply → cl_p_proc_id

§

Client side Attachment



WrightAttachment

a-role : WrightRole
a-port : WrightPort

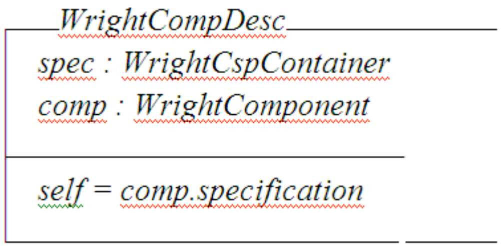
self = a-role.a-port ∧ *self* ∈ *a-port.a-roles*
a-port.protocol ∈ *a-role.protocol.refine*

att_cl_p_cs_con: instance of the class WrightAttachent

att_cl_p_cs_con.a-role = c_role

att_cl_p_cs_con.a-port = cl_p

Client component description



WrightCompDesc

spec : WrightCspContainer
comp : WrightComponent

self = comp.specification

cl_desc: instance of the class WrightCompDesc

cl_desc.spec = cl_comp_proc_cont

cl_desc.comp = client

Client component behavior

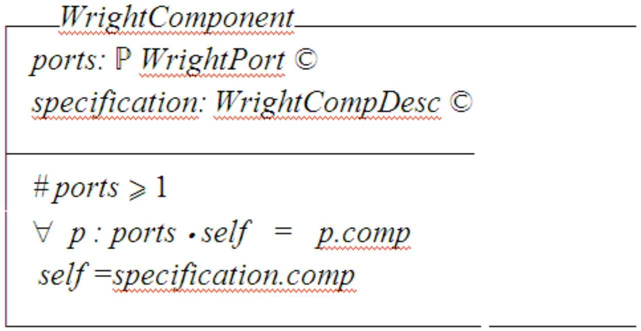
cl_comp_proc_cont : instance of the class WrigthCspContainer

associated events: internalCompute, cl_p_request, cl_p_reply

process identifier: __________

cl_comp_proc_id = internalCompute → cl_p_request →
cl_p_reply → ⊓
cl_comp_proc_id §

2. Client-server components: 2.2 The Server



WrightComponent

ports: $\mathbb{P}$ *WrightPort* ©

specification: *WrightCompDesc* ©

$\# ports \geqslant 1$

$\forall\ p : ports \bullet self = p.comp$

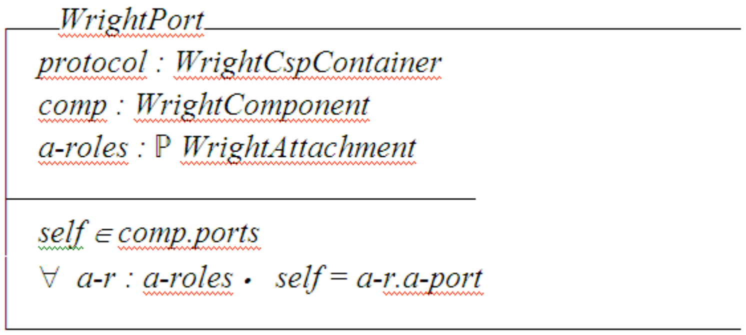
$self = specification.comp$

server : instance of class WrigthComponent

server.ports ={sv_p}

server.specification = sv_desc

Server port



WrightPort

protocol : *WrightCspContainer*

comp : *WrightComponent*

a-roles : $\mathbb{P}$ *WrightAttachment*

$self \in comp.ports$

$\forall\ a\text{-}r : a\text{-}roles \bullet\ self = a\text{-}r.a\text{-}port$

sv_p : instance of the class
WrithPort

sv_p.protocol =
sv_p_proc_cont

sv_p.comp = server

sv_p.a-roles =
{att_sv_p_cs_con}

Server port protocol

sv_p_proc_cont: instance of the class WrightCSpContainer

associated events:

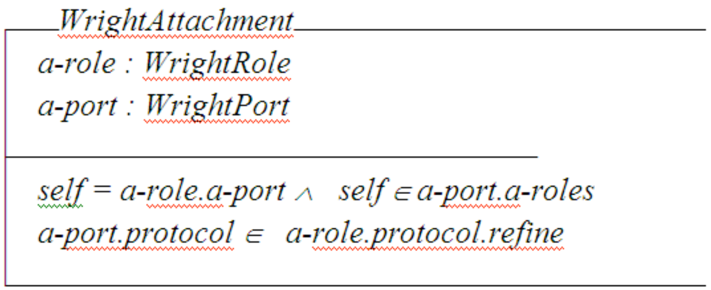
sv_p_request, sv_p_reply

process identifier: ________

sv_p_proc_id = sv_p_request → sv_p_reply → sv_p_proc_id

□ §

Server side Attachment



WrightAttachment

a-role : WrightRole
a-port : WrightPort

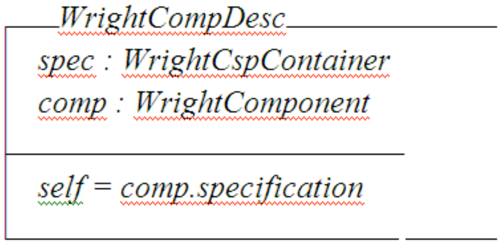
$self = a\text{-}role.a\text{-}port \wedge self \in a\text{-}port.a\text{-}roles$
$a\text{-}port.protocol \in a\text{-}role.protocol.refine$

att_sv_p_cs_con: instance of the class WrightAttachent

att_sv_p_cs_con.a-role = s_role

att_sv_p_cs_con.a-port = sv_p

Server Component description



WrightCompDesc

spec : WrightCspContainer

comp : WrightComponent

self = comp.specification

sv_desc: instance of the class WrightCompDesc

sv_desc.spec = sv_comp_proc_cont

sv_desc.comp = server

Server Component behavior

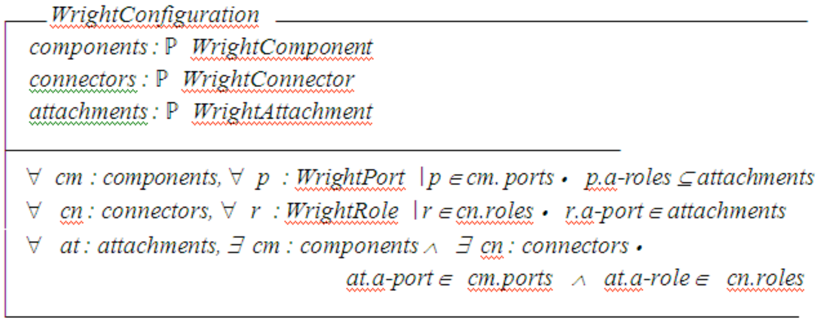
sv_comp_proc_cont : instance of the class WrigthCspContainer

associated events: internalCompute, sv_p_request, sv_p_reply

process identifier: ________

sv_comp_proc_id = sv_p_request → InternalCompute →
sv_p_reply → sv_comp_proc_id □ §

Client-server configuration



WrightConfiguration

$components : \mathbb{P}\ WrightComponent$

$connectors : \mathbb{P}\ WrightConnector$

$attachments : \mathbb{P}\ WrightAttachment$

$\forall\ cm : components, \forall\ p : WrightPort \mid p \in cm.ports \bullet p.a\text{-}roles \subseteq attachments$

$\forall\ cn : connectors, \forall\ r : WrightRole \mid r \in cn.roles \bullet r.a\text{-}port \in attachments$

$\forall\ at : attachments, \exists\ cm : components \wedge \exists\ cn : connectors \bullet$
$at.a\text{-}port \in cm.ports \wedge at.a\text{-}role \in cn.roles$

cl_sv_conf : instance of the class WrigthConfiguration

cl_sv_conf.components = {client, server}

cl_sv_conf.connectors = {cs_con}

cl_sv_conf.attachements = {
att_cl_p_cs_con,
att_sv_p_cs_con}

Concluding remarks

- Checking the validity of the built metamodel.
  - directly? How?
  - Indirectly: through a mapping between our metamodel and a ‘valid’ metamodel of Wright, built for instance using UML or graph transformation?
- Checking the validity of a Wright model.
  - Might be done by deriving (automatiquely) an instance of our meta-model, and showing that the derived instance satisfies the predicates specified in our meta-model.

Some References

1. R. Allen, A Formal Approach to Software Architecture, PhD thesis, 1997
2. S-K. Kim and D. Carrington, A Formal Mapping between UML Models and Object-Z Specifications, 2000
3. N. Amalio and F. Polack, Comparison of Formalisation Approaches of UML Class Constructs in Z and Object-Z, 2002
4. D. Roe et al., Mapping UML Models incorporating OCL Constraints into Object-Z, 2003
5. J. Ivers et al., Documenting Component and Connector Views with UML 2.0, 2004
6. P. Hentynka, F. Plasil, The Power of MOF-based Mata-modeling of Components, 2005
7. M. Navarčík, Using UML with OCL as ADL, 2005
8. M. Bettaz, M. Maouche, Towards Mobile Z Schemas, 2005
9. S. Kell, Rethinking Software Connectors, 2007
10. D. Bisztray, K. Ehrig, and R. Heckel, Case Study: UML to CSP transformation, 2007
11. M. Bettaz, M. Maouche & R. Heckel, From Graph Transformation to Z Notation, 2008,