The Component System …
a bit further

Contact: werner.van.belle@vub.ac.be

Catch up

- Component System
  - Asynchronous message delivery
  - Scheduling
  - Naming Service
- Component
  - Own code & data space
  - Reactive

Sending Messages ?

→ To contact another component the Httpd can send a message to another component using the special .. Notation

```
...{...
message RespondTo()
  {dispatcher(URL)..GenerateHtml(<Url:URL>);}
...}
```

Use of the Architecture

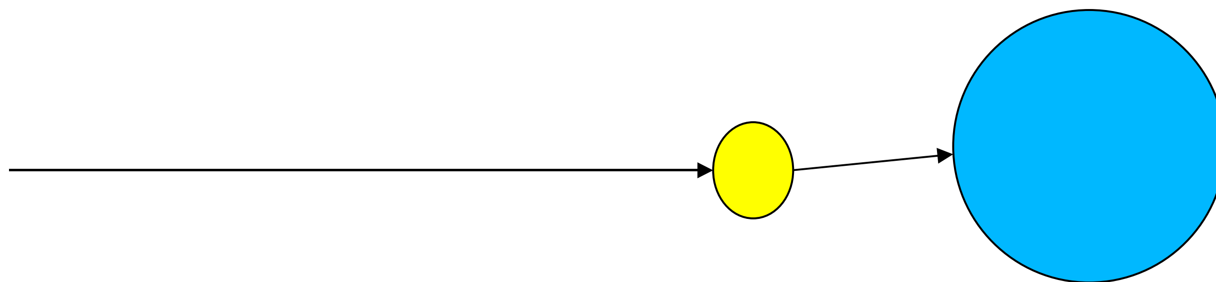
- Implementing timing contracts
- The generation of ‘glue’ components
- The generation of interface adaptors

Problems we encountered

- Unable to *monitor* components
- Unable to filter messages *towards* components
- Unable to filter messages *from* components

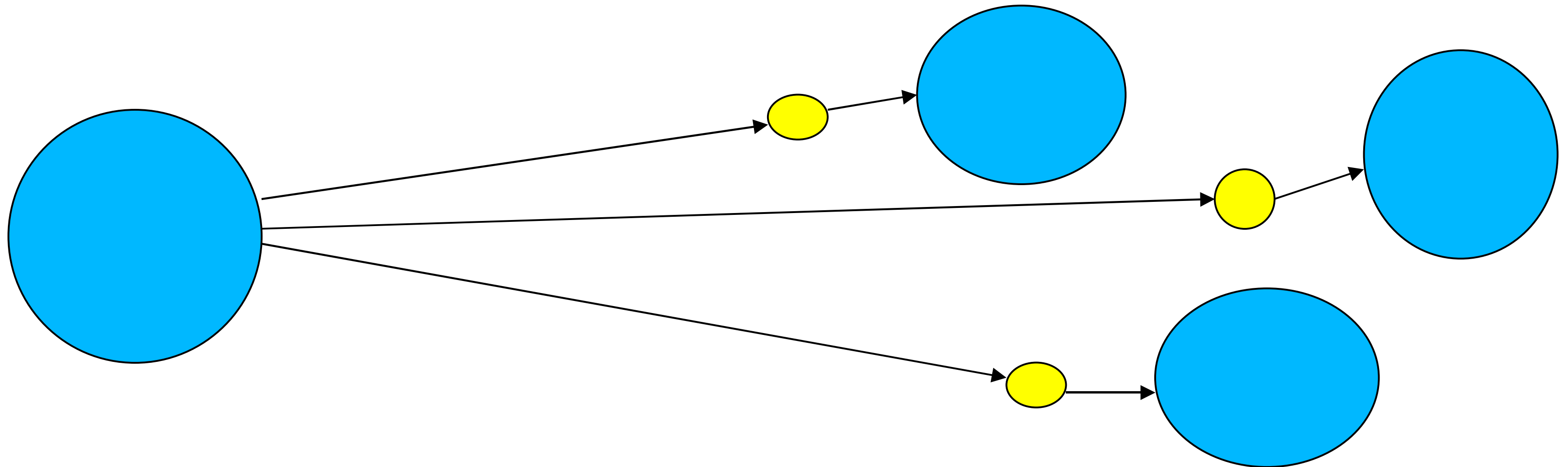
Filtering messages to a component

- Write a stub, with the same interface
  - Difficult to adapt at runtime
  - Need a stub object for every real object



Filtering Messages from a component

- Place stubs at all possible receivers of the object
  - Requires a lot of stub-management
  - Very difficult if receivers already exists



What we need

- Better meta level architecture
  - Dynamic incoming wrappers
  - Dynamic outgoing wrappers
  - Write these wrappers as components

Proxy-tables (i)

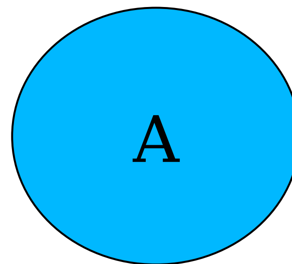
- Before sending a message, the CS checks whether there is a proxy.
- If there is one, we wrap the message in an SendMessage(<Message:..>) message and queue it.
- Otherwise the message goes to the scheduler

Proxy-tables (ii)

- Before receiving a message, the CS checks whether there is a proxy.
- If there is one, we wrap the message in an ReceiveMessage(<Message:..>) message and queue it.
- If there is an immediate target, we call receiveMessage upon the target.
- Otherwise, we wrap the message in an Undeliverable(<Message:...>) message and queue it.

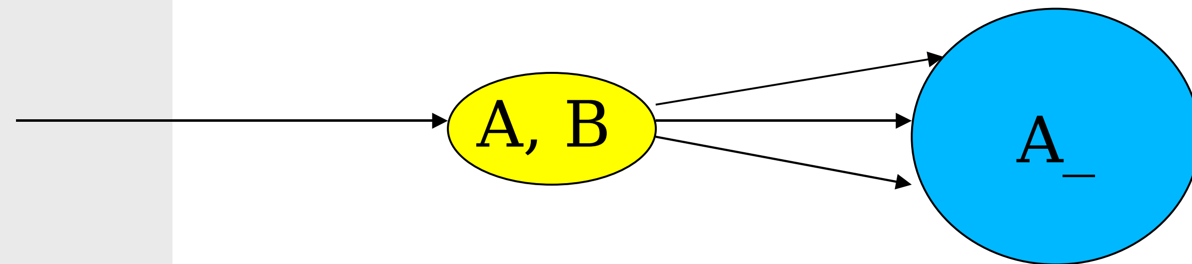
Filtering all Incoming messages (i)

Sender/receiver	Sending Proxy	Receiving Proxy
A	/	/



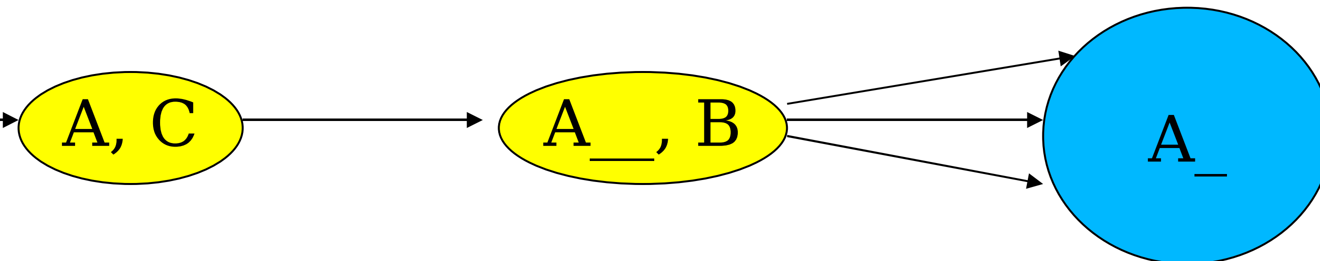
Filtering all Incoming messages (ii)

Sender/receiver	Sending Proxy	Receiving Proxy
A	/	B
A_	/	/



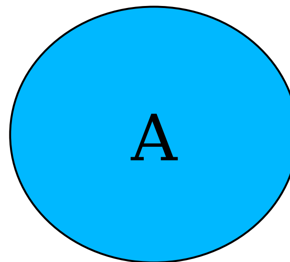
Filtering all Incoming messages (iii)

Sender/receiver	Sending Proxy	Receiving Proxy
A	/	C
A__	/	B
A__	/	/



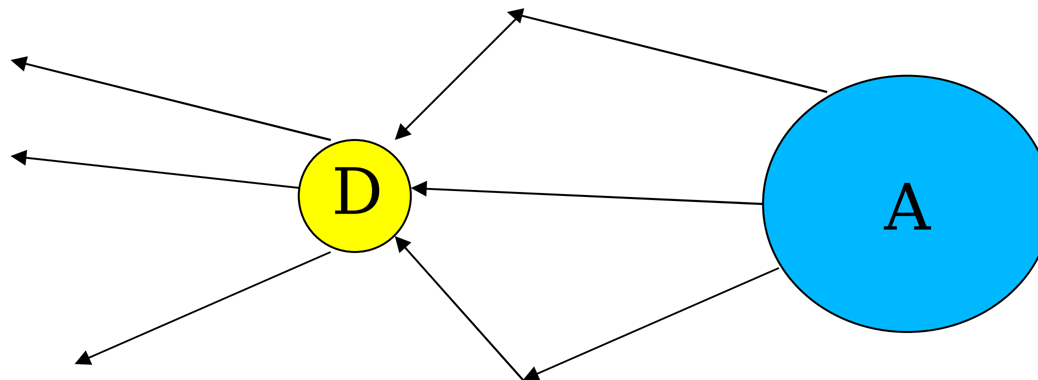
Filtering all Outgoing messages (i)

Sender/receiver	Sending Proxy	Receiving Proxy
A	/	/



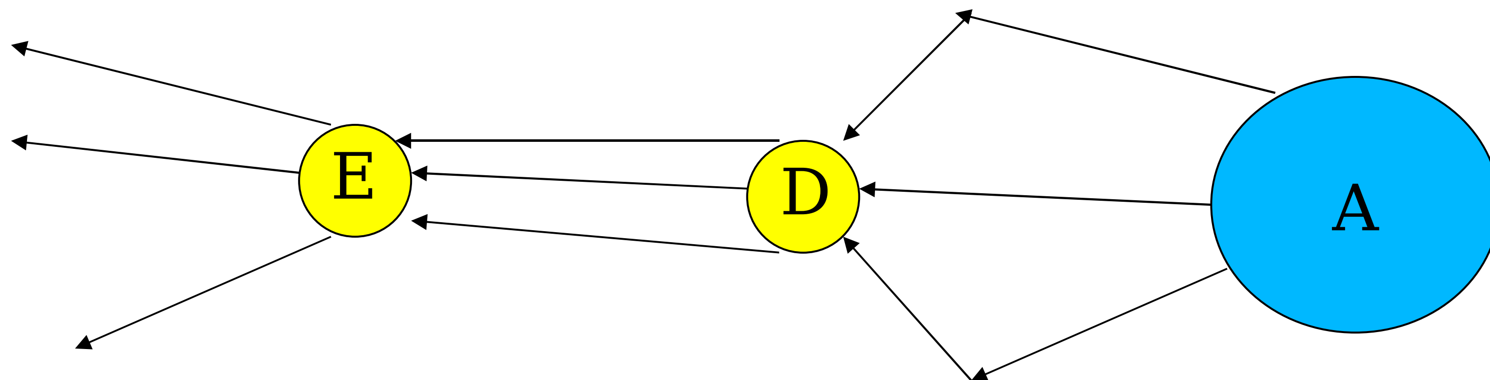
Filtering all Outgoing messages (ii)

Sender/receiver	Sending Proxy	Receiving Proxy
A	D	/



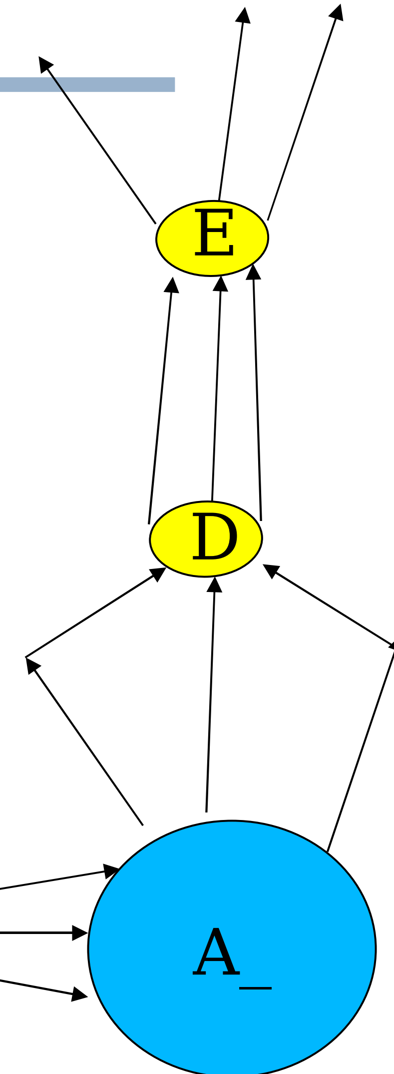
Filtering all Outgoing messages (iii)

Sender/receiver	Sending Proxy	Receiving Proxy
A	D	/
D	E	/



Combining both

sender/receiver	Sending proxy	Recving proxy
A	D	C
A_	/	/
A__	/	B
D	E	/



Meta Level Architecture

- Introspection
  - Look at components
- Absorption
  - Change the behavior of components

Can we change the component system ?

The CS as a Component

- The component system has become a component itself.
  - componentsystem.CreateComponent(…)
  - “ComponentSystem”..CreateComponent(…)

The CS as a Component

- The component system has become a component itself.
  - componentsystem.DestroyComponent(…)
  - “ComponentSystem”..DestroyComponent(…)

The CS as a Component

- The component system has become a component itself except for sendMessage & receiveMessage
  - componentsystem.sendMessage(...)
  - componentsystem.receiveMessage(...)

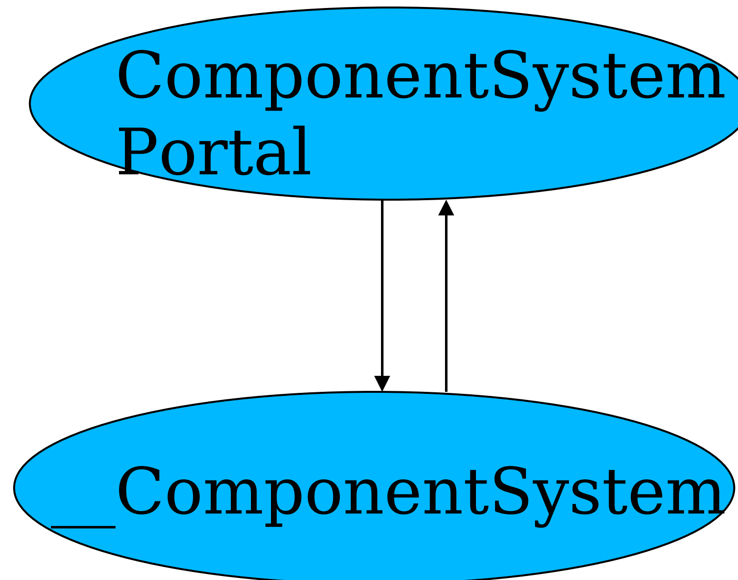
Let’s Try It: The Portal

- When creating a component, the instance name has to be prefixed with the name of the machine
- When sending a messages to a local undeliverable target, we forward it to the effective machine.
- When receiving a forwarded message, we send it through to the actual target

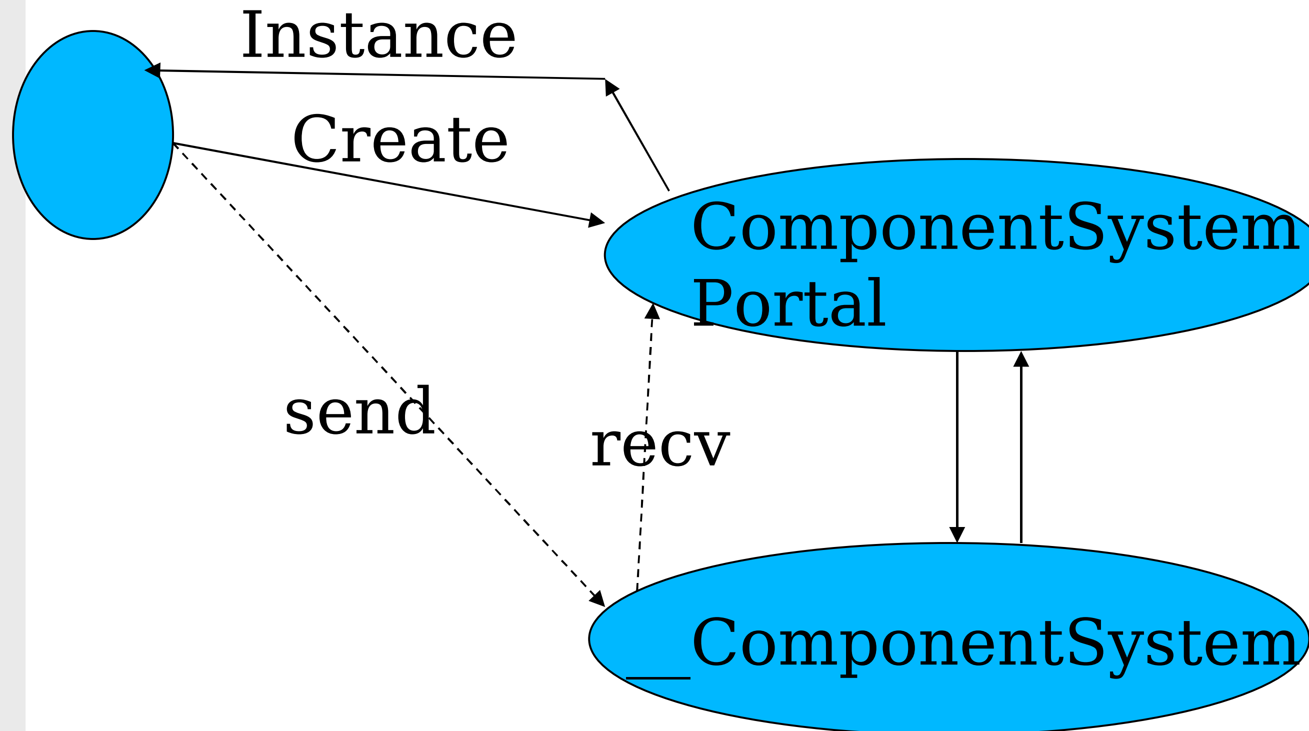
The Portal (i)

ComponentSystem

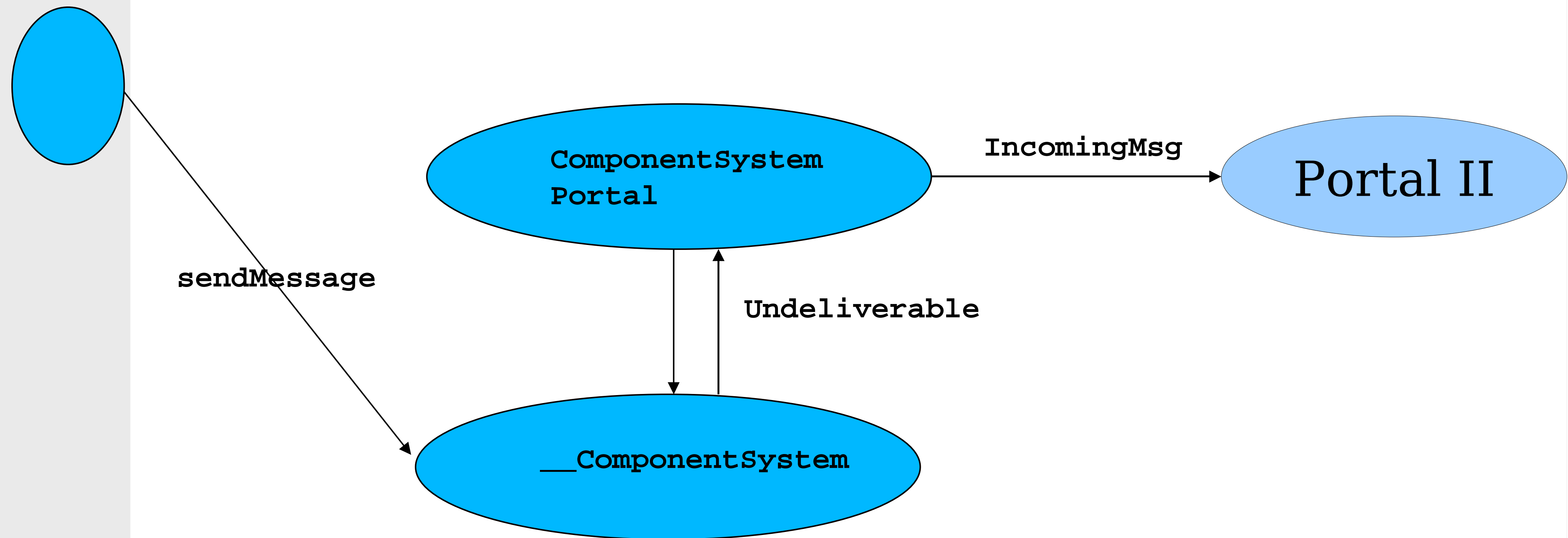
The Portal (i)



The Portal (ii): Creation



The Portal (ii): Forwarding



Advantages of using a CS

- Composing applications by means of glue components
- Much better MLA for distributed environments
- Constraints can be checked
- Semantic information can be extracted
- Plugable architecture:
  - Scheduling can be domain dependent
  - Testing of components on target architecture
  - Contract plugins

The End

Xtra: Threads vs Events (i)

- Some concurrency problems are unavoidable
  - Deadlocks
  - Race Conditions
  - Access to Shared Data

Xtra: Thread vs Events (ii)

- Threads
  - Require a Shared Data Space
  - Locking problems
- Events
  - Easy in distributed environments
  - Adds semantic information
  - *X* Logical processes can be mapped upon *Y* actual processes.

Xtra: Why No Jini ?

- Not suitable for a large number of small components
  - 2 pages to look up a service
  - 2 pages to publish a service
- Java RMI is not suitable as a paradigm
  - Listen … listen … listen…
  - Wait … wait … wait …
- No pluggable architecture
  - Discovery protocol cannot be adapted
  - Error resolution strategy is fixed