Formal Aspects of Software Architecture

J.L.Fiadeiro

University of Leicester

Plan

- **Architectures**:
  - Complexity in software development
  - Physiological vs social complexity
  - Architectures and Programming "in-the-world"
- **CommUnity**
  - Computation vs Coordination
  - Externalisation of interactions
  - Refinement vs Composition

Contributions

- ATX Software

- Antónia Lopes @ University of Lisbon

Software Architecture?

- A software architecture for a system is the structure or structures of the system, which comprise elements, their externally-visible behavior, and the relationships among them.

  source: L.Bass, P.Clements and R.Kazman. Software Architecture in Practice. Addison-Wesley, 1998.

- There are many views, as there are many structures, each with its own purpose and focus in understanding the organisation of the system.

What is it for?

component *(n): a constituent part*

complex *(a): composed of two or more parts*

architecture *(n):*

1 *: formation or construction as, or as if, the result of conscious act;*

2 *: a unifying or coherent form or structure*

A case of "complexity"

A case of "complexity"

in-the-head

mnemonics

result-driven

symbolic information

elementary control flow

- "One man and his problem…" (and **his program**, and **his machine**)
- The Science of **Algorithms** and **Complexity**
- not so much **Engineering** but more of **Craftsmanship** (one of a kind)
- a case for **virtuosi**

A case of "complexity"

in-the-head

mnemonics

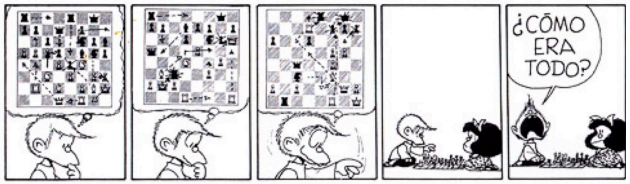
result-driven

symbolic information

elementary control flow

Example

- "One man and his problem…" (and **his program**, and **his machine**)



Using a pocket calculator to select the next move

A case of "complexity"

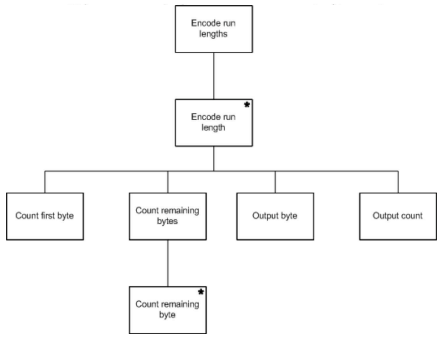
in-the-head	in-the-small
mnemonics	**I/O specs**
result-driven	**algorithms**
symbolic information	**data structures and types**
elementary control flow	**execute once termination**

- The need for commercialisation…
- **"One man and his problem…"** (and **his program**, but **their machine)**
- The Science of **Program Analysis and Construction**
- Commerce, but not yet **Engineering**

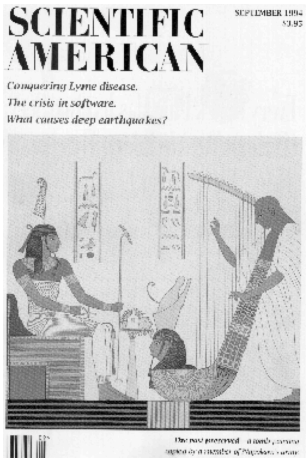
A case of "complexity"

in-the-head	in-the-small
mnemonics	**I/O specs**
result-driven	**algorithms**
symbolic information	**data structures and types**
elementary control flow	**execute once termination**

Program *Architectures*



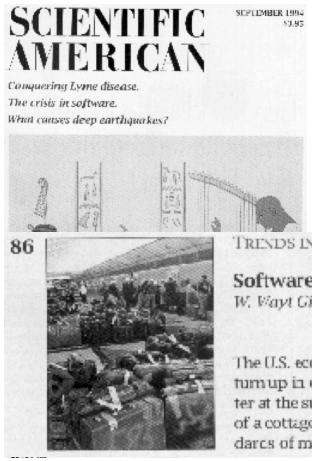
10 years ago, the "software crisis"


SCIENTIFIC AMERICAN

SEPTEMBER 1994
$3.95

Conquering Lyme disease.
The crisis in software.
What causes deep earthquakes?

10 years ago, the "software crisis"


SCIENTIFIC AMERICAN

SEPTEMBER 1994
$3.95

Conquering Lyme disease.
The crisis in software.
What causes deep earthquakes?

86

TRENDS IN COMPUTING

Software's Chronic Crisis

W. Wayt Gibbs, staff writer

The U.S. economy, and indeed all society, has plunged into cyberspace. Computers turn up in everything from toasters and aircraft-control systems to the cash register at the supermarket checkout. Yet software remains largely the custom product of a cottage industry. Can it ever be manufactured so that it meets industrial standards of mass production and reliability?

10 years ago, the "software crisis"

- *The challenge of complexity is not only large but also growing. […]. To keep up with such demand, programmers will have to change the way that they work. "**You can't build skyscrapers using carpenters**," Curtis quips.*
- *[…] Musket makers did not get more productive until Eli Whitney figured out how to manufacture interchangeable parts that could be assembled by any skilled workman. In like manner, software parts can, if properly standardized, be **reused** at many different scales.*
- *[…]In April, NIST announced that it was creating an Advanced Technology Program to help engender **a market for component-based software**.*

A case of "complexity"

in-the-head	in-the-small	in-the-large
mnemonics	I/O specs	**complex specs**
result-driven	algorithms	**system modules**
symbolic information	data structures and types	**databases, persistence**
elementary control flow	execute once termination	**continuous execution**

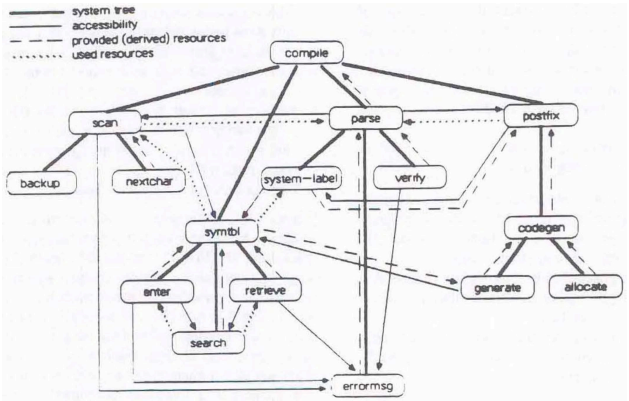
- **"One man and his problem…"** (but **their programs**)
- The Science of **Software Specification and Design**
- **Engineering**

The case for MILs

- **Modelling Interconnection Languages** for programming-in-the-large (DeRemer and Kron 75)
- Address the global structure of a system in terms of
  - what its modules and resources are
  - how they fit together in the system
- Interconnection may be data or control oriented
- Descriptions are concise, precise and verifiable

The case for MILs

Architectures of Usage



The case for new mathematics

The case for new mathematics

11

- **Algebraic techniques for structuring specifications**
  - "Putting Theories together to Make Specifications"
  - The theory of **Institutions**
  - The role of **Category Theory**

ICFEM 2005

University of Leicester

The case for new mathematics

11

- **Algebraic techniques for structuring specifications**
  - "Putting Theories together to Make Specifications"
  - The theory of **Institutions**
  - The role of **Category Theory**

- **Temporal logics** for continuous/reactive execution

ICFEM 2005

University of Leicester

The case for objects/components

- *In like manner, software parts can, if properly standardized, be **reused** at many different scales.*
- *[…]In April, NIST announced that it was creating an Advanced Technology Program to help engender **a market for component-based software**.*

ICFEM 2005

University of Leicester

The case for objects/components

- *In like manner, software parts can, if properly standardized, be **reused** at many different scales.*
- *[…]In April, NIST announced that it was creating an Advanced Technology Program to help engender **a market for component-based software**.*

- Builds on a powerful methodological metaphor – **clientship**

The case for objects/components

12

- *In like manner, software parts can, if properly standardized, be **reused** at many different scales.*
- *[…]In April, NIST announced that it was creating an Advanced Technology Program to help engender **a market for component-based software**.*

- Builds on a powerful methodological metaphor – **clientship**

ICFEM 2005 University of Leicester

- Inheritance hierarchies for **reuse**

The case for objects/components

12

- Builds on a powerful methodological metaphor – **clientship**
- Inheritance hierarchies for **reuse**

- Software **construction** becomes like child's play

ICFEM 2005 University of Leicester

Yet, in 2003 the crisis was going on

- *Computing has certainly got faster, smarter and cheaper, but it has also become much more complex.*
- *Ever since the orderly days of the mainframe, which allowed tight control of IT, computer systems have become ever more distributed, more heterogeneous and harder to manage. […]*
- *In the late 1990s, the internet and the emergence of e-commerce "broke IT's back". Integrating incompatible systems, in particular, has become a big headache.*
- *applications will no longer be a big chunk of software that runs on a computer but a combination of web services*

ICFEM 2005 University of Leicester

Yet, in 2003 the crisis was going on

- *Computing has certainly got faster, smarter and cheaper, but it has also become much more complex.*
- *Ever since the orderly days of the mainframe, which allowed tight control of IT, computer systems have become ever more distributed, more heterogeneous and harder to manage. […]*
- *In the late 1990s, the internet and the emergence of e-commerce "broke IT's back". Integrating incompatible systems, in particular, has become a big headache.*
- *applications will no longer be a big chunk of software that runs on a computer but a combination of web services*

Yet, in 2003 the crisis was going on

13

- *Computing has certainly got faster, smarter and cheaper, but it has also become much more complex.*
- *Ever since the orderly days of the mainframe, which allowed tight control of IT, computer systems have become ever more distributed, more heterogeneous and harder to manage. […]*
- *In the late 1990s, the internet and the emergence of e-commerce "broke IT's back". Integrating incompatible systems, in particular, has become a big headache.*
- *applications will no longer be a big chunk of software that runs on a computer but a combination of web services*

The Economist, May 10, 2003

ICFEM 2005 University of Leicester

Yet a case of "complexity"?

in-the-head	in-the-small	**in-the-large**
mnemonics	I/O specs	complex specs
result-driven	algorithms	system modules
symbolic information	data structures and types	databases, persistence
elementary control flow	execute once termination	continuous execution

- **"One man and his problem…"** (but **their programs**)
- The Science of **Software Specification and Design**
- **Engineering**

Yet a case of "complexity"?

14

in-the-head	in-the-small	**in-the-large**
mnemonics	I/O specs	complex specs
result-driven	algorithms	system modules
symbolic information	data structures and types	databases, persistence
elementary control flow	execute once termination	continuous execution

- "One man and his problem…" (but their programs)
- **"One man and everybody's problems…"**

ICFEM 2005 University of Leicester

A case of "complexity"

in-the-head	in-the-small	in-the-large	**in-the-world**
mnemonics	I/O specs	complex specs	**evolving**
result-driven	algorithms	system modules	**sub-systems & interactions**
symbolic information	data structures and types	databases, persistence	**separation data computation**
elementary control flow	execute once termination	continuous execution	**distribution & coordination**

Same complexity?

Same complexity?

- **"Physiological" complexity**
  - derives from the need to account for problems/situations that are "complicated" in the sense that they offer great difficulty in understanding, solving, or explaining
  - there is nothing necessarily wrong or faulty in them; they are just the unavoidable result of a necessary combination of parts or factors

Same complexity?

- **"Physiological" complexity**
  - derives from the need to account for problems/situations that are "complicated" in the sense that they offer great difficulty in understanding, solving, or explaining
  - there is nothing necessarily wrong or faulty in them; they are just the unavoidable result of a necessary combination of parts or factors
- **"Social" complexity**
  - derives from the number and "open" nature of interactions that involve "autonomic" parts of a system;
  - it is almost impossible to predict what properties can emerge and how they will evolve as a result of the interactions in place or the dynamics of the population itself.

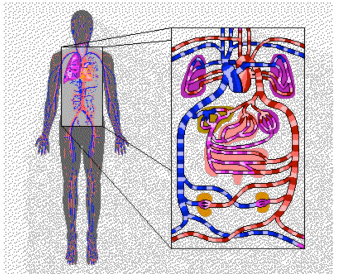
Same Science & Engineering?

- "Physiological" complexity
  - server-to-server, static, linear interaction based on identities
  - compile or design time integration
  - **architectures of usage**
  - **product structure**

Same Science & Engineering?

17

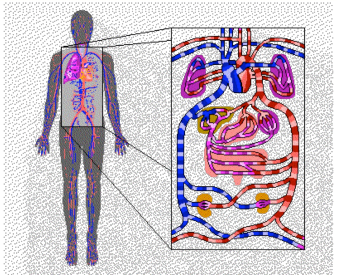
- "Physiological" complexity
  - server-to-server, static, linear interaction based on identities
  - compile or design time integration
  - **architectures of usage**
  - **product structure**



Same Science & Engineering?

17

- "Physiological" complexity
  - server-to-server, static, linear interaction based on identities
  - compile or design time integration
  - **architectures of usage**
  - **product structure**

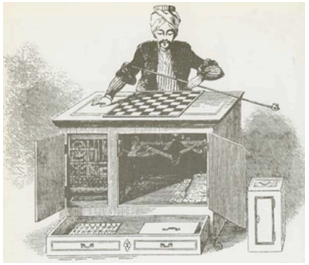


- "Social" complexity
  - dynamic, mobile and unpredictable interactions based on properties
  - "late" or "just-in-time" integration
  - **contracts of interaction**
  - **evolving structure**

Same Science & Engineering?

17

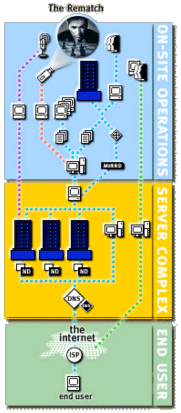
- "Physiological" complexity
  - server-to-server, static, linear interaction based on identities
  - compile or design time integration
  - **architectures of usage**
  - **product structure**



- "Social" complexity
  - dynamic, mobile and unpredictable interactions based on properties
  - "late" or "just-in-time" integration
  - **contracts of interaction**
  - **evolving structure**

Same Science & Engineering?

- **"Physiological" complexity**
  - server-to-server, static, linear interaction based on identities
  - compile or design time integration
  - **architectures of usage**
  - **product structure**
- **"Social" complexity**
  - dynamic, mobile and unpredictable interactions based on properties
  - "late" or "just-in-time" integration
  - **contracts of interaction**
  - **evolving structure**



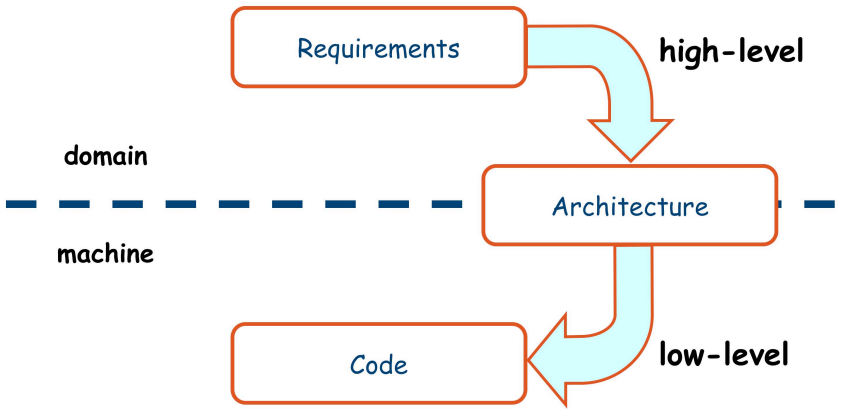
Two different relationships

- **Implements**
  - a given module is defined in terms of facilities provided by/to other modules;
  - composition mechanisms glue pieces together by indicating for each use of a facility where its corresponding definition is provided
- **Interacts**
  - components are treated as independent entities that may interact with each other along well defined lines of communication (connectors)

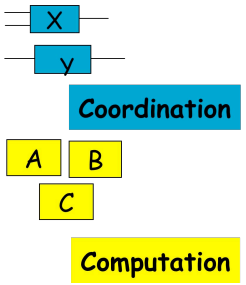
Run-time Architectures

- The Components&Connectors view
- The "Interacts" relationship
- One generation later
  - Perry and Wolf (92)
  - Shaw and Garlan (96)
  - Bass, Clements, Kazman (98)
- Partly inspired by (civil) architects (Alexander)

Architectures in Software Design

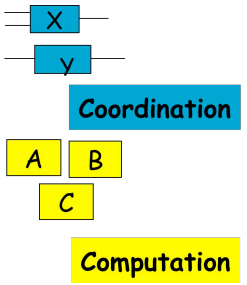


Architecture-based approaches



Summary 21

Architecture-based approaches

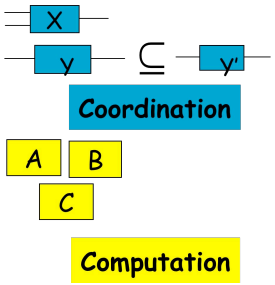


Compositionality wrt refinement

ICFEM 2005 University of Leicester

Summary 21

Architecture-based approaches

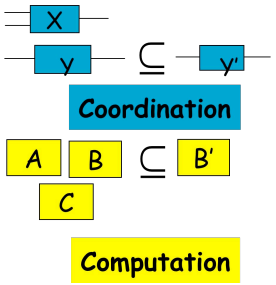


Compositionality wrt refinement

ICFEM 2005 University of Leicester

Summary 21

Architecture-based approaches

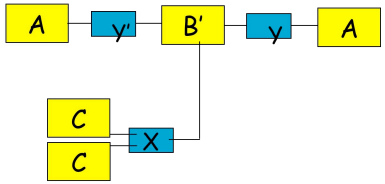


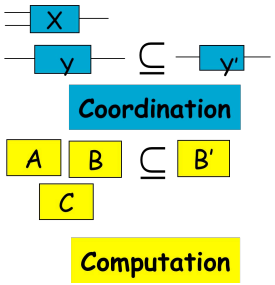
Compositionality wrt refinement

ICFEM 2005 University of Leicester

Summary

Architecture-based approaches





Compositionality wrt refinement
wrt evolution

Need for formality

- **Architectures = Box & Lines ?**
  - is there a shared understanding of what they mean?
  - how easy is it to communicate details ("up" and "down")?
  - what degree of analytic leverage are we given?
  - how informed are we for selecting among alternatives?
- **We need a formal approach supporting**
  - abstraction: capturing the essential
  - precision: knowing what exactly is being addressed
  - analysis: predicting what properties will emerge
  - refinement: coding according to standard reference models
  - automation: tool support

How?

- Architecture description languages (ADLs) have been proposed as a possible answer
- Several prototype ADLs and supporting tools have been proposed

Rapide	events with simulation and animation
UniCon	emphasizing heterogeneity and compilation
Wright	formal specification of connector interactions
Aesop	style-specific arch design languages
Darwin	service-oriented architectures
SADL	SRI language emphasizing refinement
Meta-H	arch description for avionics domain
C-2	arch style using implicit invocation
ACME	open-ended approach ("XML for architectures")

Purpose of ADLs

- An ADL is a language that provides features for modelling a software system's conceptual architecture, at least:
  - components
  - connectors
  - configurations
- The purpose of an ADL is to
  - provide models, notations, and tools to describe components and their interactions
  - support large-scale, high-level designs
  - support principled selection and application of architectural paradigms

CommUnity

- Not a full-fledged ADL
  - its purpose is not to support large-scale, industrial architectural design
  - but to serve as a test bed for formalising architectural notions and techniques
  - and a prototype for extensions (e.g. mobility)
  - but has found its way into industrial practice
- Full mathematical semantics
  - the semantics is largely "language independent"
  - supports reasoning and prototyping
  - supports heterogeneity (based on General Systems Theory)

Origins

A confluence of contributions from

- (Re)Configurable Distributed Systems
  - exoskeletal software
- Parallel Program Design
  - superposition
- Coordination Models and Languages
  - separation of concerns (Computation / Coordination)
- The categorical imperative
  - Goguen's approach to General Systems Theory

Architectural elements

- **Components**
  - model entities/devices/machines (software or "real world"), that keep an internal state, perform computations, and are able to synchronise with their environment and exchange information through channels
  - "designs" given in terms of communication channels and actions
- **Connectors**
  - model entities whose purpose is to coordinate interactions between components
  - "structured designs" given in terms of a "glue" and collection of "roles" (as in Wright)
  - can be superposed at run-time over given components
- **Configurations**
  - diagrams in a category of designs as objects and superposition as morphisms;
  - composition (emergent behaviour) given by colimit construction

Designing components

An example

The design of a "naïve" bank account

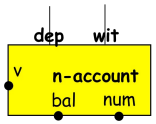
```
design n-account is
   out num:nat, bal:int
   in  v: nat
   do  dep: true → bal:=v+bal
   []  wit: bal≥v → bal:=bal-v
```

Designing components

An example

The design of a "naïve" bank account

```
design n-account is
   out num:nat, bal:int
   in  v: nat
   do  dep: true → bal:=v+bal
   []  wit: bal≥v → bal:=bal-v
```



Channels

- Provide for interchange of data
  - actions do **not** have I/O parameters!
  - reading from a channel does not consume the data!
- **Output** channels `out(V)`
  - allow the environment to observe the state of the component, and for the component to transmit data to the environment
  - the component controls the data that is made available; the environment can only read the data
- **Input** channels `in(V)`
  - allow the environment to make data available to the component
  - the environment controls the data that is made available; the component can only read the data
- **Private** channels `prv(V)`
  - model communication inside (different parts of) the component;
  - the environment can neither read from nor write into private channels

Actions

- Provide for **synchronisation** with the environment (e.g. to transmit or receive new data made available through the channels)
- Provide for the **computations** that make available or consume data

`do g[D(g)] : L(g), U(g) → R(g)`

- **Write** frame `D(g)`
  - the local channels (out, prv) into which the action can write data
- **Computation** `R(g)`
  - how the execution of the action uses the data read on the input channels and changes the data made available on the local channels
- **Guards** `L(g), U(g)`
  - set of states in which the action may be enabled `L(g)`
  - set of states in which the action must be enabled `U(g)`
  - `U(g)` ⊃ `L(g)`

Designing components

Another example

The design of a VIP-account that ***may*** accept a withdrawal when the balance together with a given credit amount is greater than the requested amount, and ***will*** accept any withdrawal for which there are funds available to match the requested amount:

```
design vip-account[CRE:nat] is
   out num: nat, bal:int
   in  v: nat
   do  dep[bal]: true → bal'=v+bal
   []  wit[bal]: bal+CRE≥v, bal≥v → bal'≤bal-v
```

Superposition

- A structuring mechanism for the design of systems that allows to build on already designed components by "augmenting" them while "preserving" their properties.
- Typically, the additional behaviour results from the introduction of new channels and corresponding assignments (that may use the values of the channels of the base design).

Applying Superposition

An example

Extending the design of n-account to control how many days the balance has exceeded a given amount since it was last reset.

```
design e-account[MAX:int] is
  out  num:nat, bal:int, count:int
  in   v,day:nat
  prv  d:int
  do   dep[bal,d,count]: true →
          bal:=v+bal
       ||
          d:=day
       ||
          if bal≥MAX then count:=count+(day-d)
  []   wit[bal,d,count]:
          bal≥v → bal:=bal-v
       ||
          d:=day
       ||
          if bal≥MAX then count:=count+(day-d)
  []   reset [d,count]:
          true, false → count:=0||d:=day
```


Characterising Superposition

The relationship between a design P_1 and a design P_2 obtained from P_1 through the superposition of additional behaviour, can be modelled as a mapping between the channels and actions of the two designs

$$\sigma:P_1 \rightarrow P_2$$

subject to some constraints.

Superposition Morphisms

A superposition morphism $\sigma:P_1 \rightarrow P_2$ consists of

- a total function $\sigma_{ch}:V_1 \rightarrow V_2$ s.t.
  - Sorts, privacy and availability of channels are preserved
  - Input channels may become output channels
- a partial mapping $\sigma_{ac}:\Gamma_2 \rightarrow \Gamma_1$ s.t.
  - Privacy/availability of actions is preserved
  - Domains of channels are preserved

Superposition Morphisms

A superposition morphism $\sigma: P_1 \to P_2$ consists of

- a total function $\sigma_{ch}: V_1 \to V_2$ s.t.
  - $sort_2(\sigma_{ch}(v)) = sort_1(v)$
  - $\sigma_{ch}(out(V_1)) \subseteq out(V_2)$
  - $\sigma_{ch}(in(V_1)) \subseteq out(V_2) \cup in(P_2)$
  - $\sigma_{ch}(prv(V_1)) \subseteq prv(V_2)$

o Sorts, privacy and availability of channels are preserved
o Input channels may become output channels

- a partial mapping $\sigma_{ac}: \Gamma_2 \to \Gamma_1$ s.t.
  - $\sigma_{ac}(sh(\Gamma_2)) \subseteq sh(\Gamma_1)$
  - $\sigma_{ac}(prv(\Gamma_2)) \subseteq prv(\Gamma_1)$
  - $\sigma_{ch}(D_1(\sigma_{ac}(g))) \subseteq D_2(g)$
  - $\sigma_{ac}(D_2(\sigma_{ch}(v))) \subseteq D_1(v)$

o Privacy/availability of actions is preserved
o Domains of channels are preserved

Superposition Morphisms

and, moreover, for every g in Γ_2 s.t. $\sigma_{ac}(g)$ is defined

- $R_2(g) \supseteq \underline{\sigma}(R_1(\sigma_{ac}(g)))$
- $L_2(g) \supseteq \underline{\sigma}(L_1(\sigma_{ac}(g)))$
- $U_2(g) \supseteq \underline{\sigma}(U_1(\sigma_{ac}(g)))$

o Effects of actions must be preserved or made more deterministic

o The bounds for enabling conditions of actions can be strengthened but not weakened

Superposition Morphisms

36

and, moreover, for every g in Γ_2 s.t. $\sigma_{ac}(g)$ is defined

- $R_2(g) \supseteq \underline{\sigma}(R_1(\sigma_{ac}(g)))$
- $L_2(g) \supseteq \underline{\sigma}(L_1(\sigma_{ac}(g)))$
- $U_2(g) \supseteq \underline{\sigma}(U_1(\sigma_{ac}(g)))$

o Effects of actions must be preserved or made more deterministic

o The bounds for enabling conditions of actions can be strengthened but not weakened

ICFEM 2005 University of Leicester

Superposition Morphisms: Examples


```
design n-account is
  out num:nat, bal:int
  in  v:nat
  do  dep[bal]: true → bal:=v+bal
  []  wit[bal]: bal≥v → bal:=bal-v
```

inclusion

```
design e-account[MAX:int] is
  out  num:nat, bal:int, count:int
  in   v,day:nat
  prv  d:int
  do   dep[bal,d,count]: true →
          bal:=v+bal
       ‖  d:=day
       ‖  if bal≥MAX then count:=count+(day-d)
  []   wit[bal,d,count]:
          bal≥v → bal:=bal-v
       ‖  d:=day
       ‖  if bal≥MAX then count:=count+(day-d)
  []    reset [d,count]:
          true, false → count:=0‖d:=day
```

Superposition Morphisms: Examples

Another example

```
design account is
   out num:nat, bal:int
   in  v: nat
   do  dep: true → bal:=v+bal
   []  wit: true → bal:=bal-v
```

inclusion

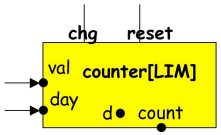
```
design n-account is
   out num:nat, bal:int
   in  v: nat
   do  dep: true → bal:=v+bal
   []  wit: bal≥v → bal:=bal-v
```


Externalising superposed behaviour

- These examples represent two typical kinds of superposition
  - monitoring
  - regulation
- The superposed behaviour can be captured by a component
  - monitor — Support reuse
  - regulator
- and the new design is obtained by interconnecting the underlying design with this component.

Externalising the counter

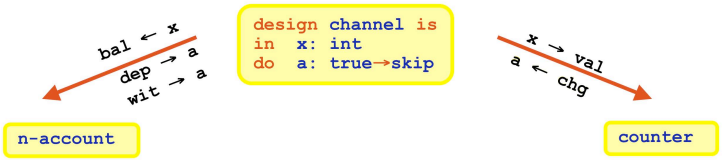
A design of a counter that counts how many days a value has exceed a given value, since the last time it was reset




```
design counter[LIM:int] is
   in  val,day:nat
   out count:int
   prv d:int
   do  chg[d,count]: true →
         d:=day
       ‖ if val≥LIM then count:=count+(day-d)
   []  reset[d,count]: true, false → count:=0‖d:=day
```


Externalising the counter

To identify which channels and actions of the account are involved in the monitoring by the counter, we use the diagram



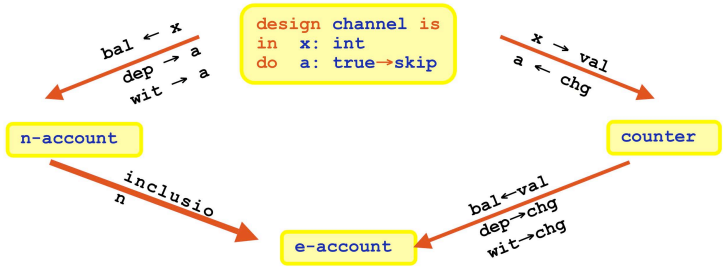
This diagram captures the configuration of a system with two components — n-account and counter — that are interconnected through a third design (a communication channel)

Configurations

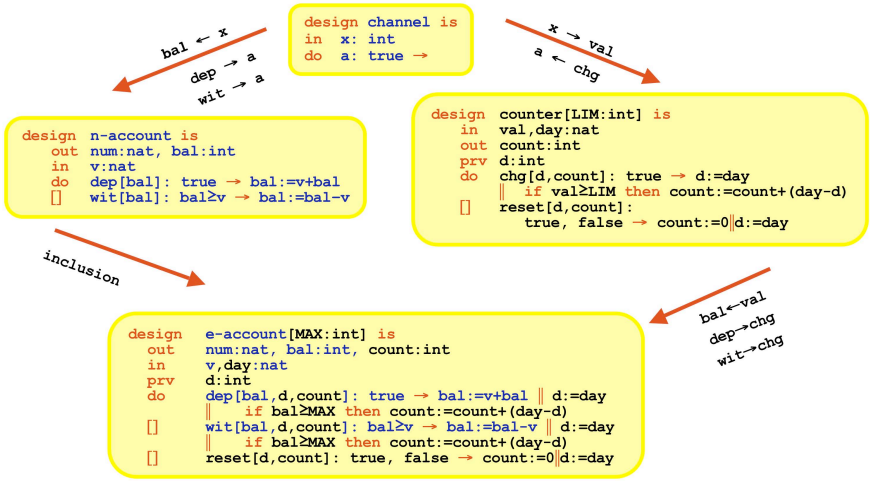
- Using diagrams whose nodes are labelled by designs and whose arcs are labelled by superposition morphisms, it is possible to design large systems from simpler components.
- Interactions between components are made explicit through the corresponding name bindings.
- Name bindings are represented as additional nodes labelled with designs and edges labelled by morphisms.

Semantics of Configurations

What's the relationship between **e-account** and the configuration?

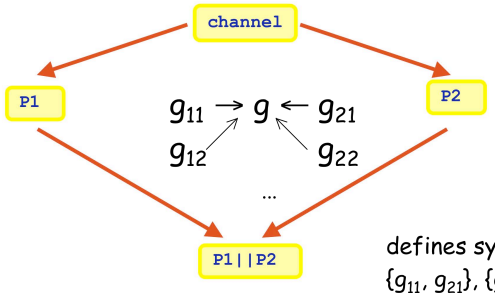


Semantics of Configurations



Semantics of Configurations

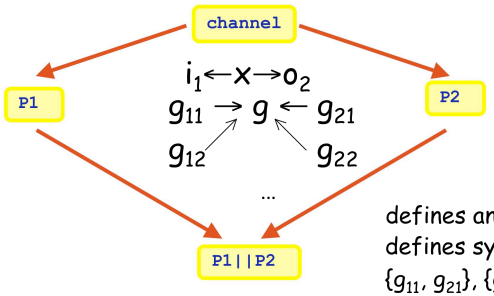
The semantics of configurations is given by the "amalgamated sum" (**colimit**) of the diagram.



defines synchronisation sets $\{g_{11}, g_{21}\}$, $\{g_{12}, g_{21}\}$,...

Semantics of Configurations

The semantics of configurations is given by the "amalgamated sum" (colimit) of the diagram.



defines an I/O connection
defines synchronisation sets
$\{g_{11}, g_{21}\}, \{g_{12}, g_{21}\},\ldots$

Semantics of Configurations

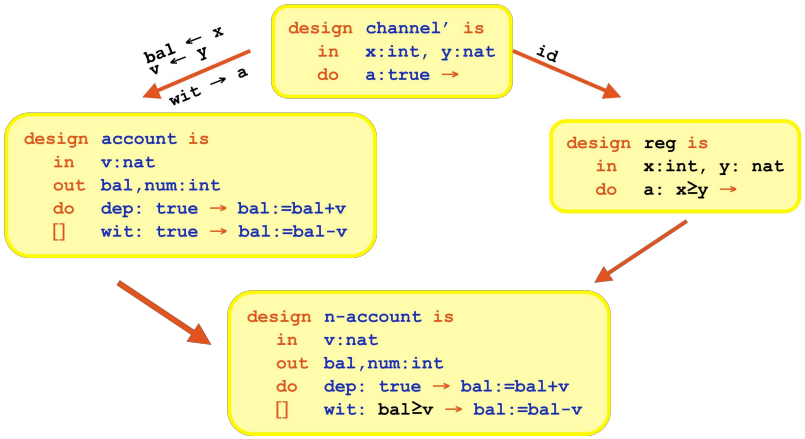
The colimit of such design diagrams

- Amalgamates channels involved in each i/o interconnection and the result is an output channel of the system design
- Represents every synchronisation set $\{g_1,g_2\}$ by a single action $g_1|g_2$ with
  - safety bound: conjunction of the safety bounds of g_1 and g_2
  - progress bound: conjunction of the progress bounds of g_1 and g_2
  - conditions on next state: conjunction of conditions of g_1 and g_2

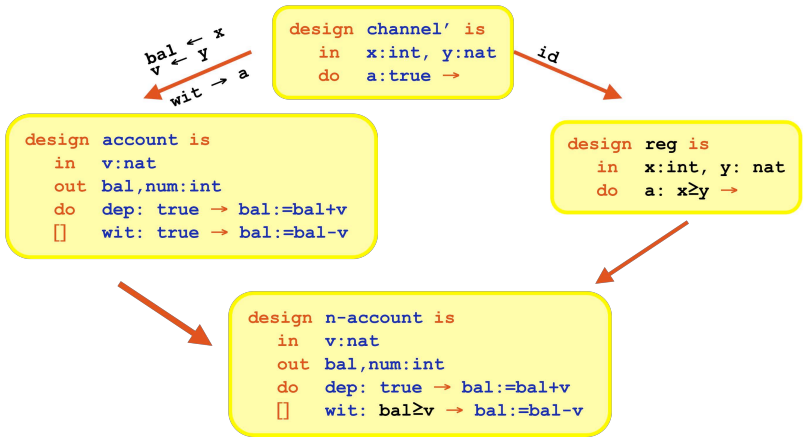
Configurations

- Not every diagram represents a meaningful configuration.
- Restrictions on diagrams that make them well-formed configurations:
  - An output channel of a component cannot be connected (directly or indirectly through input channels) with output channels of the same or other components.
  - Private channels and private actions cannot be involved in the connections.
- These restrictions cannot be captured by the notion of superposition because they involve the whole diagram.

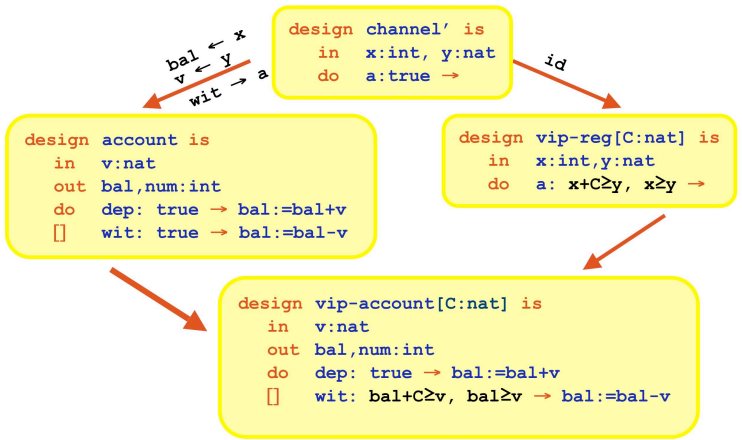
Externalising the regulator



Externalising the regulator



vip-account



Recall: architectural elements

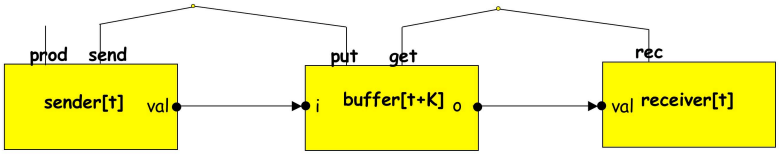
- **Components**
  - model entities/devices/machines (software or "real world"), that keep an internal state, perform computations, and are able to synchronise with their environment and exchange information through channels
  - "designs" given in terms of communication channels and actions
- **Connectors**
  - model entities whose purpose is to coordinate interactions between components
  - "structured designs" given in terms of a "glue" and collection of "roles" (as in Wright)
  - can be superposed at run-time over given components
- **Configurations**
  - diagrams in a category of designs as objects and superposition as morphisms;
  - composition (emergent behaviour) given by colimit construction

From simple to complex interactions

- The configuration diagrams presented so far express **simple** and **static** interactions between components
  - action synchronisation
  - the interconnection of input channels of a component with output channels of other components
- More complex interaction protocols can also be described by configurations...

Bounded asynchronous interaction

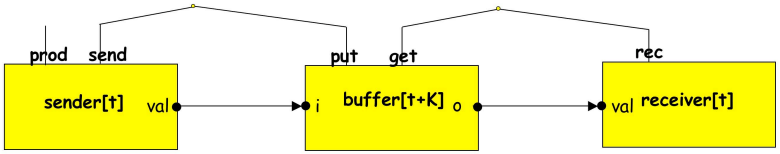
A generic sender and receiver communicating asynchronously, through a bounded buffer



Bounded asynchronous interaction

52

A generic sender and receiver communicating asynchronously, through a bounded buffer



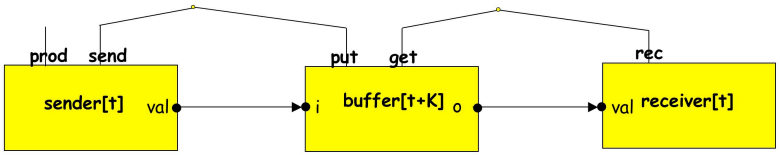

```
design sender[t] is
out   val:t
prv   rd:bool
do    prod[val,rd]:¬rd,false→rd'
[]    send[rd]:rd,false → ¬rd'
```

ICFEM 2005
University of Leicester

Bounded asynchronous interaction

52

A generic sender and receiver communicating asynchronously, through a bounded buffer

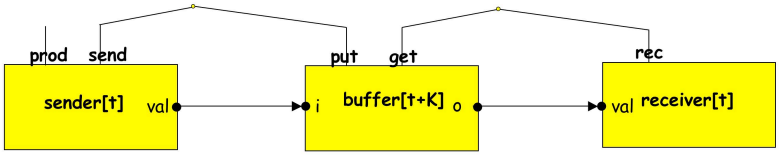



```
design sender[t] is
out   val:t
prv   rd:bool
do    prod[val,rd]:¬rd,false→rd'
[]    send[rd]:rd,false → ¬rd'
```

```
design receiver[t] is
in   val:t
do   rec:true,false→
```

ICFEM 2005
University of Leicester

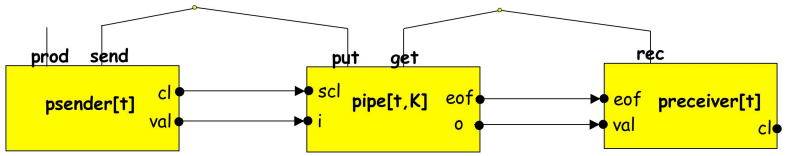
Bounded asynchronous interaction



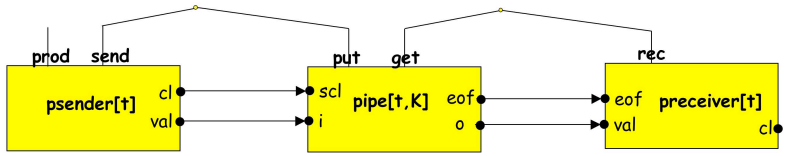

```
design buffer[t; K:nat] is
in    i:t
out   o:t
prv   q:queue(K,t); rd:bool
do    put:¬full(q)→q:=enqueue(i,q)
[]prv next:¬empty(q)∧¬rd →o:=head(q)‖q:=tail(q)‖rd:=true
[]    get:rd → rd:=false
```

ICFEM 2005
University of Leicester

Communicating through a pipe

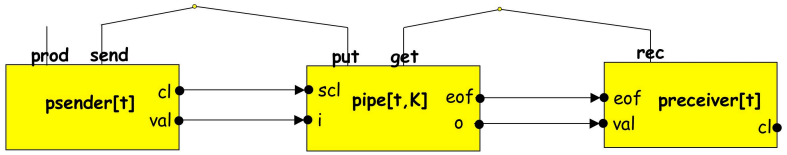


Communicating through a pipe




```
design psender[t] is
out   val:t, cl:bool
prv   rd:bool
do    prod[val,rd]:¬rd∧¬cl,false→rd'
[]prv close[cl]:¬rd∧¬cl,false→cl'
[]    send[rd]:rd,false→¬rd'
```

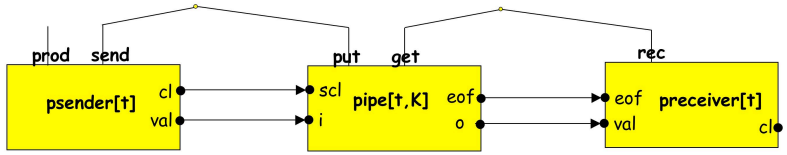

Communicating through a pipe




```
design psender[t] is
out   val:t, cl:bool
prv   rd:bool
do    prod[val,rd]:¬rd∧¬cl,false→rd'
[]prv close[cl]:¬rd∧¬cl,false→cl'
[]    send[rd]:rd,false→¬rd'
```

```
design preceiver[t] is
in    val:t, eof:bool
out   cl:bool
do    rec:¬eof∧¬cl,false→
[]prv close:¬cl,¬cl∧eof→cl'
```

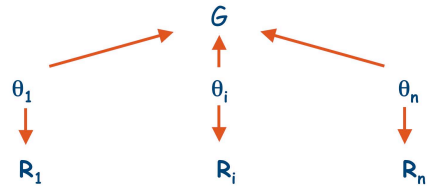

Communicating through a pipe




```
design pipe[t,K:nat] is
in    i:t, scl:bool
out   o:t, eof:bool
prv   q:queue(K,t);rd:bool
do    put: ¬full(q)→q:=enqueue(i,q)
[]prv next: ¬empty(q)∧¬rd→o:=head(q)‖q:=tail(q)‖rd:=true
[]    get: rd→rd:=false
[]prv signal: scl∧empty(q)∧¬rd→eof:=true
```

Connectors

- A **connector** is a well-formed configuration of the form



G is the glue and **R**'s are the roles

- Its **semantics** is given by the colimit of the diagram

Refinement

Connectors can be **applied** (instantiated) to components that refine (are instances of) their roles

A **refinement** mapping

$$\sigma: P_1 \rightarrow P_2$$

supports the identification of a way in which the design P_1 is refined by P_2.

Refinement morphisms

A refinement morphism $\sigma: P_1 \rightarrow P_2$ consists of

- a total function $\sigma_{ch}: V_1 \rightarrow \mathbf{Term}(V_2)$ s.t.

- a partial mapping $\sigma_{ac}: \Gamma_2 \rightarrow \Gamma_1$ s.t.

Refinement morphisms

A refinement morphism $\sigma: P_1 \rightarrow P_2$ consists of

- a total function $\sigma_{ch}: V_1 \rightarrow \mathbf{Term}(V_2)$ s.t.

Sorts are preserved as well as the border between the component and its environment

- a partial mapping $\sigma_{ac}: \Gamma_2 \rightarrow \Gamma_1$ s.t.

Domains of channels are preserved
Every action that models interaction has to be implemented

Refinement morphisms

A refinement morphism $\sigma: P_1 \to P_2$ consists of

- a total function $\sigma_{ch}: V_1 \to Term(V_2)$ s.t.
  - $sort_2(\sigma_{ch}(v)) = sort_1(v)$
  - $\sigma_{ch}(out(V_1)) \subseteq out(V_2)$
  - $\sigma_{ch}(in(V_1)) \subseteq in(V_2)$
  - $\sigma_{ch}(prv(V_1)) \subseteq Term(loc(V_2))$

Sorts are preserved as well as the border between the component and its environment

- a partial mapping $\sigma_{ac}: \Gamma_2 \to \Gamma_1$ s.t.
  - $\sigma_{ac}(sh(\Gamma_2)) \subseteq sh(\Gamma_1)$
  - $\sigma_{ac}(prv(\Gamma_2)) \subseteq prv(\Gamma_1)$
  - $\sigma_{ac}^{-1}(g) \neq \emptyset,\ g \in sh(\Gamma_1)$
  - $\sigma_{ch}(D_1(\sigma_{ac}(g))) \subseteq \underline{D}_2(g)$
  - $\sigma_{ac}(\underline{D}_2(\sigma_{var}(v))) \subseteq D_1(v),\ v \in loc(V_1)$

Domains of channels are preserved
Every action that models interaction has to be implemented

Refinement morphisms

and, moreover, for every g in Γ_2 s.t. $\sigma_{ac}(g)$ is defined

and for every g_1 in Γ_1

Effects of actions must be preserved or made more deterministic.

The interval defined by the safety and progress bounds of each action must be preserved or reduced

Refinement morphisms

and, moreover, for every g in Γ_2 s.t. $\sigma_{ac}(g)$ is defined

- $R_2(g) \supseteq \underline{\sigma}(R_1(\sigma_{ac}(g)))$
- $L_2(g) \supseteq \underline{\sigma}(L_1(\sigma_{ac}(g)))$

and for every g_1 in Γ_1

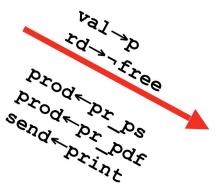
- $\underline{\sigma}(U_1(g_1)) \supseteq \bigvee_{\{g2:\underline{\sigma}(g2)=g1\}} U_2(g_2)$

Effects of actions must be preserved or made more deterministic.

The interval defined by the safety and progress bounds of each action must be preserved or reduced

worduser - a refinement of sender

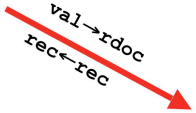

```
design sender(ps+pdf) is
out val:ps+pdf
prv rd:bool
do  prod[val,rd]:¬rd,false→rd'
[]  send[rd]:rd,false → ¬rd'
```




```
design user is
out p:ps+pdf
prv free:bool, w:MSWord
do  save[w]: true,false →
[]  pr_ps[p,free]: free → p:=ps(w)‖free:=false
[]  pr_pdf[p,free]: free → p:=pdf(w)‖free:=false
[]  print[free]: ¬free → free:=true
```

printer: a refinement of receiver


```
design receiver(ps+pdf) is
in  val:ps+pdf
do  rec[]:true,false→
```




```
design printer is
out rdoc:ps+pdf
prv busy:bool, pdoc:ps+pdf
do  rec:¬busy→pdoc:=rdoc||busy:=true
[]  end_print:busy,false→busy:= false
```


Structuring systems vs Refinement

It is essential that

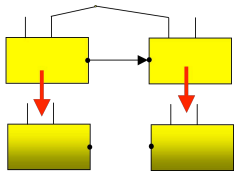
the gross modularisation of a system
in terms of
components and their interconnections

be "respected" when component designs are refined into more concrete ones

Compositionality

Structuring systems vs Refinement

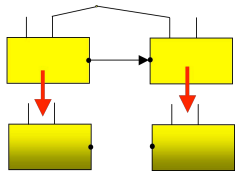
If the descriptions of the components of a system are refined into more concrete ones



Structuring systems vs Refinement

63

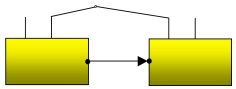
If the descriptions of the components of a system are refined into more concrete ones



1. It is possible to propagate the interactions previously defined

Structuring systems vs Refinement

If the descriptions of the components of a system are refined into more concrete ones

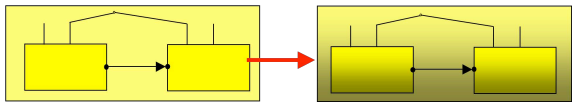


1. It is possible to propagate the interactions previously defined

Structuring systems vs Refinement

63

If the descriptions of the components of a system are refined into more concrete ones



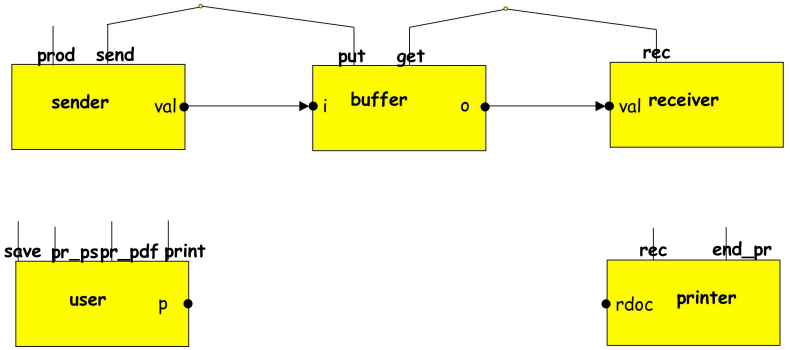
1. It is possible to propagate the interactions previously defined

2. The resulting description of the system refines the previous one

ICFEM 2005
University of Leicester

Connector instantiation

Example

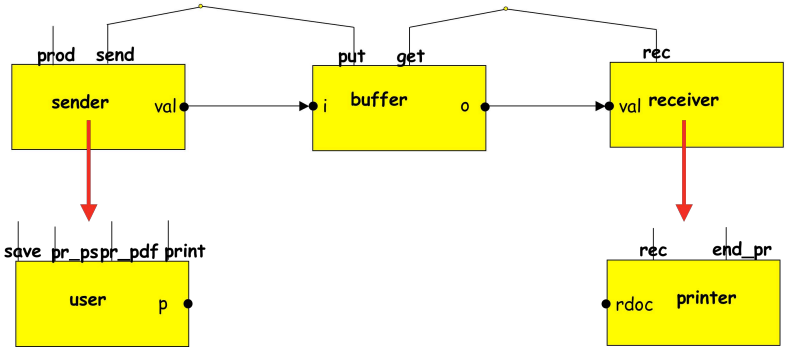


ICFEM 2005
University of Leicester

Connector instantiation

64

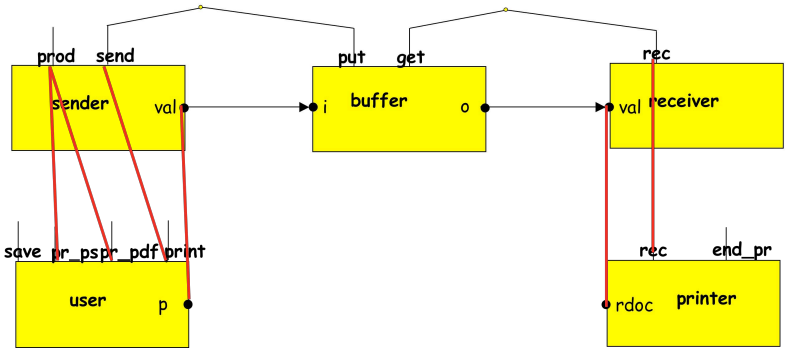
Example



ICFEM 2005
University of Leicester

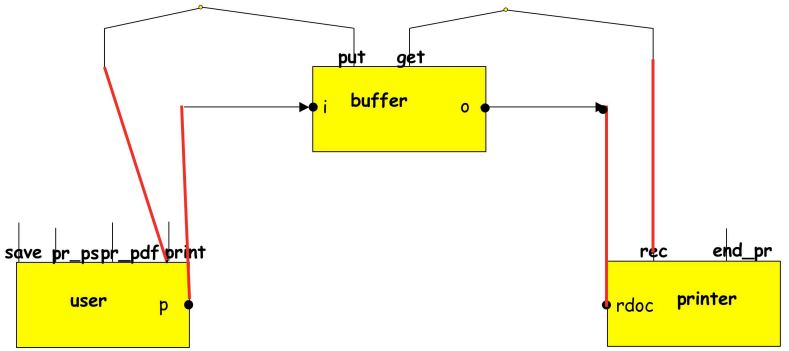
Connector instantiation

Example

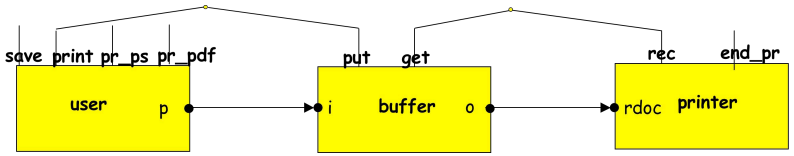


Propagation of the interconnections

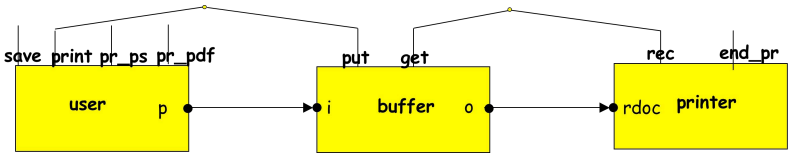
Example



Compositionality

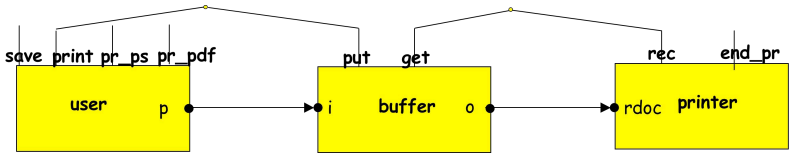


Compositionality



Compositionality ensures that properties inferred from the more abstract description hold also for the more concrete (refined) one

Compositionality

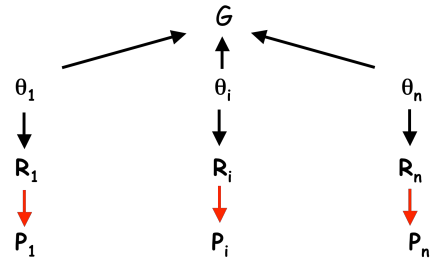


Compositionality ensures that properties inferred from the more abstract description hold also for the more concrete (refined) one

Eg: **in order message delivery** does not depend on the speed at which messages are produced and consumed

Connectors - Instantiation

- An **instantiation** of a connector consists of, for each of its roles R, a design P together with a refinement morphism ϕ:R→P

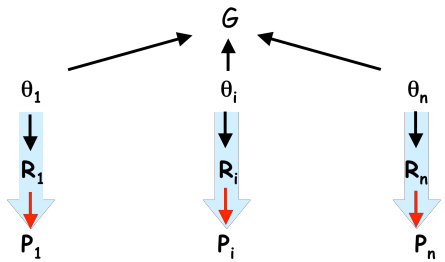


The **semantics** of a connector instantiation is the colimit of the diagram

Connectors - Instantiation

67

- An **instantiation** of a connector consists of, for each of its roles R, a design P together with a refinement morphism ϕ:R→P



The **semantics** of a connector instantiation is the colimit of the diagram

ICFEM 2005
University of Leicester

Systematising Configurations

We have seen that

- Complex interaction protocols can be described by configurations, independently of the concrete components they will be applied to; they can be used in different contexts
- The use of such interaction protocols in a given configuration corresponds to defining the way in which the generic participating components are refined by the concrete components

ICFEM 2005
University of Leicester

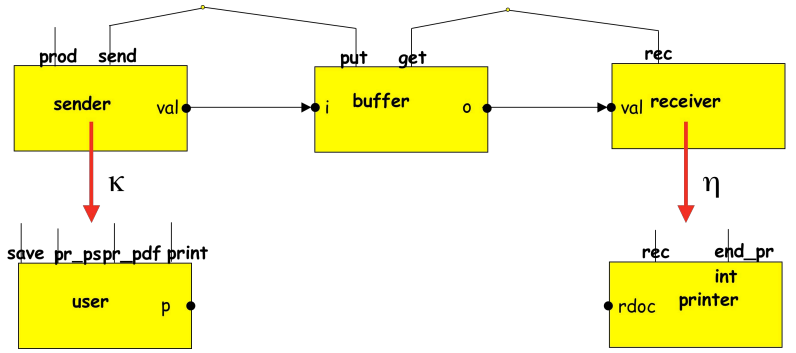
Systematising Configurations

We have seen that

- Complex interaction protocols can be described by configurations, independently of the concrete components they will be applied to; they can be used in different contexts **Connector Types**
- The use of such interaction protocols in a given configuration corresponds to defining the way in which the generic participating components are refined by the concrete components

Instantiation of Connectors

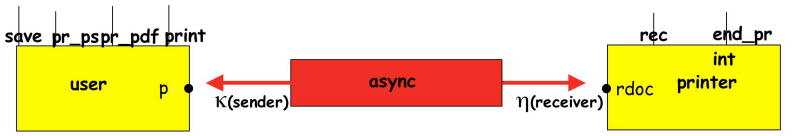
Systematising Configurations



Systematising Configurations

We may elevate the abstractions used to describe configurations...

Systematising Configurations



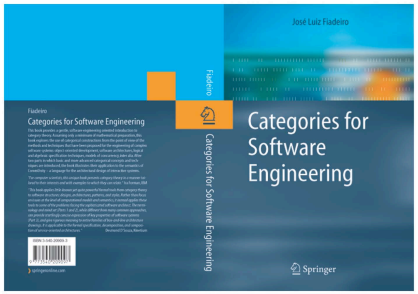
... and define them in terms of computational components and connectors

Concluding remarks

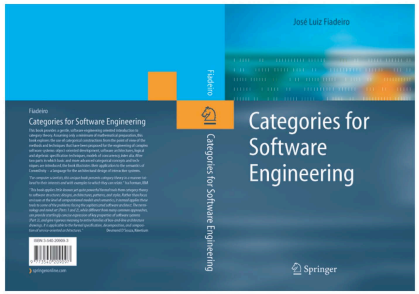
The age of "interactions":

- A truly ***great challenge!***
- Requires "new" Engineering methods and techniques
  - Interactions as first-class entities
  - Interaction-centric architectures
- Requires "new" Scientific basis
  - General Systems Theory
  - A good chance for more work in **TAC**...

Learn more ...



Learn more ...



www.fiadeiro.org/jose/CommUnity