Hierarchical Modeling for Computational Biology

Carsten Maus, Mathias John, Mathias Röhl, *Adelinde Uhrmacher*

University of Rostock

June 3, 2008

Agenda

I Introduction and Context
II Modular-hierarchical modeling with *DEVS
III π calculus
IV Components
V Summary

Hierarchies

The word “hierarchy” derives from the Greek (hierarches) ”high-priest” and (hieros), ”sacred” + (arkho), ”to lead, to rule”

The Assumption of the Virgin by Francesco Botticini, National Gallery London

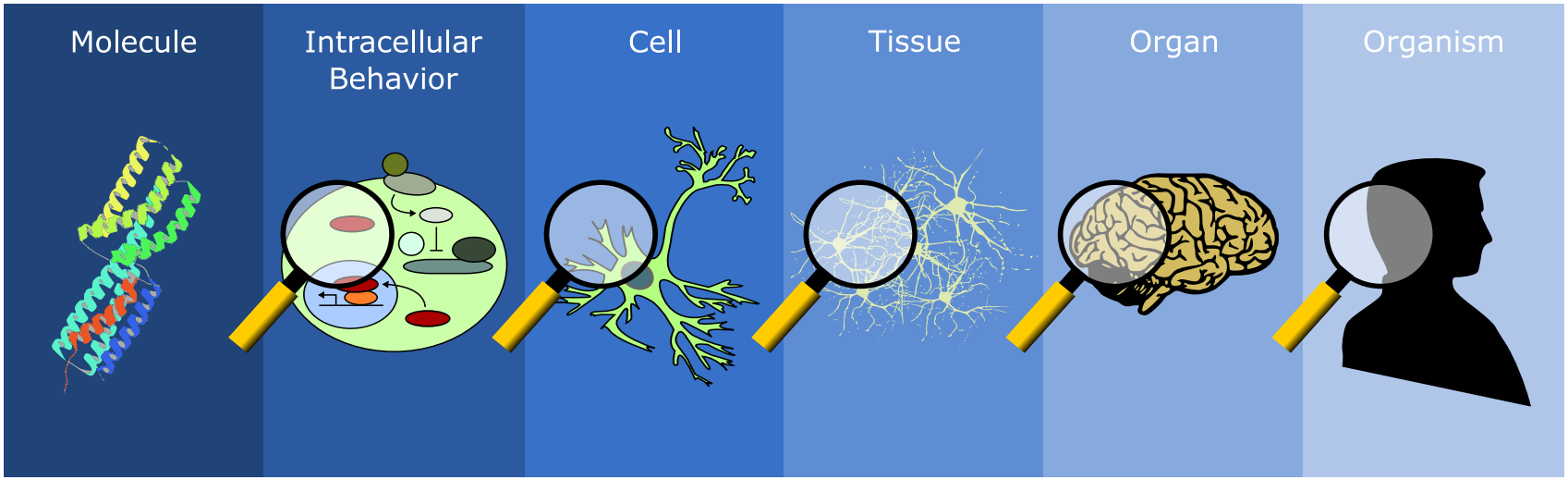
Hierarchies

A hierarchy is an arrangement of objects, people, elements, values, grades, orders, classes, etc., in a ranked or graduated series.
Hierarchies

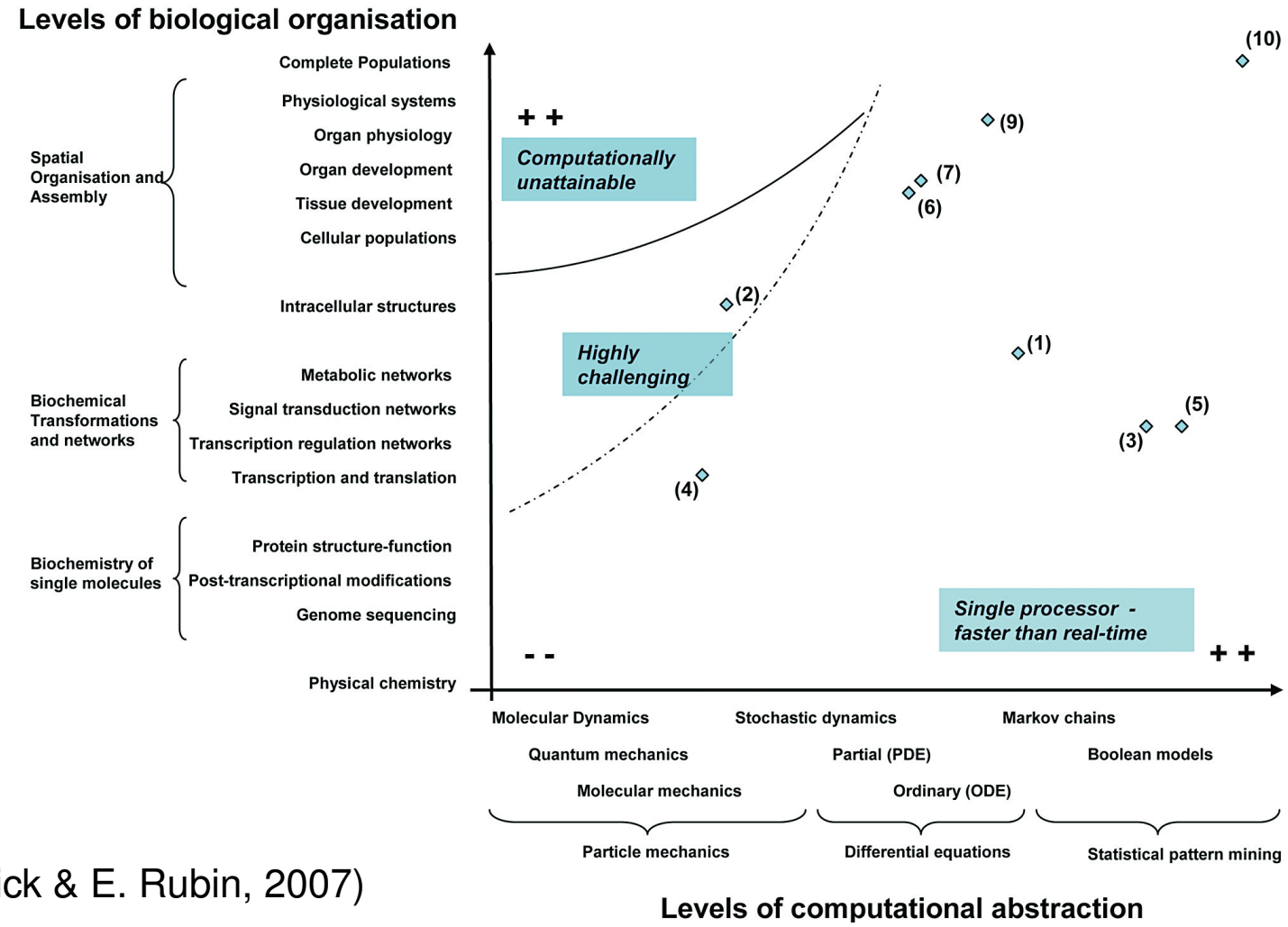
- are ubiquitous
- cognitive means
- separating important from less important elements
- ranking elements
- reduce level of detail

Hierarchies in Biology

“behavior at any level is explained in terms of the level below, and its significance is found in the level above” (Webster 1979)



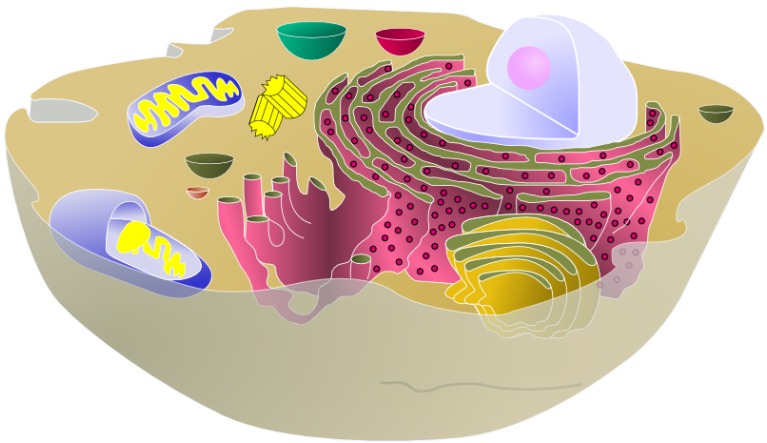
Biological vs. Computational Hierarchies

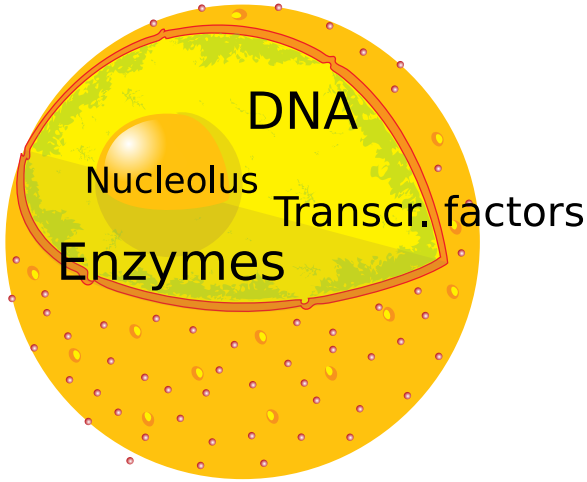


(G. Broderick & E. Rubin, 2007)

The Cell – A Hierarchical Perspective

- Components can be structured into classes of similar kinds, e.g. golgi, ER, and nucleus form organelles, i.e. membrane-bound compartments of the cell, → categorization hierarchy.
- The cell is *composed of* cytoplasm and several organelles → composition hierarchies.
- A closer look into the nucleus reveals additional distinct structures and components which might play a role depending on the objective of the simulation study → abstraction hierarchy.





(modified from Wikipedia)

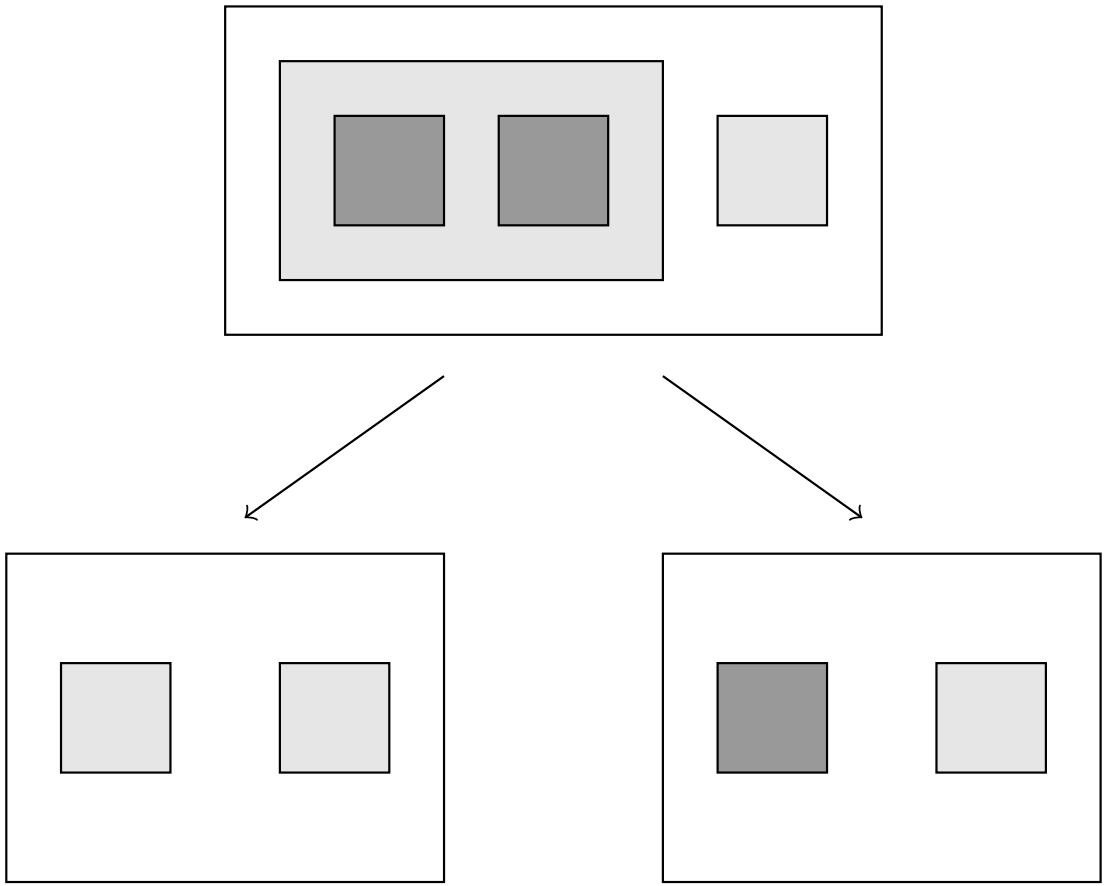
Which hierarchy are we interested in?

Hierarchies include

- categorization: is-a-relation (“The objective criterion for being in the same category is having common properties. But there is no objectivist criterion for which properties are to count.” (George Lakoff))
- abstraction: is-more-abstract-than, is-more-detailed-than (which might imply substituting one model component by a more abstract or refined one, or to combine different “abstract ones” in a model)
- composition: is-part-of (composition hierarchies are the sine qua non of hierarchical modeling, and handling complex systems)

In the following we will focus on the latter.

Compositional and abstraction hierarchy



compos. level $=$ abstract. level

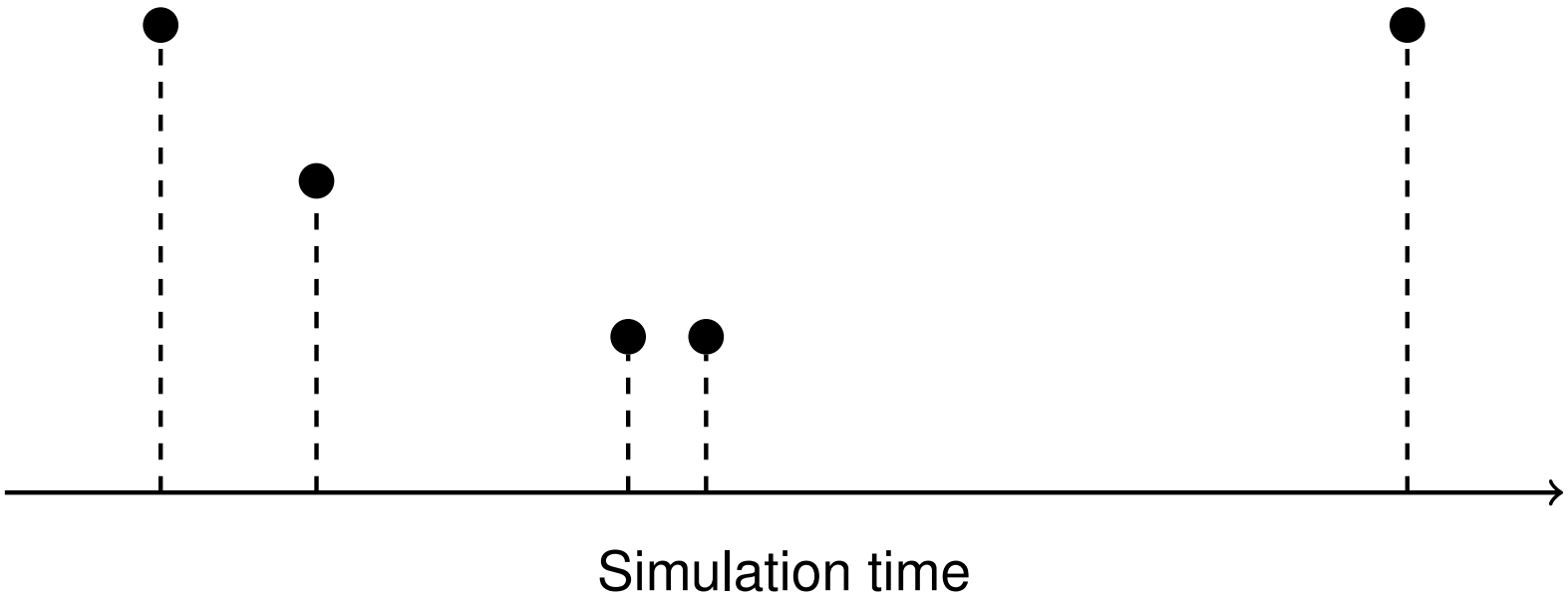
compos. level $\neq$ abstract. level

Part II

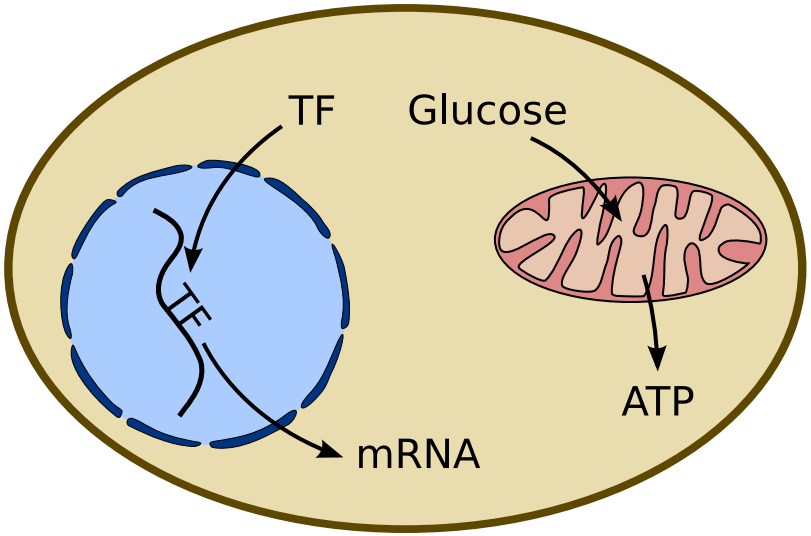
DEVS -Discrete Event Systems Specification

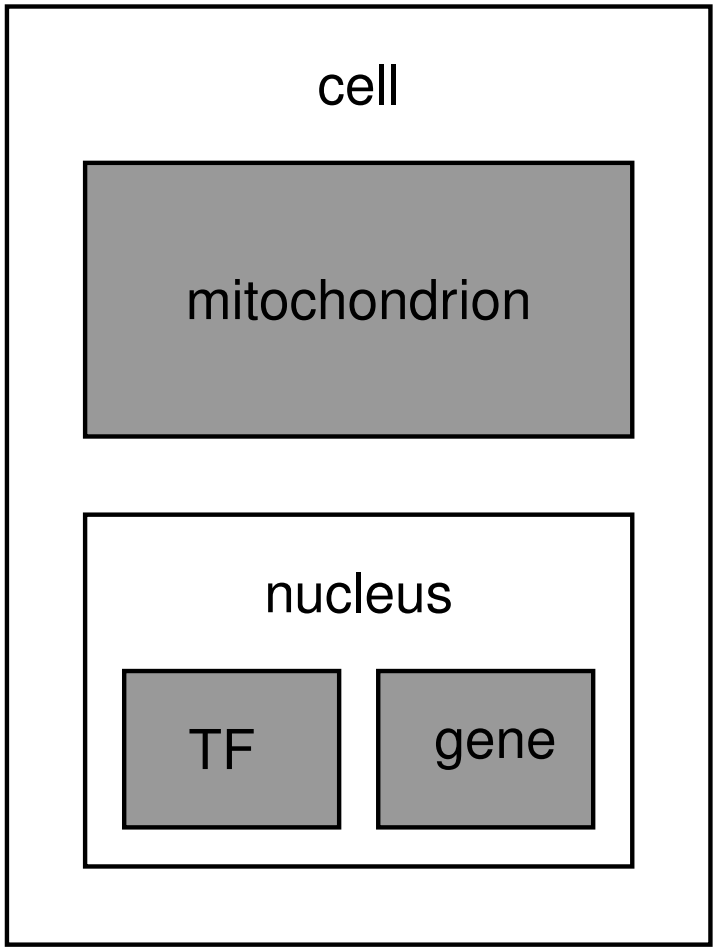
Discrete Event Systems Specification (DEVS)

- Developed by Zeigler in the 70s
- System theoretic roots
- Continuous time base
- Events at discrete time points
- Designed as a formalism for simulation (abstract simulator)



DEVS and compositional modeling





Buttom up: atomic P-DEVS model

atomic P-DEVS

$$\langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda \rangle$$

X	structured set of inputs
Y	structured set of outputs
S	structured set of states
$ta : S \rightarrow \mathcal{R}^{\geq 0} \cup \{\infty\}$	time advance function
$\delta_{ext} : Q \times X^b \rightarrow S$	external state transition function, with
$Q = \{(s, e) : s \in S, 0 \leq e < ta(s)\}$	state set incl. elapsed time
$\delta_{int} : S \rightarrow S$	internal state transition function
$\delta_{con} : S \times X^b \rightarrow S$	confluent transition function
$\lambda : S \rightarrow Y$	output function

Container: coupled P-DEVS model

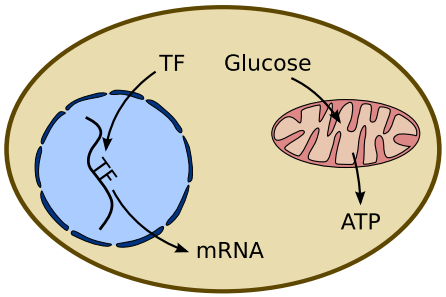
coupled P-DEVS

$$\langle X, Y, D, M_i, I_i, Z_{i,j} \rangle$$

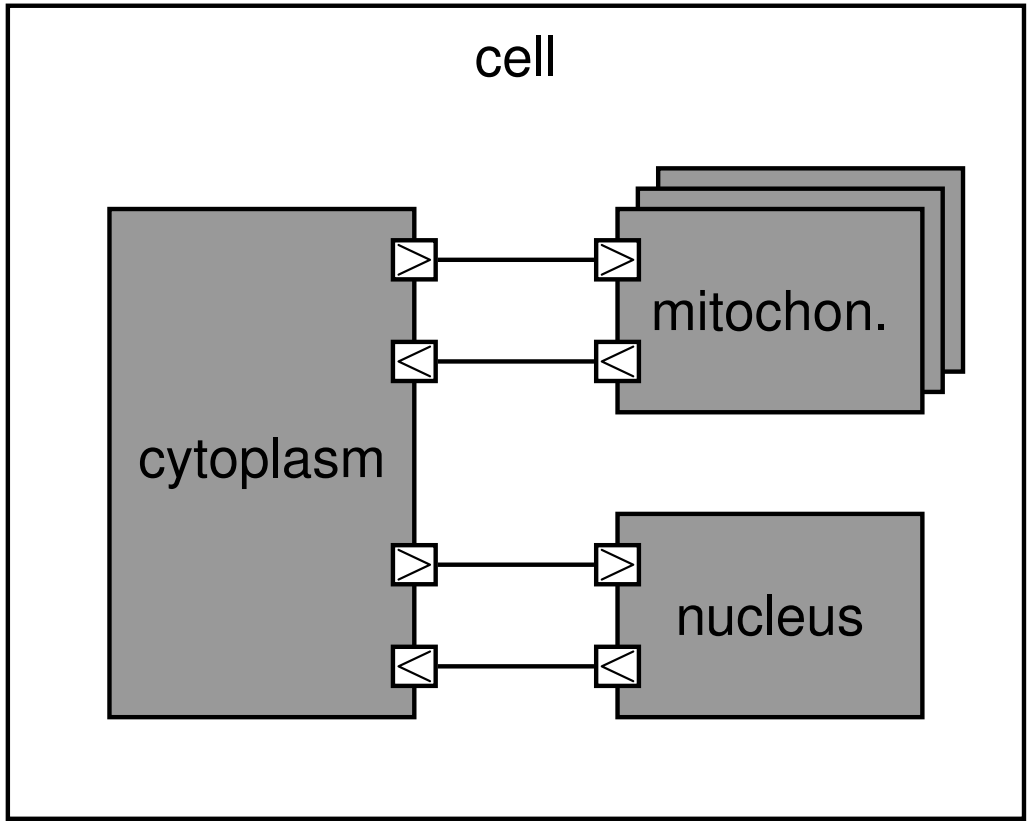
X	structured set of inputs
Y	structured set of outputs
D	name set of components
M_i	structured set of components
I_i	set of influencers of each component
$Z_{i,j}$	input output translation function

The result: modular, composition of models based on their interfaces (input and output sets and the defined couplings).

The example cell model described with P-DEVS



- diff. compartments as atomic models
- molecules on population level within atomic models
- cytoplasm, nucleus, and mitochondria on same composition level
- however at which abstraction level are they defined?

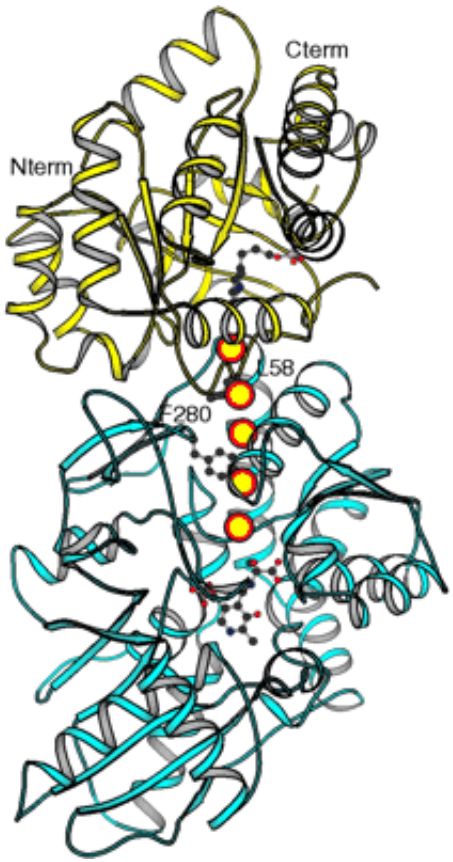


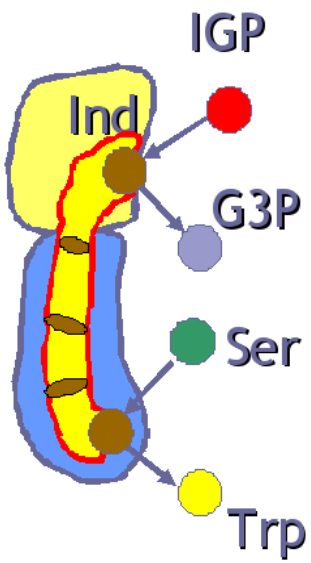
En-detail: the mitochondrion

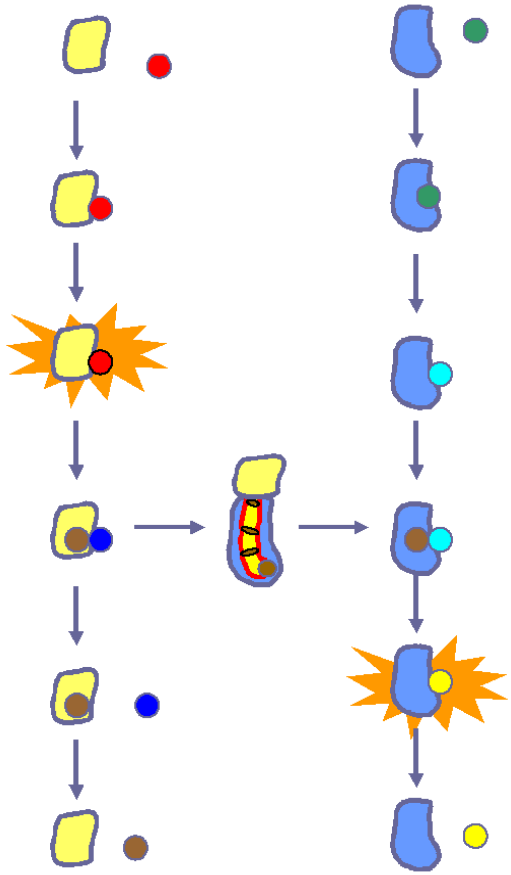
```
1   X = {glucosIn}
2   Y = {atpOut}
3   S = { (phase, #glucose, timeToNextATP) |
4           phase ∈ (idle, working),
5           #glucose ∈ N,
6           timeToNextATP ∈ R+ }
7   δext = #glucose++;
8        timeToNextATP = metabolizeDuration(#glucose);
9        phase = working
10  δint = if (#glucose > 0) then
11          #glucose--
12          timeToNextATP = metabolizeDuration(#glucose);
13          phase = working
14       else
15          phase = idle
16  δcon = δint ; δext
17  λ = atpOut("ATP")
18  ta = case phase of
19          idle: ∞
20          working: timeToNextATP
21       end case
```

Another example

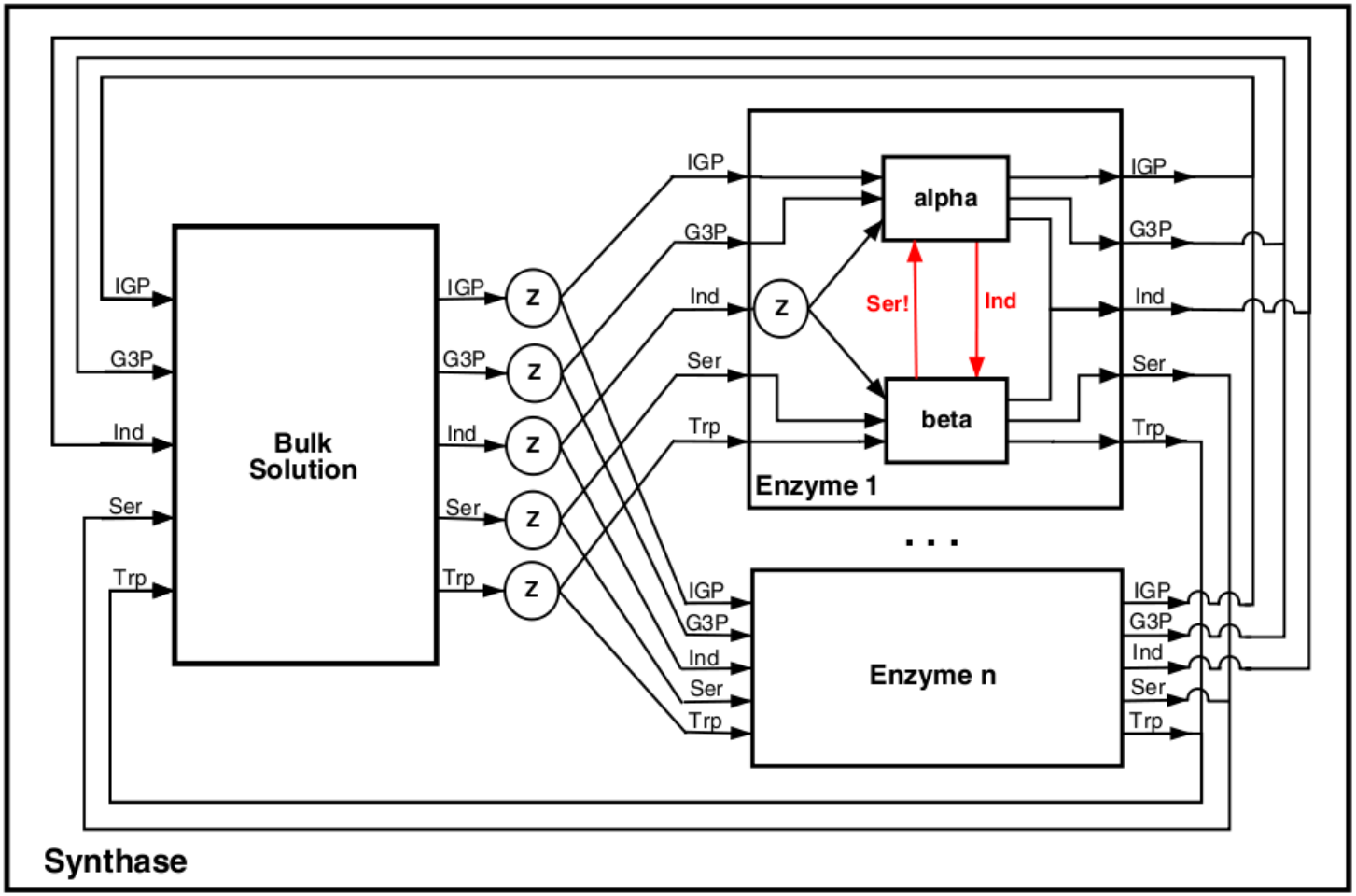
Channeling within enzyme complexes – Tryptophan synthase







Tryptophan synthase model in DEVS



Modeling biological systems with DEVS

Mapping of DEVS to biology

- seperation of individual submodels $\rightarrow$ compartments separated by membranes
- input and output ports $\rightarrow$ receptors, transport proteins or semi-permeability
- similar to StateCharts – reactive systems perspective
- modular composition of models
- different abstractions by different scaled variables, and degrees of composition

Modeling biological systems with DEVS

Mapping of DEVS to biology

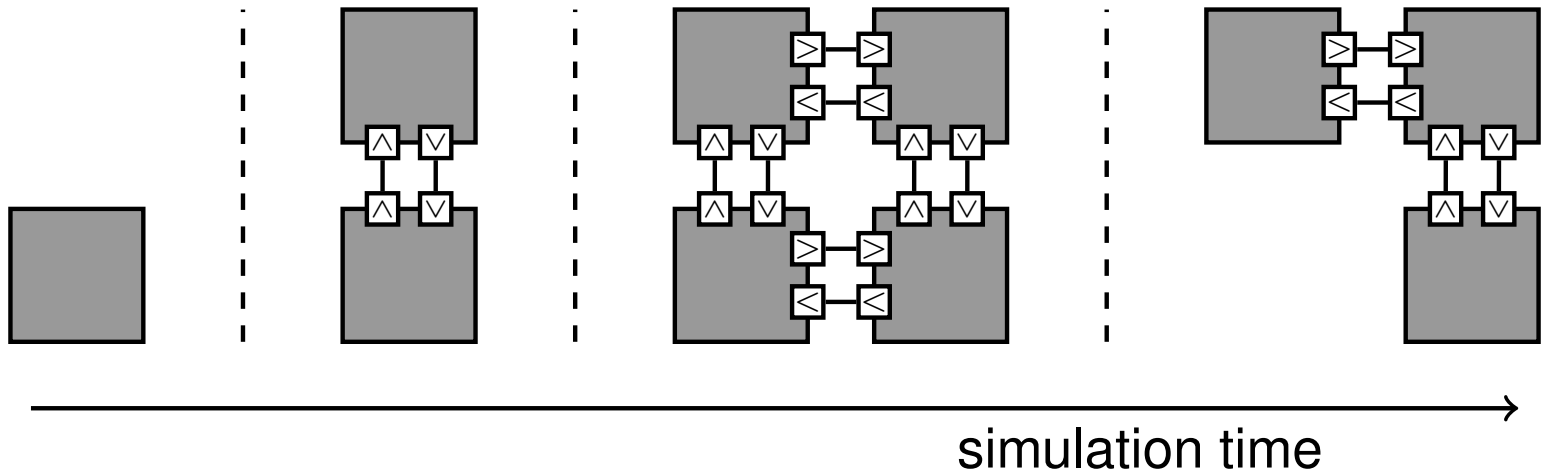
- seperation of individual submodels $\rightarrow$ compartments separated by membranes
- input and output ports $\rightarrow$ receptors, transport proteins or semi-permeability
- similar to StateCharts – reactive systems perspective
- modular composition of models
- different abstractions by different scaled variables, and degrees of composition

However, emphasis on static composition

Model structure (hierarchies, components and couplings) is static, no explicit means for reflecting different levels of abstraction.

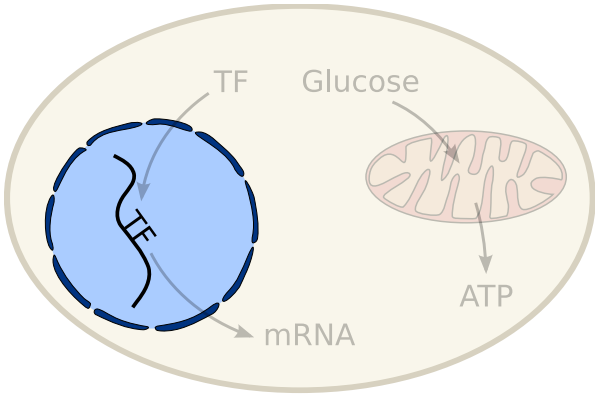
Dynamically structured models

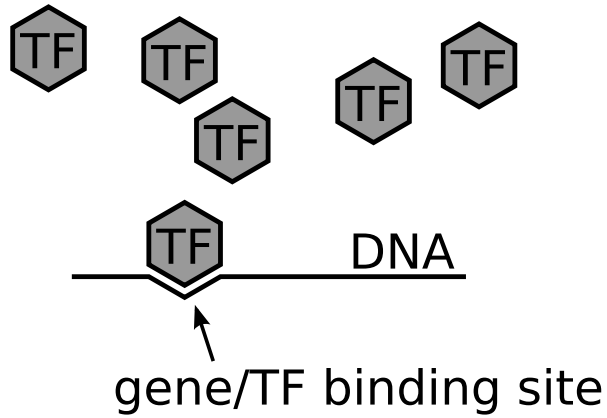
- Biology shows very often varying interaction partners.
- Model structure changes during simulation.
- Extensions of DEVS formalism needed (e.g. DynDEVS, ρ-DEVS).



Evolution of DEVS

To support modeling biological phenomena, e.g. individual-based binding of TF to DNA/gene within the nucleus



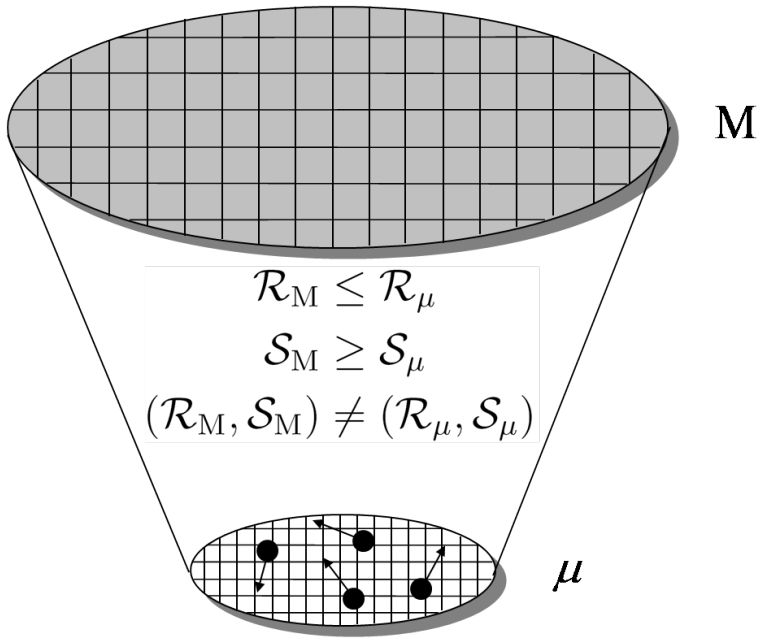


- DynDEVS supports variable composition and interaction
- ρ-DEVS, in addition variable ports and coupling functions
- ml-DEVS in addition, modeling at multiple abstraction levels, i.e. micro and macro

Macro, Micro, Megrim

micro (μ) and macro (M) models are different w.r.t.

- scope refers to elements within the systems's boundary,
- resolution refers to the smallest possible distinction in space and time.

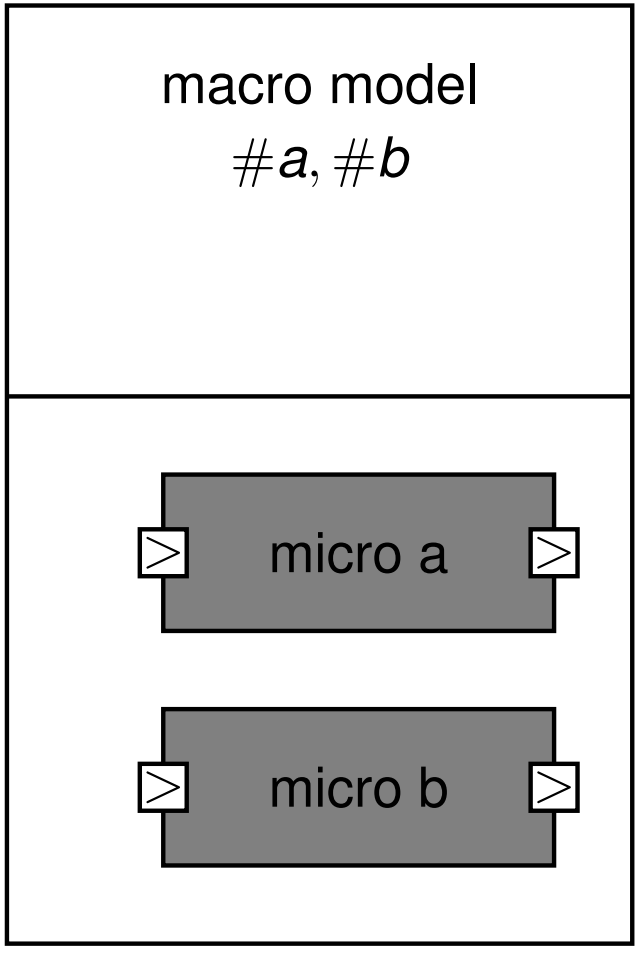


(Alex J. Ryan, 2007)

In biology the macro level refers typically to population and micro to individuals, macro and micro level are tied by upward and downward causation.

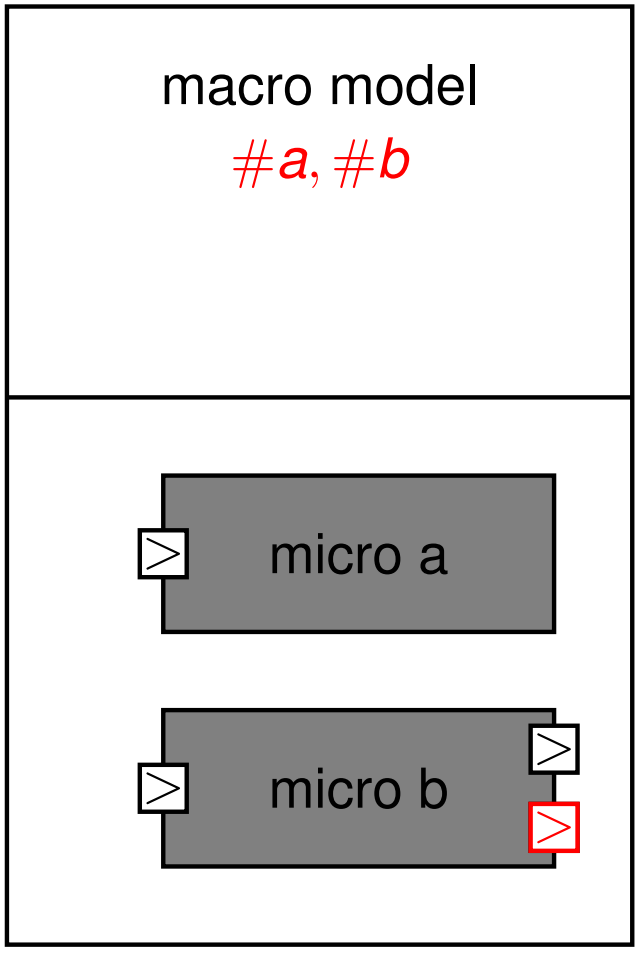
Multi-level modeling in ml-DEVS

- Coupled model with state and behavior
- Hierarchies of different abstraction levels



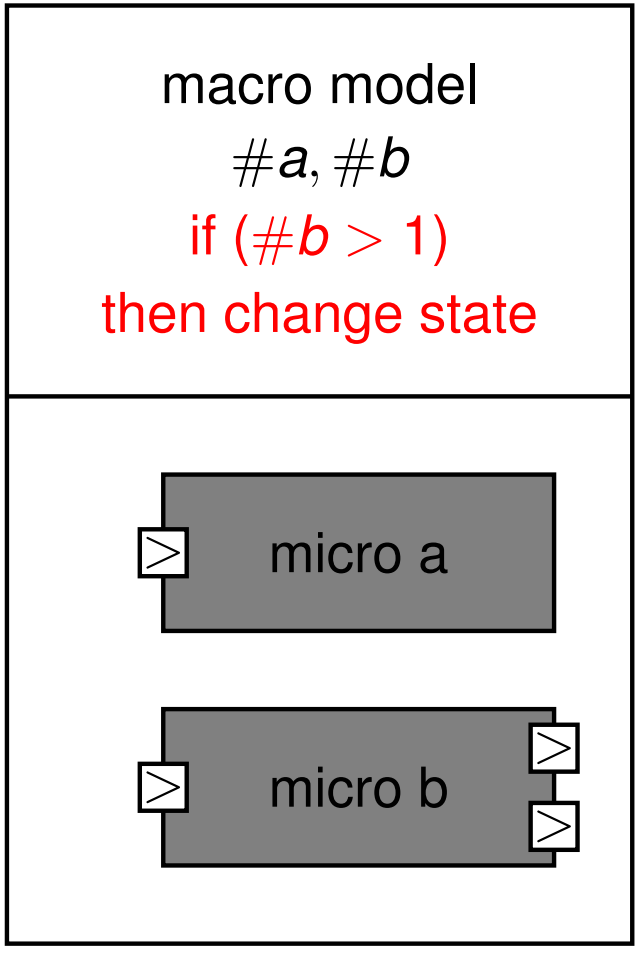
Multi-level modeling in ml-DEVS

- Coupled model with state and behavior
- Hierarchies of different abstraction levels
- Upward causation
  - Port changes (information)



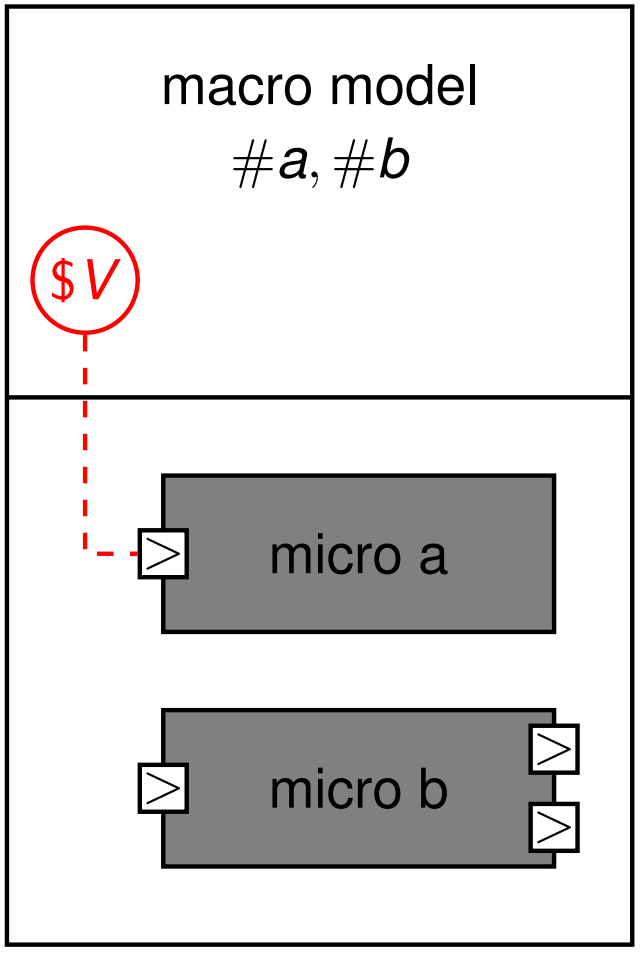
Multi-level modeling in ml-DEVS

- Coupled model with state and behavior
- Hierarchies of different abstraction levels
- Upward causation
  - Port changes (information)
  - Macro level invariant (activation)



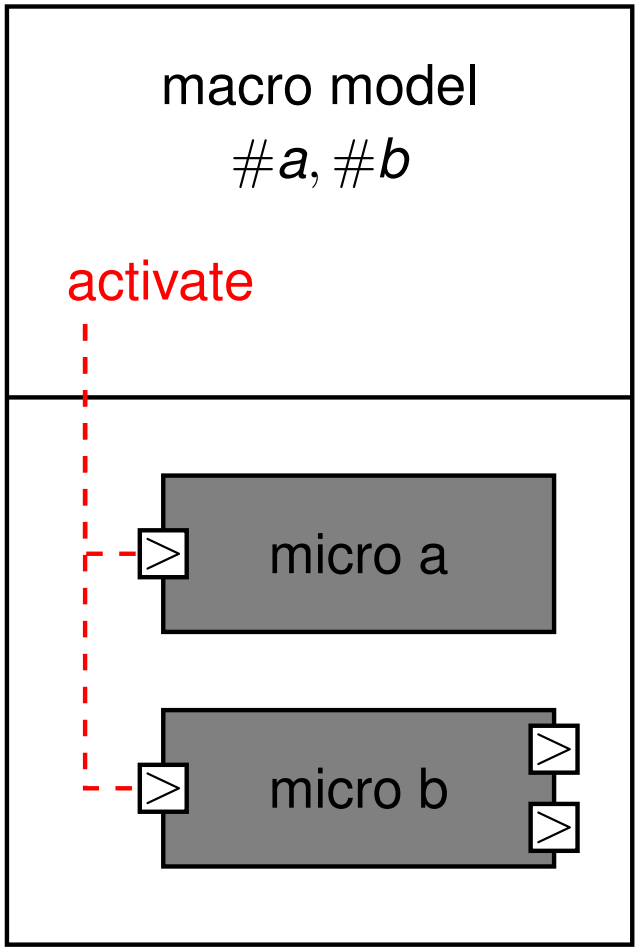
Multi-level modeling in ml-DEVS

- Coupled model with state and behavior
- Hierarchies of different abstraction levels
- Upward causation
  - Port changes (information)
  - Macro level invariant (activation)
- Downward causation
  - Value couplings (information)



Multi-level modeling in ml-DEVS

- Coupled model with state and behavior
- Hierarchies of different abstraction levels
- Upward causation
  - Port changes (information)
  - Macro level invariant (activation)
- Downward causation
  - Value couplings (information)
  - Direct sending of events (activation)



Definition: atomic ml-DEVS model

atomic ml-DEVS

$$\langle X, Y, S, s_{init}, p, \delta, \lambda, ta \rangle$$

X	structured set of inputs
Y	structured set of outputs
S	structured set of states
$s_{init} \in S$	start state
$p : S \to 2^{\mathcal{P}}$	port selection function
$\delta : X \times Q \to S$	state transition function
$\lambda : S \to Y$	output function
$ta : S \to \mathcal{R}^{\geq 0} \cup \{\infty\}$	time advance function

Definition: coupled ml-DEVS model

coupled ml-DEVS

$$\langle X, Y, S, s_{init}, p, C, MC, \delta, \lambda_{down}, v_{down}, sc, act, \lambda, ta \rangle$$

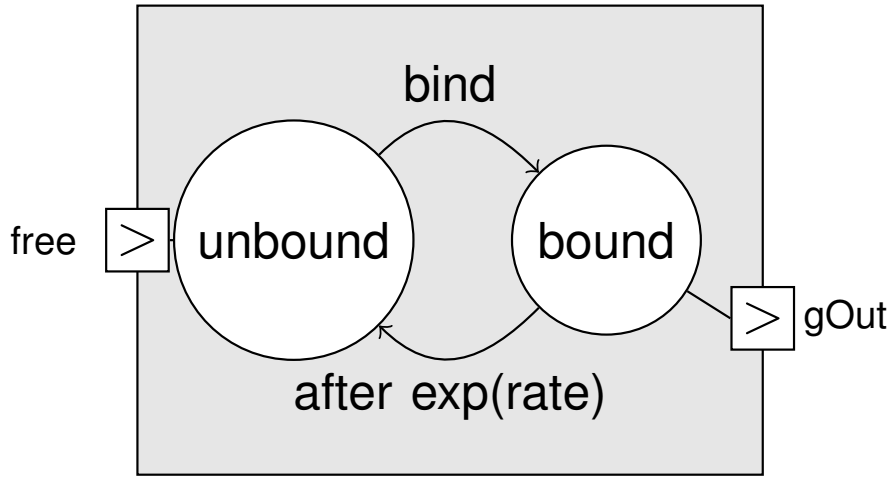
C	set of sub-models
MC	set of multi-couplings, $\{m \mid m : 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}\}$
$\delta : X \times Q \times 2^{C \times \mathcal{P}} \rightarrow S$	state transition function
$\lambda_{down} : S \rightarrow 2^{\cup_{c \in C}(X_C \times C \times \mathcal{P})}$	downward output function
$v_{down} : V_S \rightarrow \mathcal{P}$	value coupling downward
$sc : S \rightarrow 2^C \times 2^{MC}$	structural change function
$act_{up} : S \times 2^{C \times \mathcal{P}} \rightarrow \{true, false\}$	upward activation function

Coupled ml-DEVS model of the nucleus (macro)

```
1   X = { incomingTF }
2
3   Y = { producedMRNA }
4
5   S = { (timeToNextBind) | timeToNextBind ∈ R+ }
6
7   C = {TF_1...TF_n, gene}
8
9   MC = { (gene, producedMRNA, this, producedMRNA),
10         (TF, geneDock, gene, bindingsite) }
11
12  δ = if (incomingTF) then
13        addModel(TF);
14      else
15        #TF = count(TF, port free);
16        timeToNextBind = toTime(#TF × rateconst);
17
18  λ_down = if (gene port free) then
19             activate("bind", pick(TF, port free), free);
20             activate("active", gene, free);
21
22  ta = timeToNextBind
```

Atomic ml-DEVS model of transcription factor (micro)

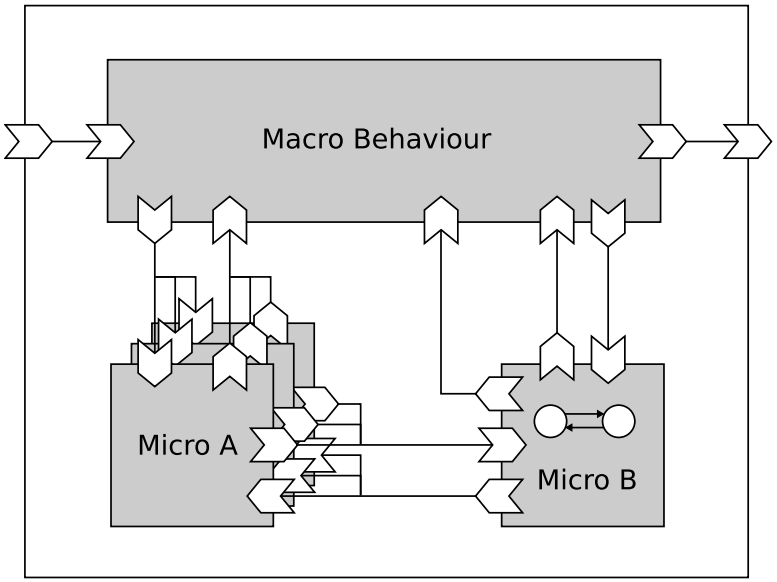
```
1   X = { free }
2
3   Y = { geneDock }
4
5   S = { (phase) | phase ∈ {unbound,
        bound} }
6
7   s_init = unbound
8
9   p = case phase of
10        unbound: (free);
11        bound: (geneDock);
12
13  δ = if (free == "bind") then
14        phase = bound;
15      if (elapsedTime == ta) then
16        phase = unbound;
17
18  λ = sendMessage(geneDock, "unbind")
19
20  ta = case phase of
21         unbound: ∞;
22         bound:
               expRandom(dissociationRate);
```



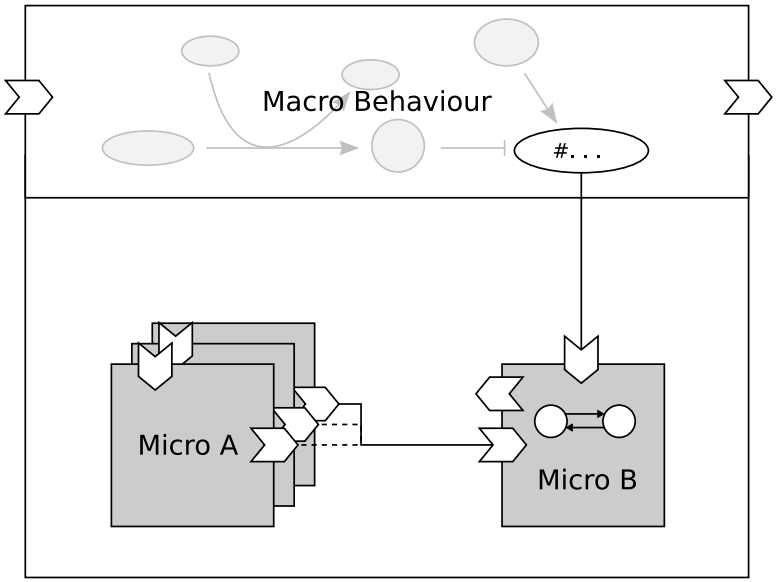
Atomic ml-DEVS model of the gene (micro)

```
1   X = { (free, bindingsite) }
2
3   Y = { producedMRNA }
4
5   S = { (phase) | phase ∈ {inactive, active} }
6
7   s_init = inactive
8
9   p = case phase of
10        inactive: (free);
11        active: (bindingsite, producedMRNA);
12
13  δ = if (free == "activate") then
14        phase = active;
15      if (bindingsite == "unbind") then
16        phase = inactive;
17
18  λ = sendMessage(producedMRNA, "mRNA")
19
20  ta = case phase of
21         inactive: ∞;
22         active: transcriptionRate;
```

Reduced model complexity with ml-DEVS



classical DEVS



ml-DEVS

Summary

- DEVS supports composition hierarchies
- Classical DEVS is restricted to static model structures, variable model structures supported by extensions
- In ml-DEVS,
  - Dynamic composition, interaction structures, and ports
  - Composition and abstraction (micro and macro) levels can be integrated in the same hierarchy
  - Upward and downward causation reduces efforts for micro-macro modeling
- DEVS emphasizes the reactive system perspective (as State Charts do)

Part III

π calculus

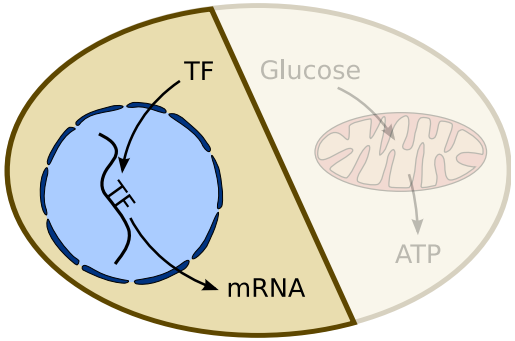
π calculus for Systems Biology

- stochastic π calculus timed extension of π calculus
- associates events with stochastic rates
- includes stochastic semantics $\longrightarrow$ direct mapping to Stochastic Simulation Algorithm (SSA)
- application rules for stochastic π calculus to Systems Biology given (”molecule as computation”)

Example in π calculus

Channel Definitions

enterNuc, *exitNuc*, *tfBind*, *prodMRNA*



Process Definitions

$$
\begin{aligned}
\mathit{TFCyt}() &= \mathit{enterNuc}?().\ \mathit{TFNuc}() \\
\mathit{Nuc}() &= \mathit{enterNuc}!().\ \mathit{Nuc}() + \mathit{exitNuc}!().\ \mathit{Nuc}() \\
\mathit{TFNuc}() &= \mathit{tfBind}!() + \mathit{exitNuc}?().\ \mathit{TFCyt}() \\
\mathit{DNA}() &= \mathit{tfBind}?().\ \mathit{DNATF}() \\
\mathit{DNATF}() &= \mathit{prodMRNA}!().\ (\mathit{MRNANuc}() \mid \mathit{DNA}() \mid \mathit{TFNuc}()) \\
\mathit{MRNANuc}() &= \mathit{exitNuc}?().\ \mathit{MRNACyt}()
\end{aligned}
$$

Initial Process

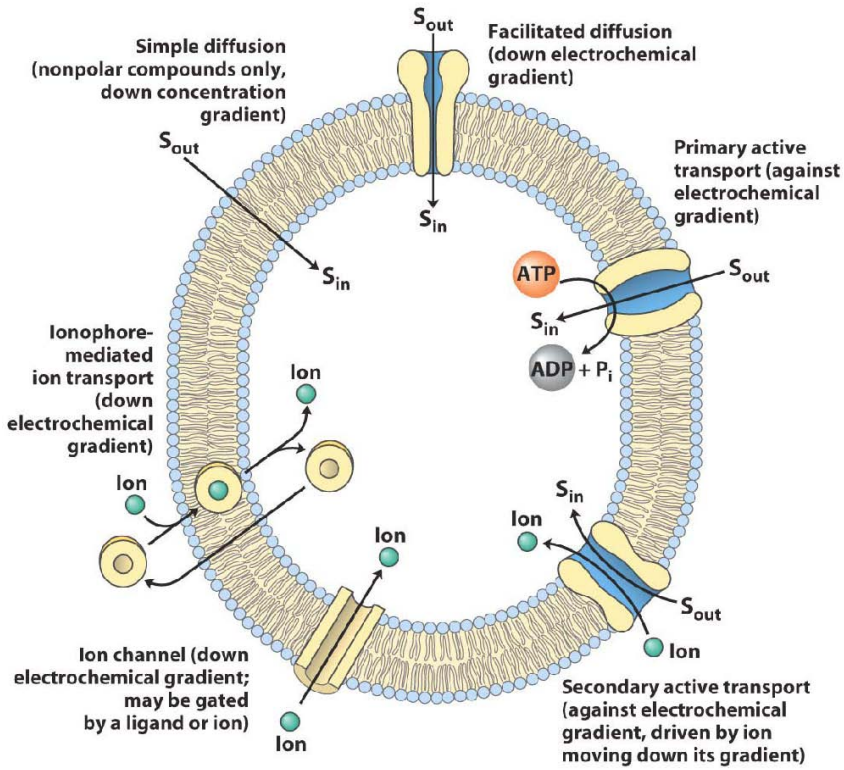
$(\mathit{TFCyt}() \mid ... \mid \mathit{TFCyt}() \mid \mathit{TFNuc}() \mid ... \mid \mathit{TFNuc}() \mid \mathit{Nuc}() \mid \mathit{DNA}())$

More Structure Needed for Hierarchical Modeling

- only π calculus elements: processes and channels
- more structure for support of hierarchical modeling needed
- different π calculus extensions, e.g.
  - Beta-binders
  - Bio Ambients

Basic Elements in Beta-binders

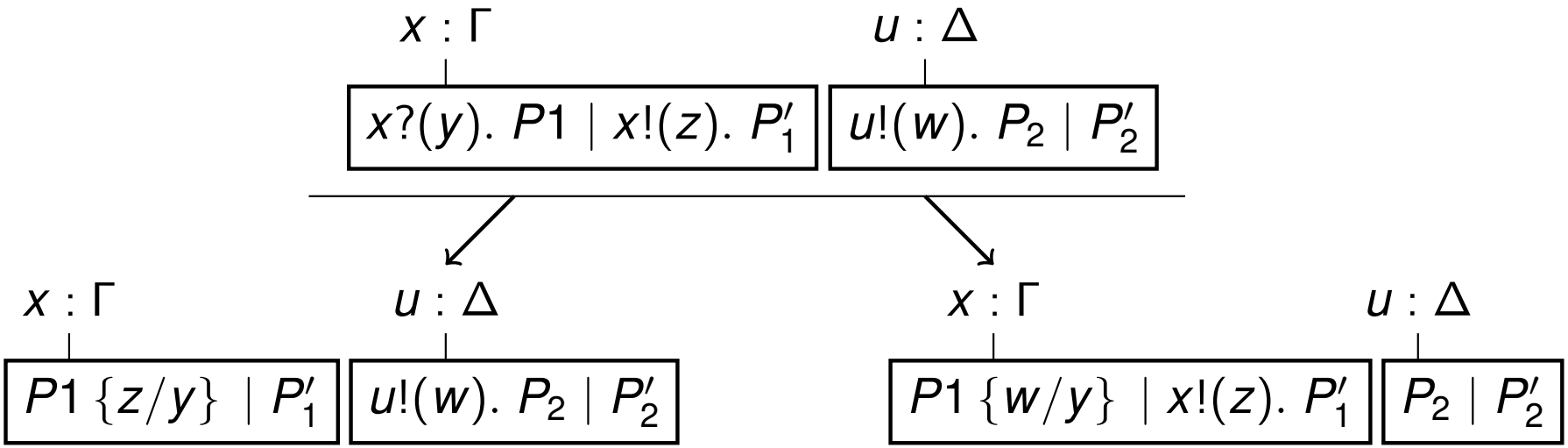
- π processes wrapped in boxes, bio-processes
- bio-processes with beta-binders
- beta-binders = sets of elementary beta-binders
- elementary beta-binders, form $\beta(x, \Gamma)$
- x = channel name, Γ = type
- types = sets of names



Biochemistry Lecture by Victor Munoz, University of Maryland,2006

Communication

- two kinds of communication
  - intra within bio-processes (as in π calculus)
  - inter between bio-processes
- communication between bio-processes only with elementary beta-binders with overlapping type sets



Modification

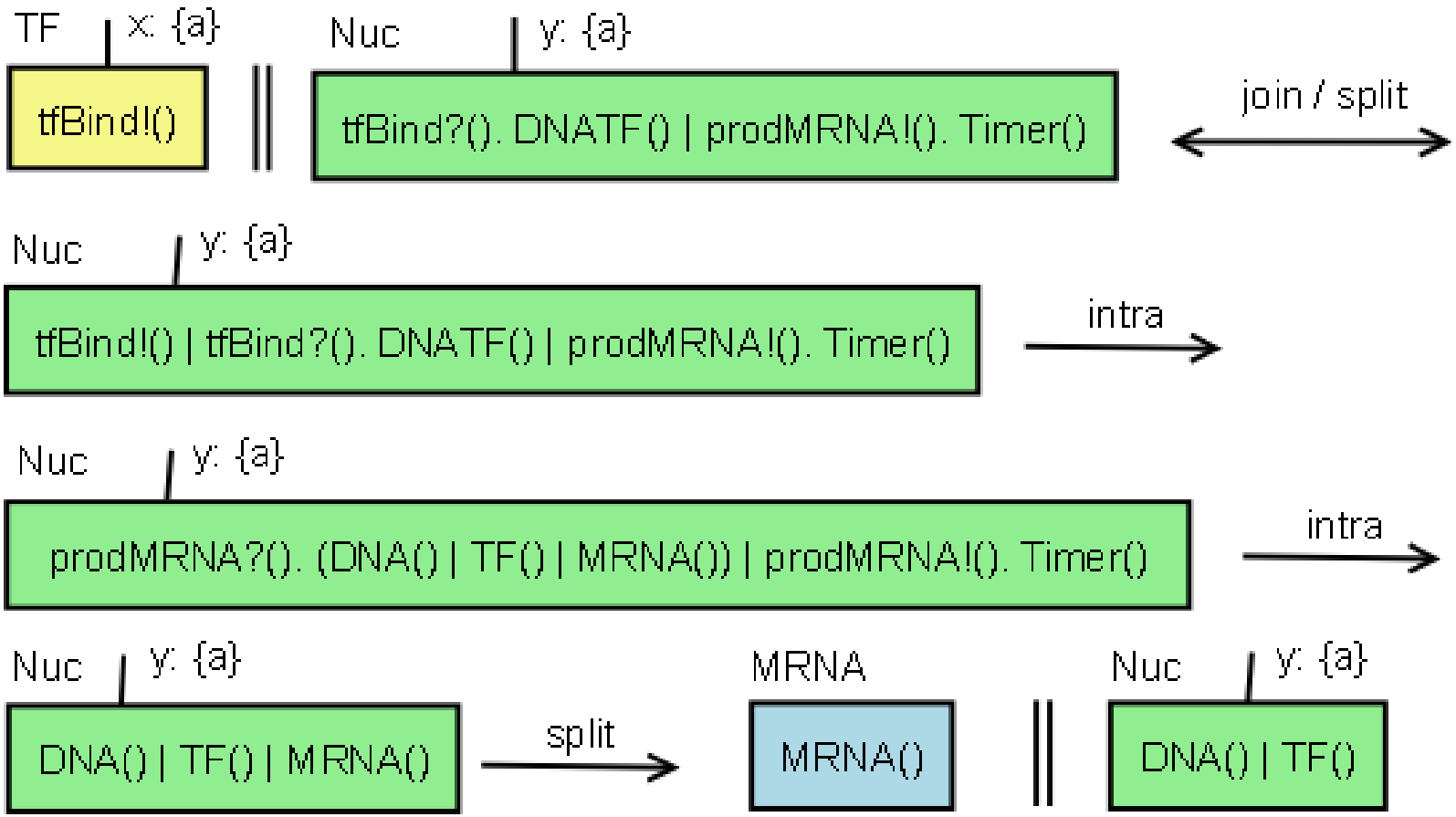
Modify beta-binders

- inner processes can modify beta-binders, 3 different operators
  - hide: disable communication on elementary beta-binder
  - unhide: enable communication on elementary beta-binder
  - expose: add fresh elementary beta-binder

Modify bio-processes

- generic functions to modify bio-processes
  - split: divide bio-processes
  - join: merge bio-processes

Example in Beta-binders



Bio Ambients Basics

Elements

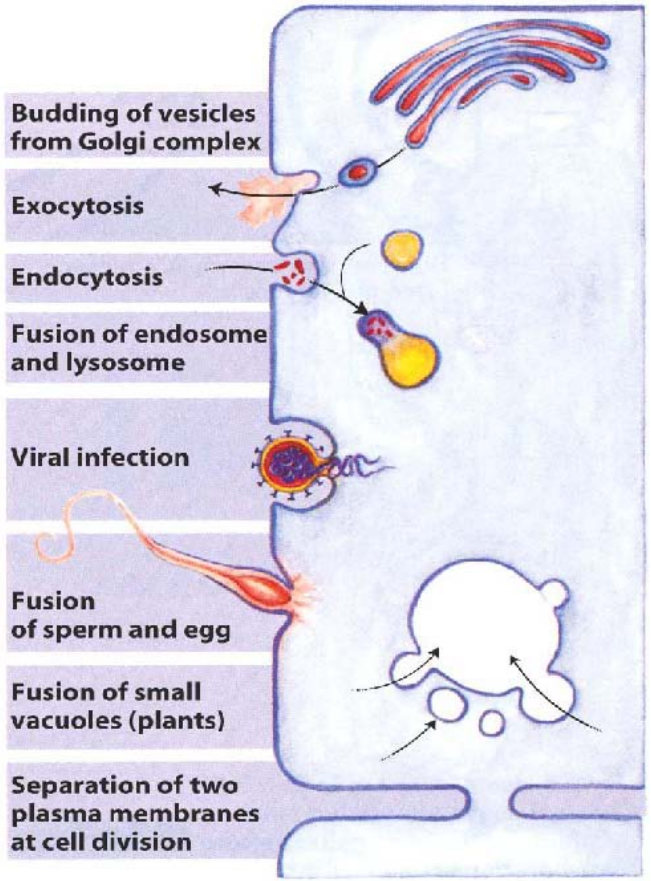
- processes wrapped in boxes, ambients
- ambients contain processes *and* ambients

Communication

- *p2c*, *c2p* from parent ambient to child and back
- *local*, *s2s* communication in ambient or between two “sibling” ambients

Modification

- *enter* / *accept* ambient enters sibling
- *exit* / *expel* ambient exits parent
- *merge* $+$ / *merge* $-$ siblings merge



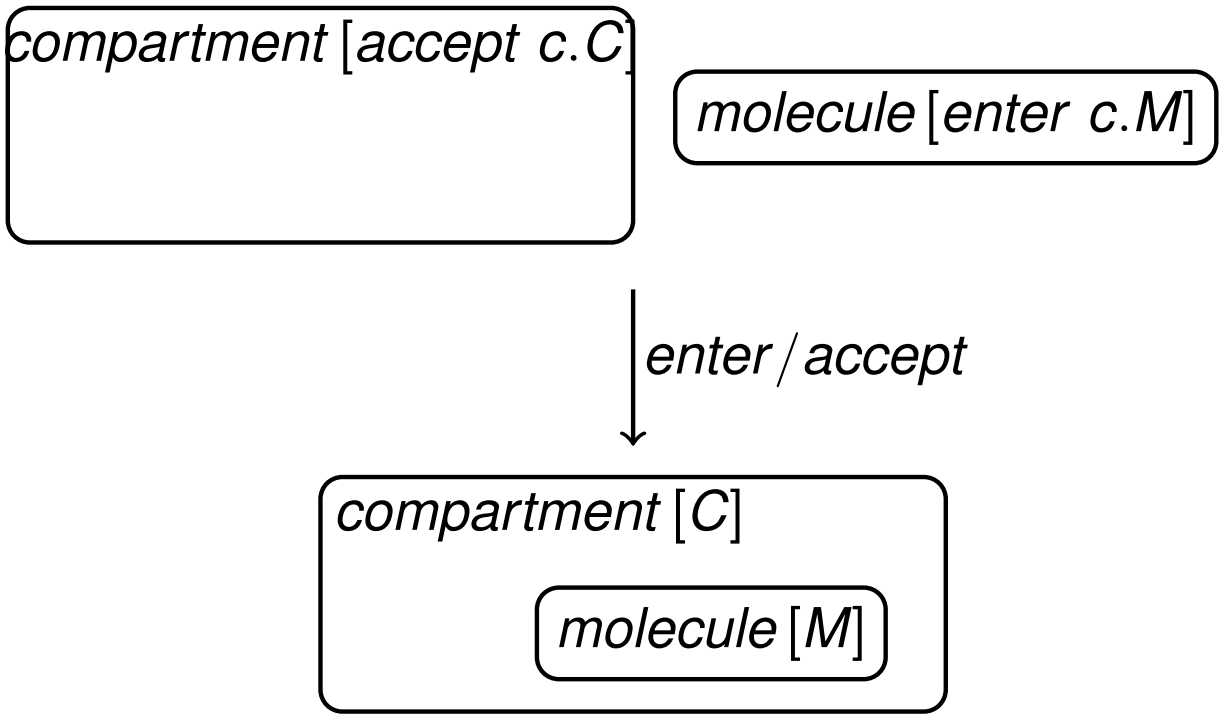
Biochemistry Lecture by Victor Munoz, University of Maryland, 2006

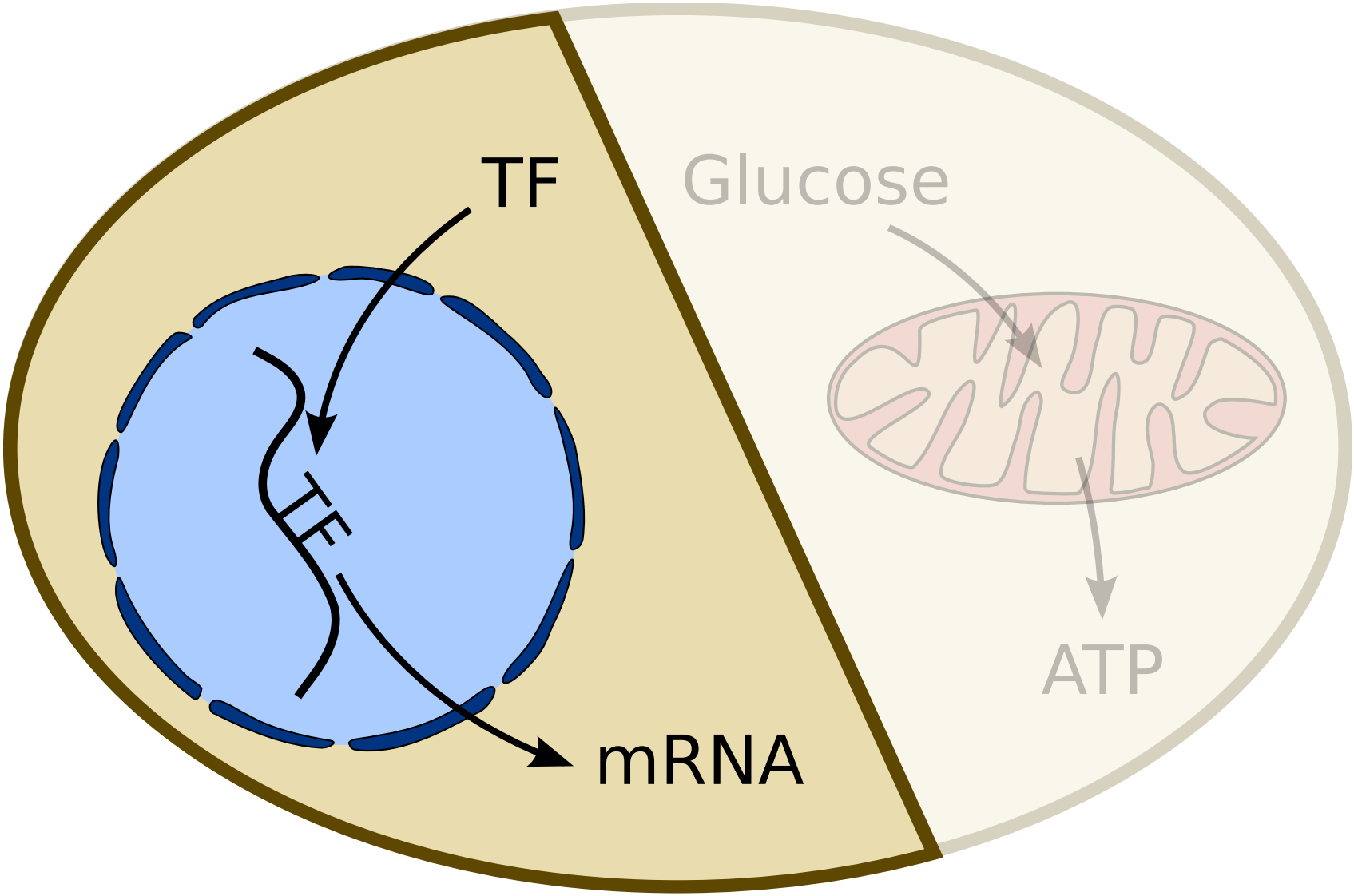
Syntax of Bio Ambients

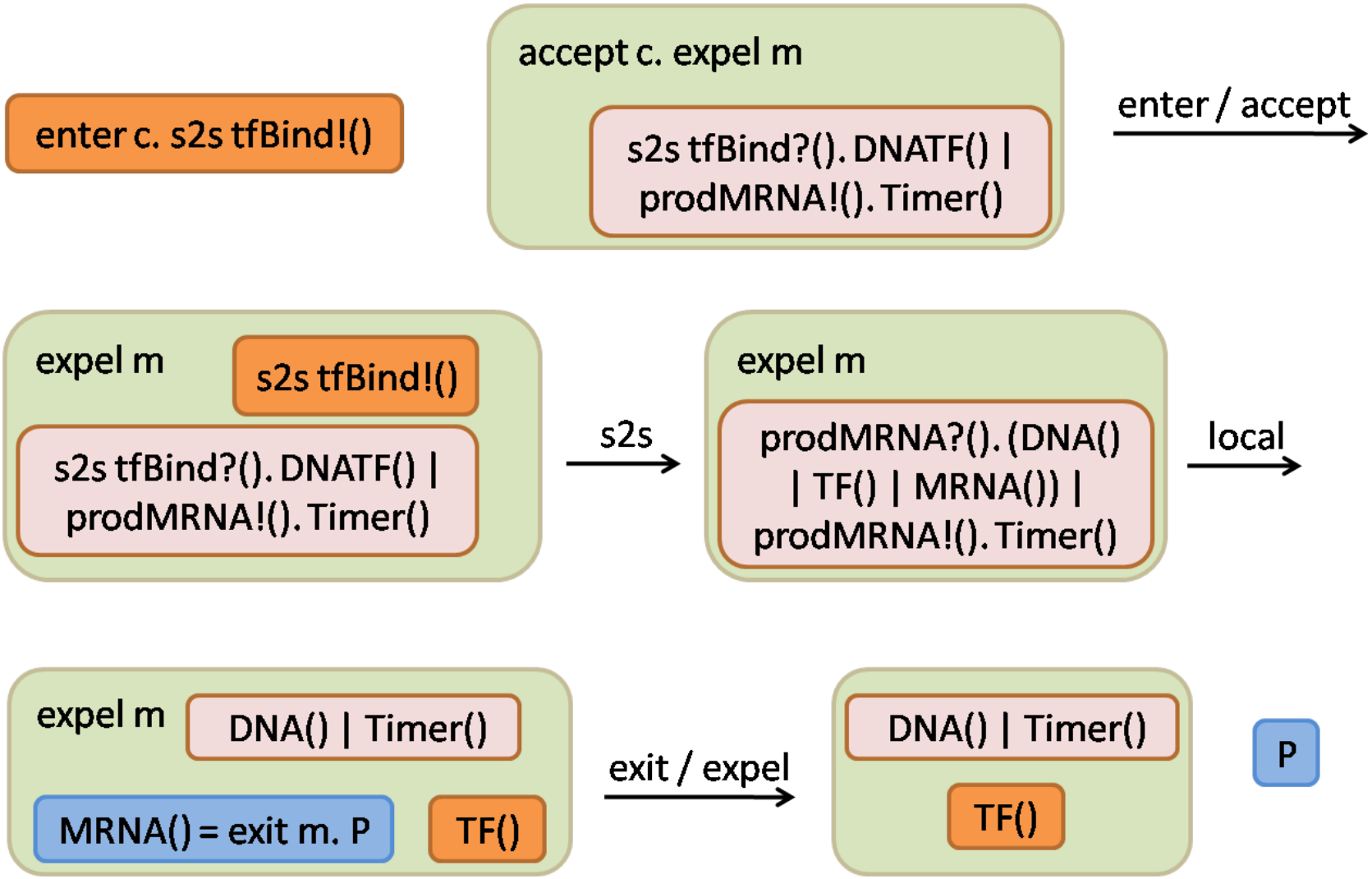
Process	P	$::=$	$P_1 \parallel P_2$	Parallel Composition
		$\mid$	$(\nu\ c).P$	ν Operator
		$\mid$	$\sum_i S_i$	Summation
		$\mid$	$[P]$	Ambient
Summation	S	$::=$	$\delta\ x!(y).P$	Send with Direction
		$\mid$	$\delta\ x?(y).P$	Receive with Direction
		$\mid$	$enter\ x.P$	Enter
		$\mid$	$accept\ x.P$	Accept
		$\mid$	$exit\ x.P$	Exit
		$\mid$	$expel\ x.P$	Expel
		$\mid$	$merge+\ x.P$	Merge+
		$\mid$	$merge-\ x.P$	Merge-
Direction	δ	$::=$	$local$	Processes in Ambient
		$\mid$	$s2s$	Ambients in Ambient
		$\mid$	$p2c$	Ambient to Nested Ambient
		$\mid$	$c2p$	Nested Ambient to Ambient

Example: Molecule Entry

- protein enters a compartment
- 2 ambients: *molecule*, *compartment*
- *molecule* provides *enter* on *c*, *compartment accept*
- synchronization $\rightarrow$ *compartment* contains *molecule*




TF
Glucose
TF
ATP
mRNA


enter c. s2s tfBind!()
accept c. expel m
s2s tfBind?(). DNATF() | prodMRNA!(). Timer()
enter / accept
expel m
s2s tfBind!()
s2s tfBind?(). DNATF() | prodMRNA!(). Timer()
s2s
expel m
prodMRNA?(). (DNA() | TF() | MRNA()) | prodMRNA!(). Timer()
local
expel m
DNA() | Timer()
MRNA() = exit m. P
TF()
exit / expel
DNA() | Timer()
TF()
P

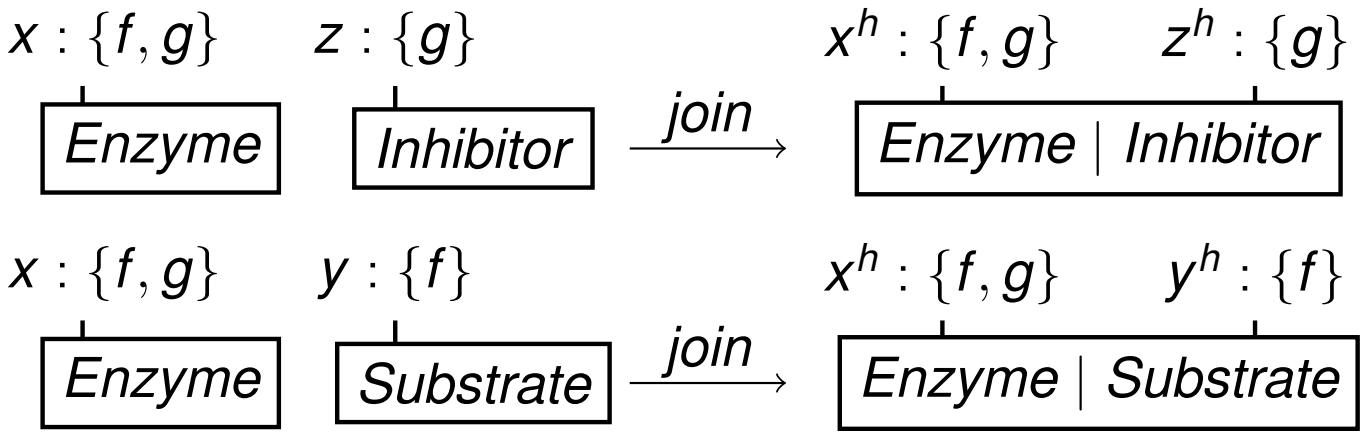
Extensions: Summary

- Beta-binders
  - wrap processes into boxes (bio-processes) with beta-binders to communicate
  - intra communication = normal π communication
  - inter communication only with overlapping types
  - modification: beta-binders (hide, unhide, expose), bio-processes (join, split)
- Bio Ambients
  - wrap processes into boxes (ambients), **nested**
  - communication directions: *p2c*, *c2p*, *s2s*, *local*
  - ambient modification: *enter*/*accept*, *exit*/*expel*, *merge* $+$ /*merge*$-$

Composition hierarchies in Beta binders

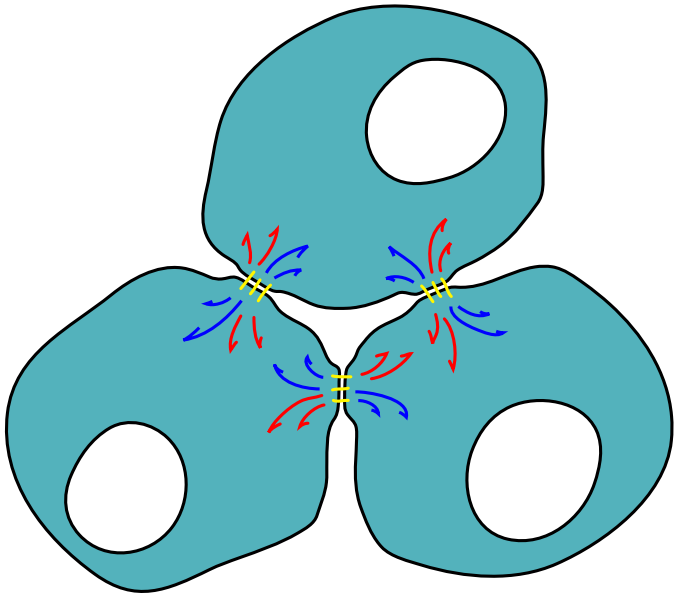
bio-processes

- separate inner processes from the outside
- beta-binders provide explicit interfaces
- model components similar to, e.g. in DEVS
- model structure highly flexible (*join*, *split*)
- *no hierarchies because no nesting*



Bio Ambients : Composition Hierarchies

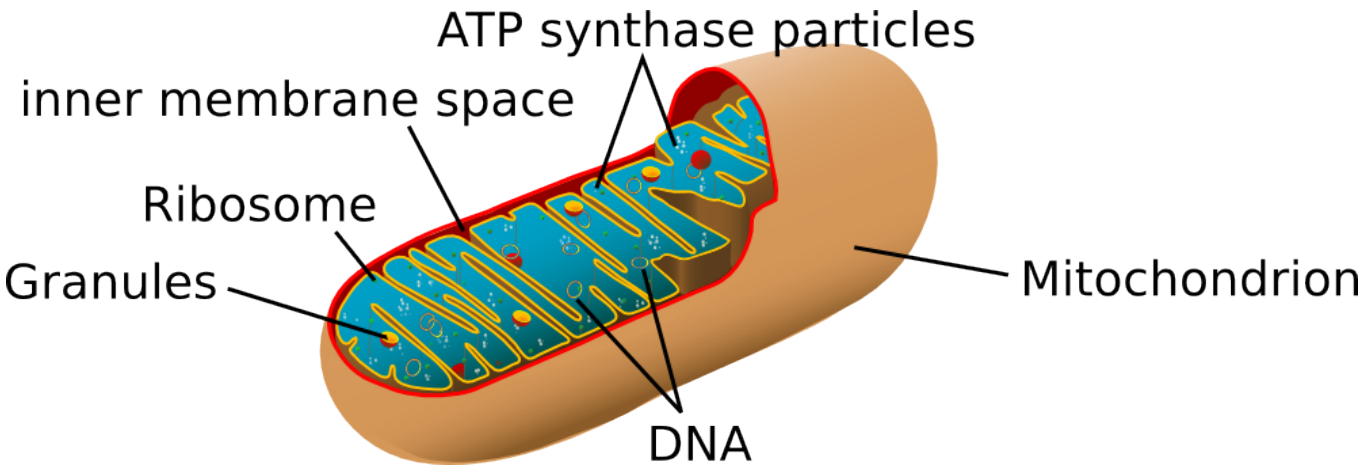
- ambients with explicit borders
- π calculus for describing interfaces
  - highly flexible



(modified from Wikipedia)

Bio Ambients : Composition Hierarchies by Nesting

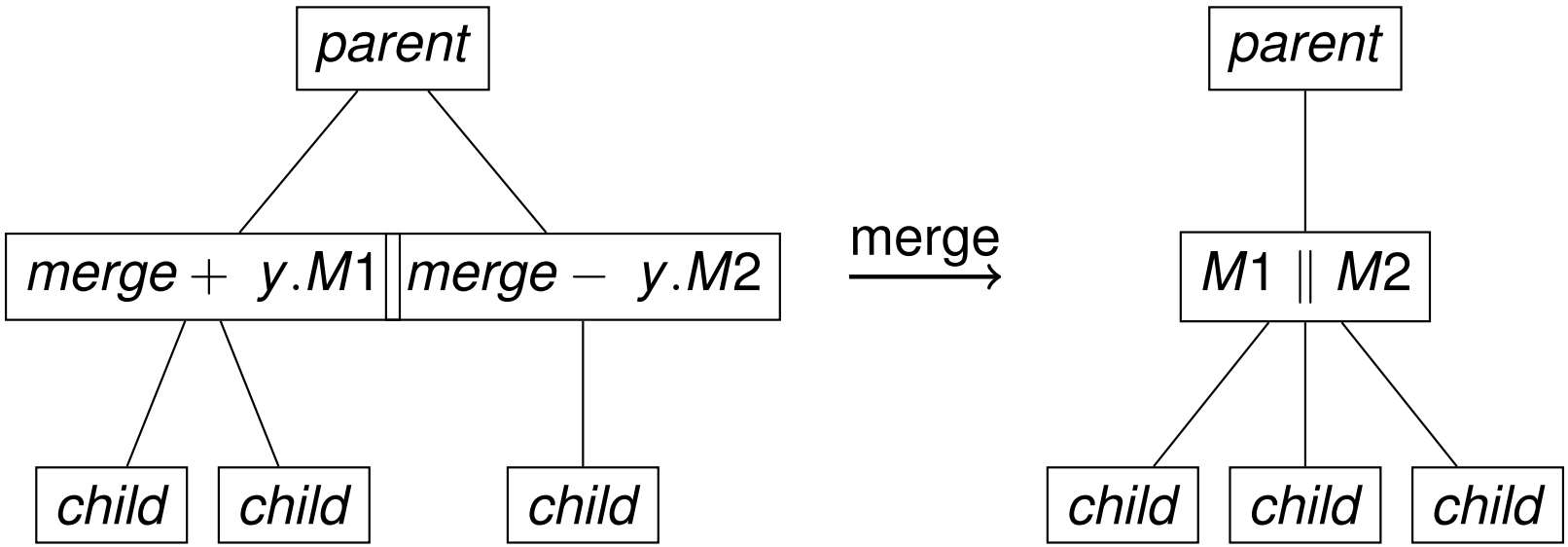
- nesting of ambients
- inner processes of ambient represent macro level for nested ambients (micro level)
- multilevel causation realized by communication directions
  - *p2c* downward causation
  - *c2p* upward causation
  - *s2s* on same level
  - *local* within one component

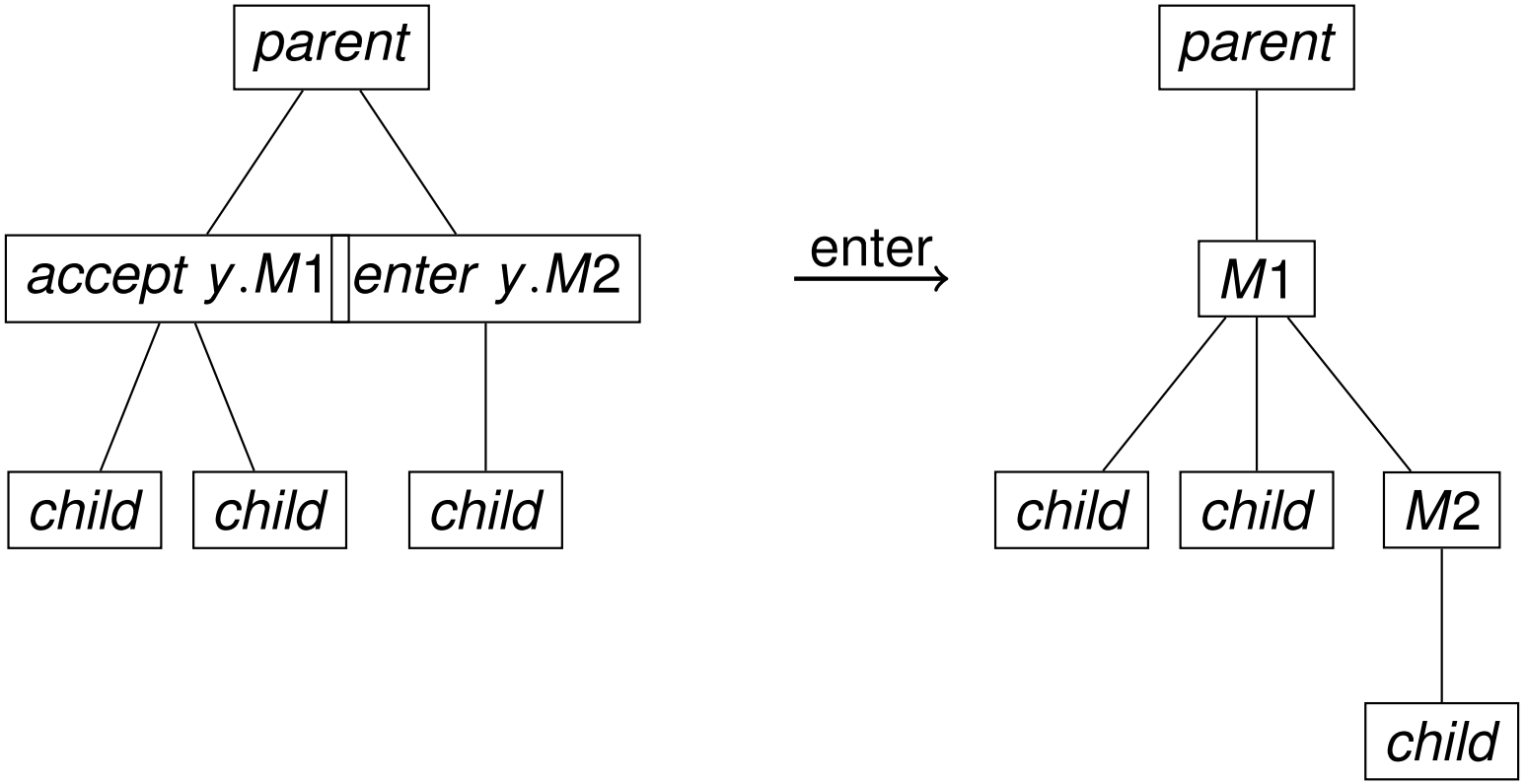


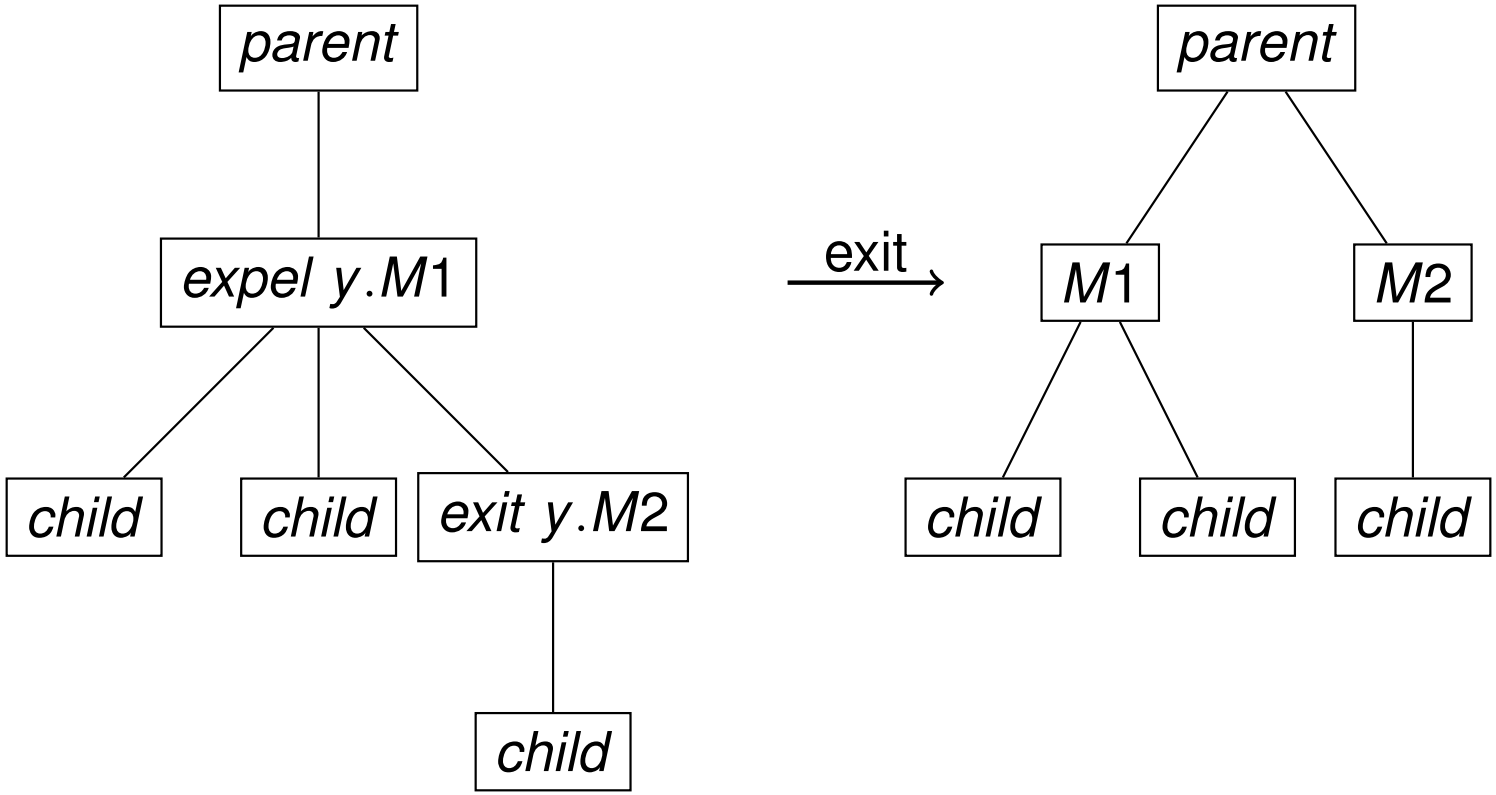
(modified from Wikipedia)

Bio Ambients : Modifying Hierarchies

- add, remove nodes with π calculus operations
- *merge* $+$ / *merge* $-$ melt two nodes (e.g. fusion of compartments)
- *enter*/*accept*, *exit*/*expel* subtree transfer (e.g. phagocytosis, cell's ejection of molecules)




parent
accept y.M1
enter y.M2
child
child
child
enter
parent
M1
child
child
M2
child


parent
expel y.M1
child
child
exit y.M2
child
exit
parent
M1
M2
child
child
child

π calculus extensions and composite hierarchies

- bio-processes
  - provide model components: beta-binders interfaces, boundary to the environment
  - dynamic interfaces: hide, unhide, expose
  - components flexible: join, split
  - however no hierarchies: no nesting of bio-processes
- ambients
  - provide model components: π processes interfaces (dynamic), boxes closure
  - hierarchies via nesting
  - hierarchical structure flexible: $merge+/merge-$, $enter/accept$, $exit/expel$

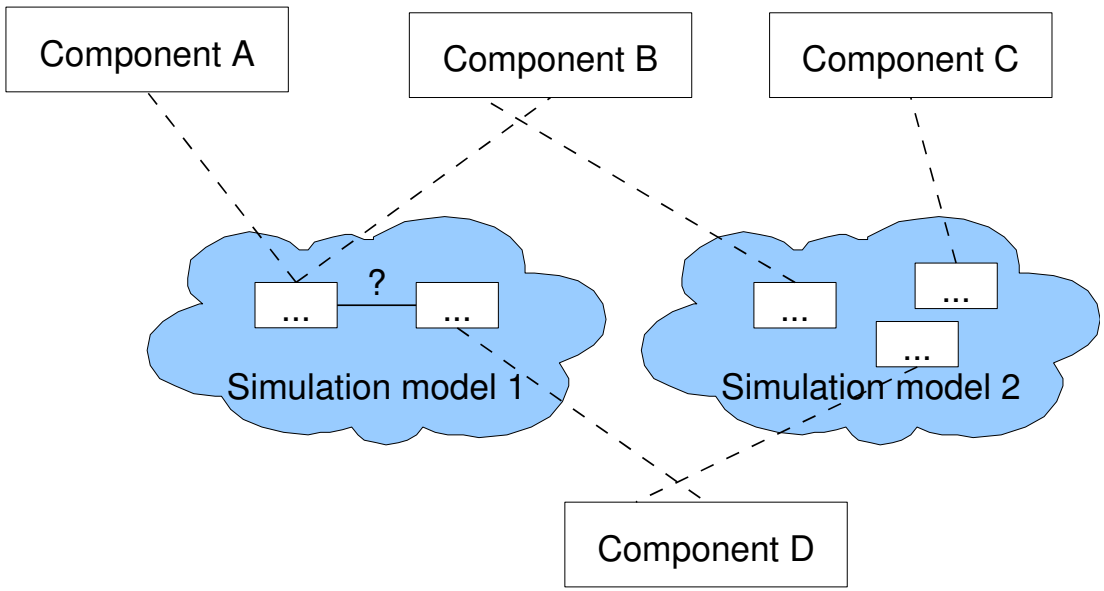
Part IV

Modeling by composition

What we really want in the end

Integrate

- Independently of each other developed components
- Into different systems, For different purposes
- Hiearchical composition



Compatibility — a prerequisite to composability

Levels of compatibility / interoperability

technical	Are components able to communicate?
syntactic	Do components use the same data structures?
semantic	Represent data structures the same things?
pragmatic	For what use has the model been built?

Components & Compositions

Derive meaning of a composition from

- Meaning of the parts
- Rules of combination

Feasible through

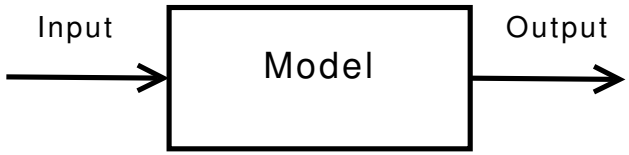
- Interface describing relevant properties
- Relation between Interfaces (Compatibility)
- Relation between Interfaces and Implementations (Refinement)

Compare to

- SBML: exchange of entire simulation models, not parts
- Modular-hierarchical modeling formalisms: no separate interface

Announce Points of Interaction

Which events can be send / received?
When can two models be coupled?



Recall DEVS

- Exchangeable events defined by sets
- Coupling constraint: $Sendable_{ModelX} \subseteq Receivable_{ModelY}$

Modeling and simulation tools

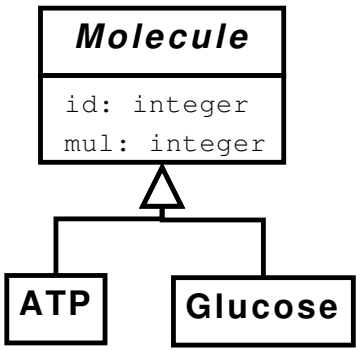
- Programming languages, e.g. Java
- Subtype relations, e.g. inheritance

XML-based approaches

- Not tool-specific
- Easily extensible data structures

Type definitions — Example

- Abstract type `Molecule`
  - Identifier
  - Multiplicity (amount)
- Concrete types
  - derived from `Molecule`
  - e.g. `ATP`, `Glucose`
- But
  - What kind of molecules are represented by the data structures?
  - How to integrate semantics?



Use: XML Schema Definitions

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="unihro/cbio/molecules"
  targetNamespace="unihro/cbio/molecules"
  xmlns:sawsdl="http://www.w3.org/2002/ws/sawsdl/spec/sawsdl#">
   <xs:complexType name="Molecule" abstract="true">
     <xs:sequence>
       <xs:element name="id" type="xs:integer" minOccurs="1"/>
       <xs:element name="mul" type="xs:integer"/>
     </xs:sequence>
   </xs:complexType>
  <xs:complexType name="ATP" sawsdl:modelReference=
        "http://www.genome.jp/dbget-bin/www_bget?cpd:C00002">
    <xs:complexContent>
      <xs:extension base="Molecule">
      </xs:extension>
    </xs:complexContent>
   </xs:complexType>
  <xs:complexType name="Glucose" ...
</xs:schema>
```

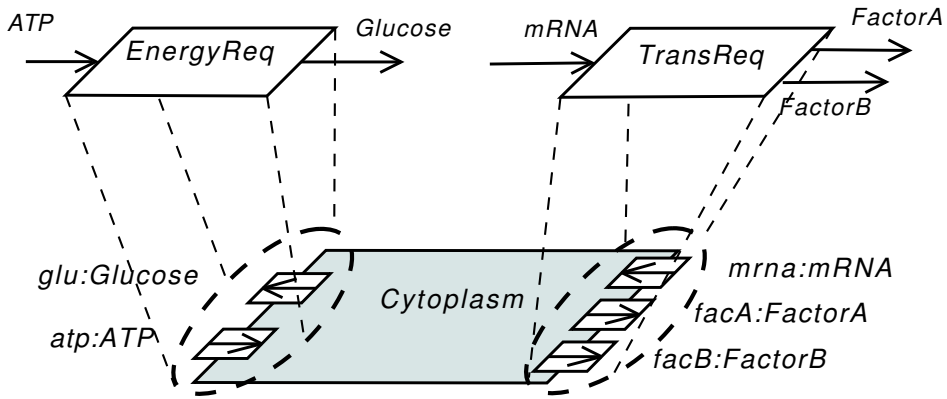
Roles — Complex Points of Interaction

Interfaces (e.g. for software and services)

- patterns of interaction (at least the structural part)
- group atomic points (methods) of interaction
- separated from implementation
- may be referenced from different components
- function as contracts

Roles

- extract interface information
- wrt. a certain aspect
- declares a set of directed event ports with a logical relation

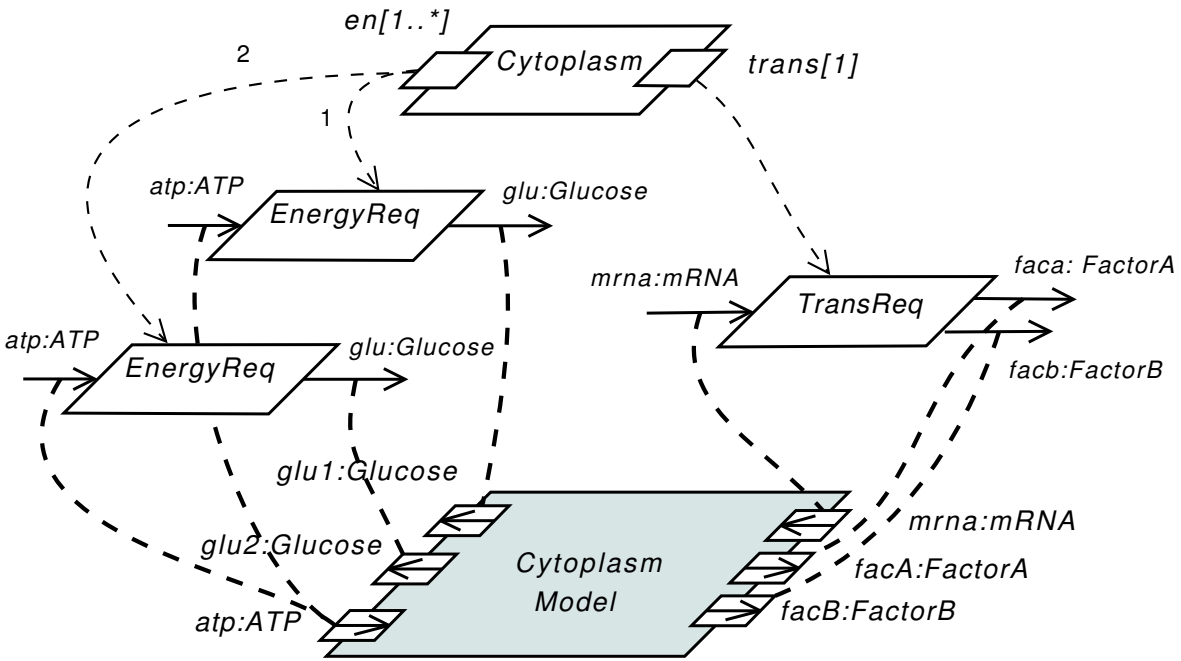


Interfaces — Representation in XML

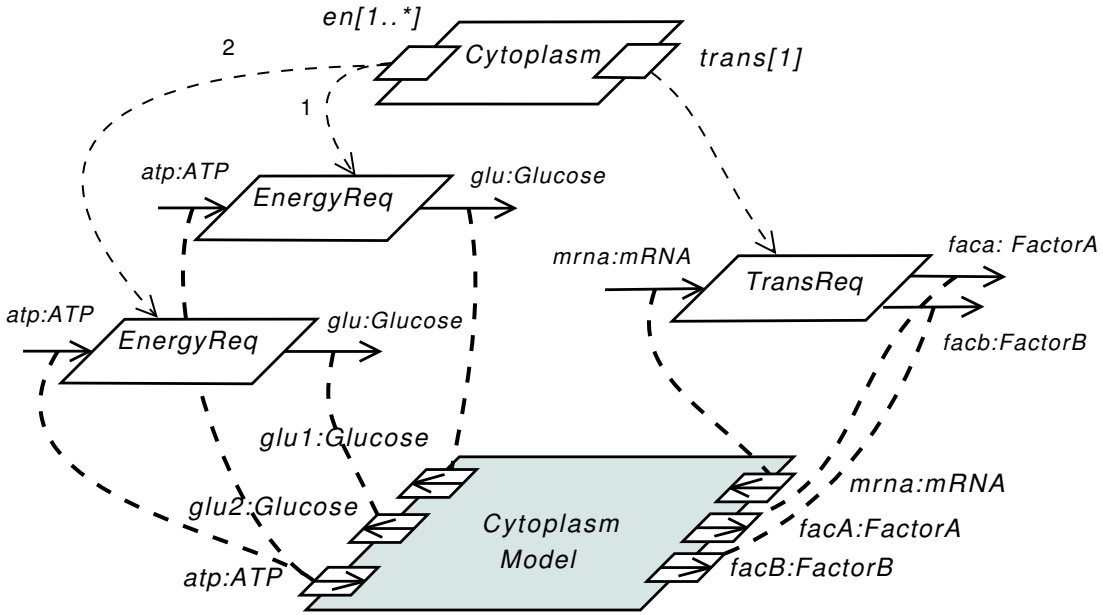
```
<interface xmlns="http://www.informatik.uni-rostock.de/cosa/publici"
           xmlns:cyto="unihro/cbio/cytoplasm"
           xmlns:cell="unihro/cbio/cell">
  <id>cyto:interface</id>
  <profile>
    <name>Cytoplasm</name>
    <application_domain>Cell simulation</application_domain>
    <description>Simple model of the a cell's cytoplasm</description>
    <objective>Represent all cell activities except that of the nucleus
        and the mitochondria</objective>
    <key_abstractions>May only be coupled to a nucleus and a set of
        mitochondria.</key_abstractions>
    <author>Mathias Roehl</author>
  </profile>
  <param name="mito" type="http://www.w3.org/2001/XMLSchema:int"
      value="1" description="number of mitochondria this model should
      be coupled to"/>
  <port minMultiplicity="1" maxMultiplicity="*">
    <name>en</name>
    <type>cell:EnergyReq</type>
  </port>
  <impl>cyto:impl</impl>
</interface>
```

Interface — Example

- Cytoplasm connected to 2 mitochondria
- Use different output ports to send `Glucose`
- $\mathscr{B} = \{(\text{“en”}, \text{“Glucose”}, 1, \text{“glu1”}), (\text{“en”}, \text{“Glucose”}, 2, \text{“glu2”}), \ldots\}$



Preserving Refinement — Example



Port “trans”

- multiplicity of 1
- For each declared port an implementation port with the same type exists

Port “en”

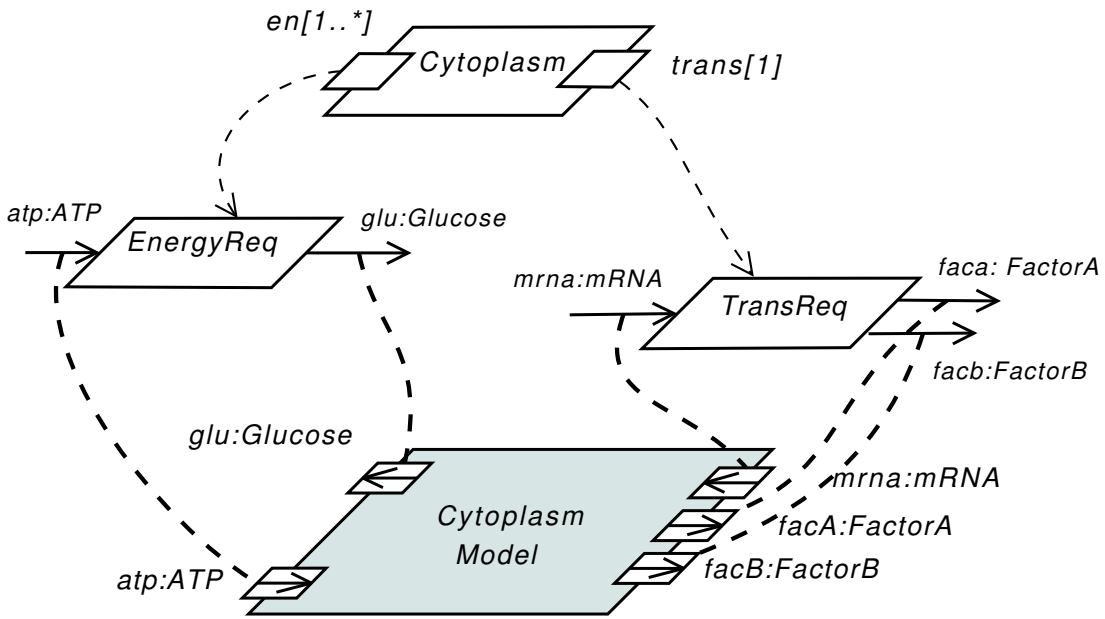
- For [1,2] declared ports are bound to model ports
- For [3,*] no model port exists!
- $\rightarrow$ no preserving refinement

Preserving Refinement — Example cont.

possible solutions

a) change multiplicity of port “en” to [1..2]

b) change model implementation and binding (one standard port for all multiplicites)



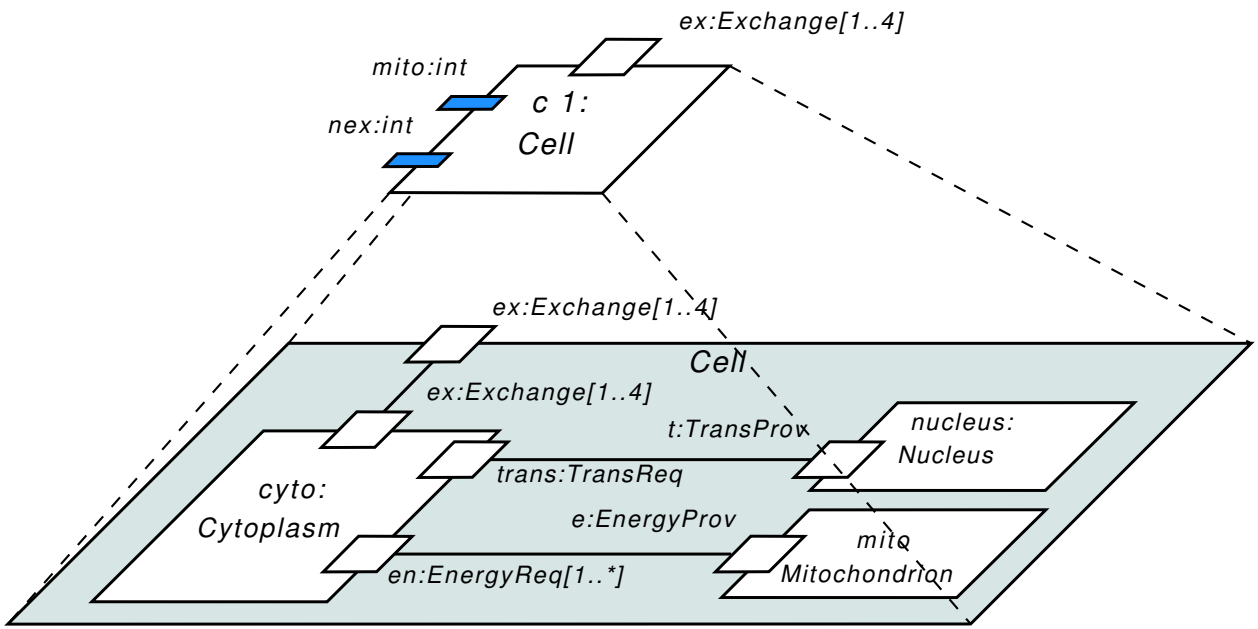
Composition

- Make no assumption about implementation
- Use component by
  - Qualified name to its interface
  - Set of parameter values
- Connect components
  - Publicized ports of the interfaces
  - at a certain position within allowed multiplicities
  - no other relations between components

Hierarchical Composition — Example

The cell as a composite component

- May itself be composed with other cells
- Publishes the port “ex”



Configuration

Parameters

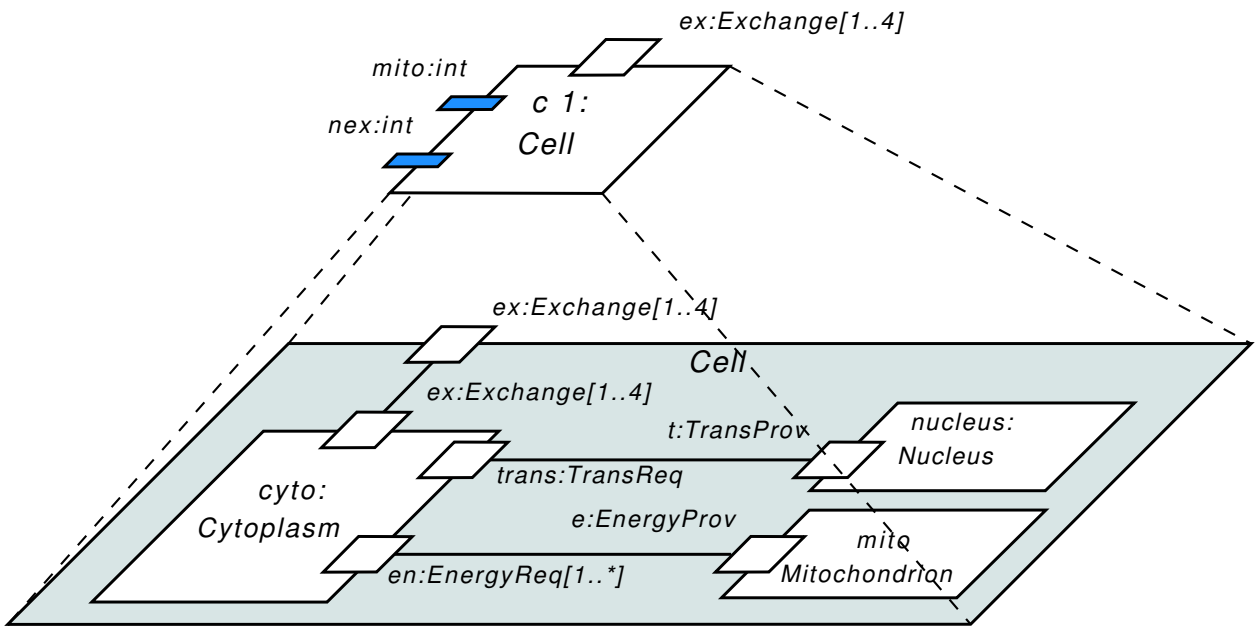
- Declared by interfaces
- What effect do parameters have?

Configurator function:

- Optional element of a component
- Changes internal structure of a component according to a concrete parametrization
- For atomic components: change contained model definition
- For composite components: change composition structure

Configuration — Example

- Parameter to configure the number of mitochondria it contains
- Configurator has to add according sub components and connections

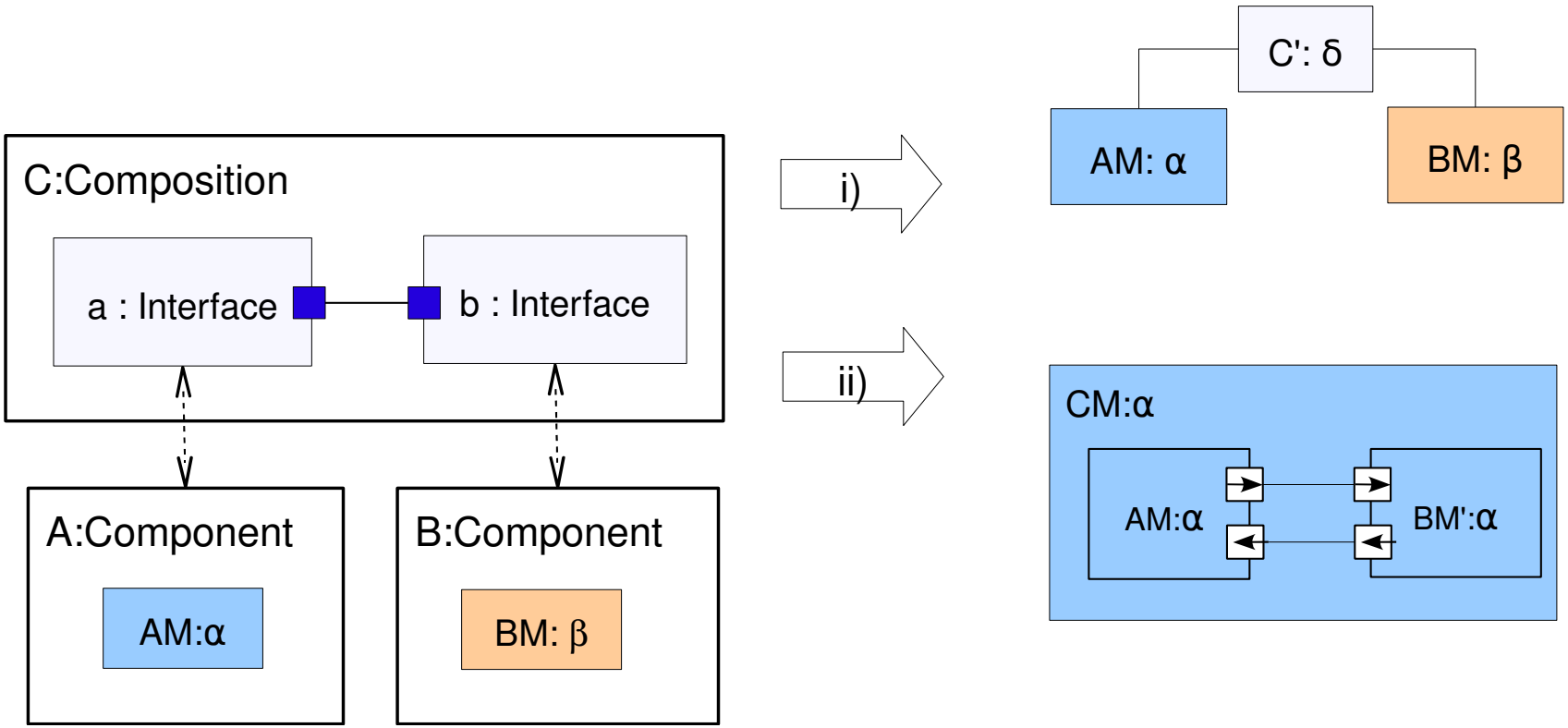


Component

Comprise

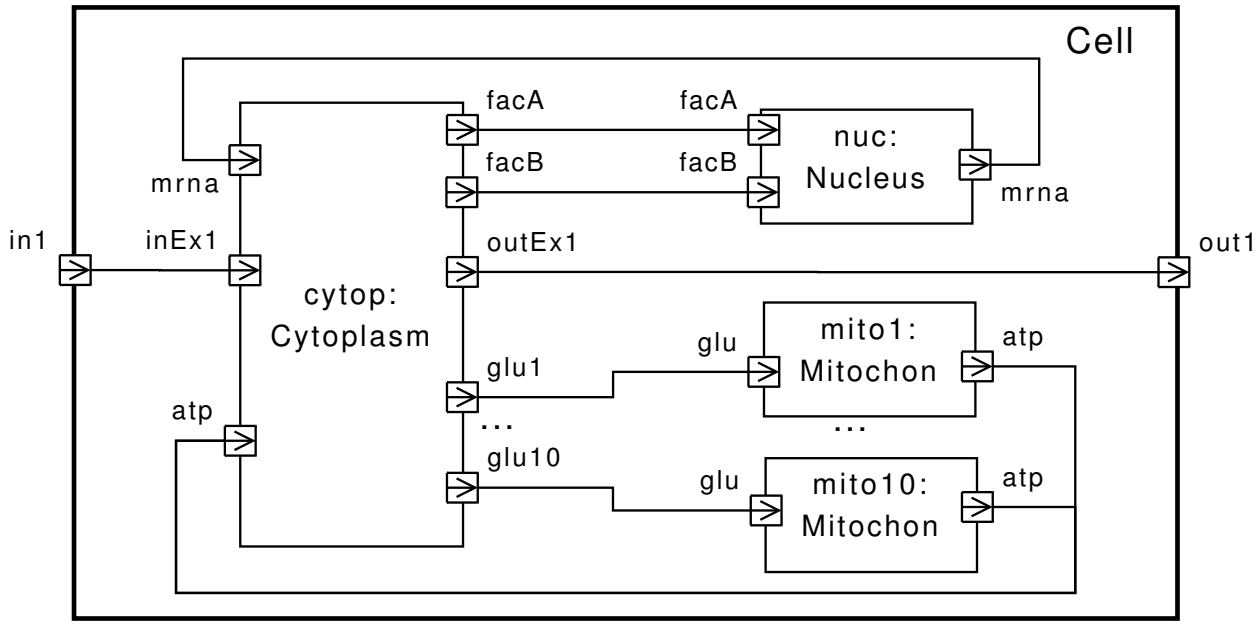
- Unique identifier
- Reference to an interface definition
- Model definition in a certain formalism
- A set of bindings to connect declared and implemented interaction capabilities
- References to sub components via their interfaces (optional)
  - Named
  - Parametrized
- Connections between sub components (optional)
- Configurator (optional)

Simulation model



Simulation model — Example

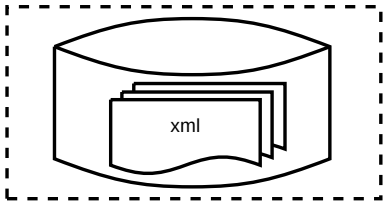
implied structure at the atomic interaction level for the parameter values “mito”=10 and “nex”=1



Stages of the composition phase

Source

- XML documents
- From (distributed) DB



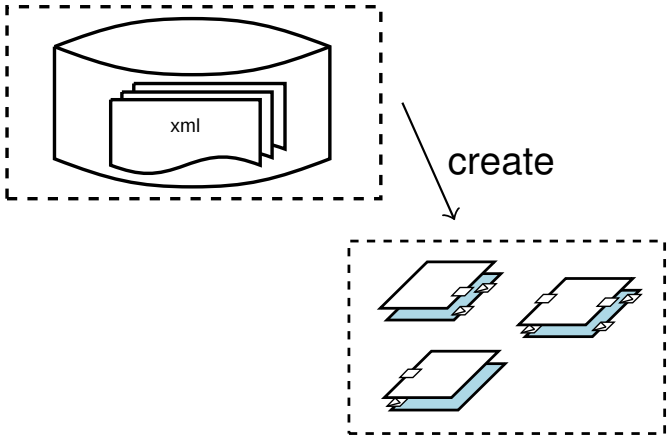
Stages of the composition phase

Source

- XML documents
- From (distributed) DB

Components

- Publish interfaces
- Customizable with parameters

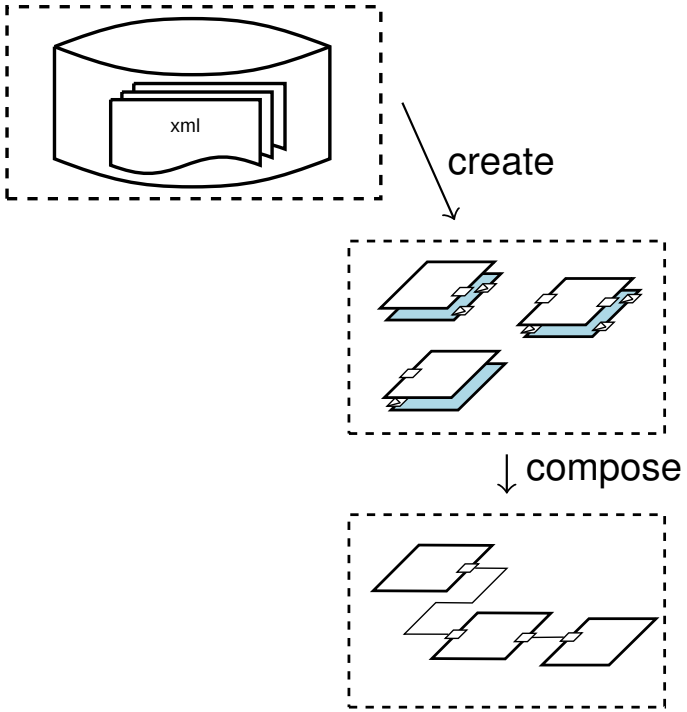


Stages of the composition phase

Source
- XML documents
- From (distributed) DB

Components
- Publish interfaces
- Customizable with parameters

Compositions
- Based on interfaces
- Hierarchical



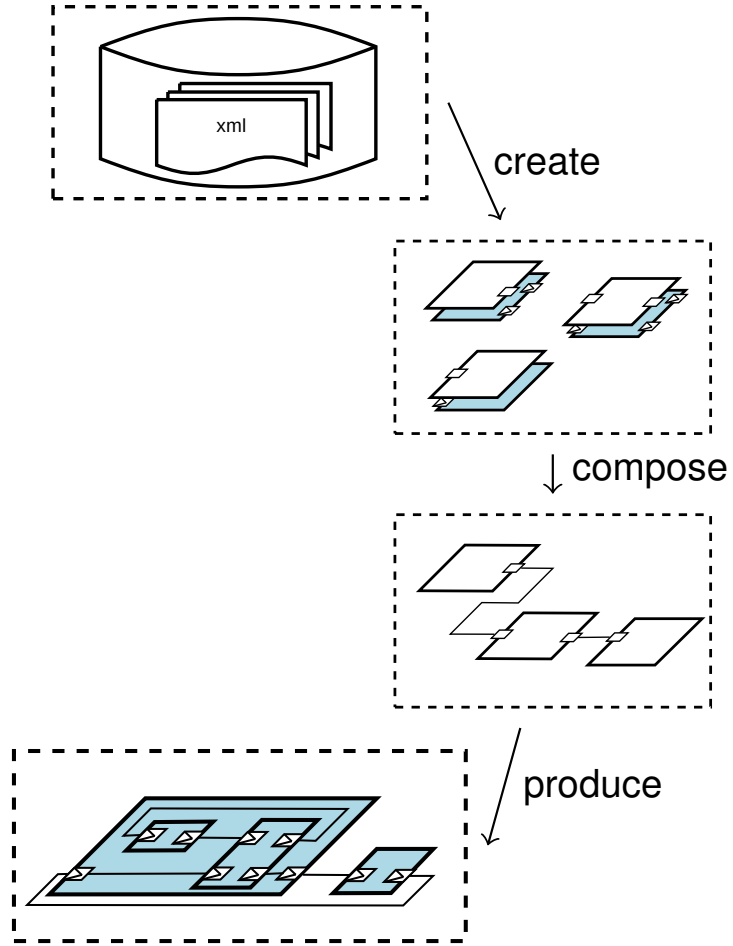
Stages of the composition phase

Source
- XML documents
- From (distributed) DB

Components
- Publish interfaces
- Customizable with parameters

Compositions
- Based on interfaces
- Hierarchical

Target
- Executable model
- e.g. Parallel DEVS, or multi-formalism model



Components Summary

Interfaces are stored separately from the model, refer to

- Types
  - platform independent,
  - support storage, retrieval, comparison
- Roles
  - group atomic interaction capabilities
  - reuse interaction capabilities for different model components,
- Parameters: configure a component to a specific usage context

Part V

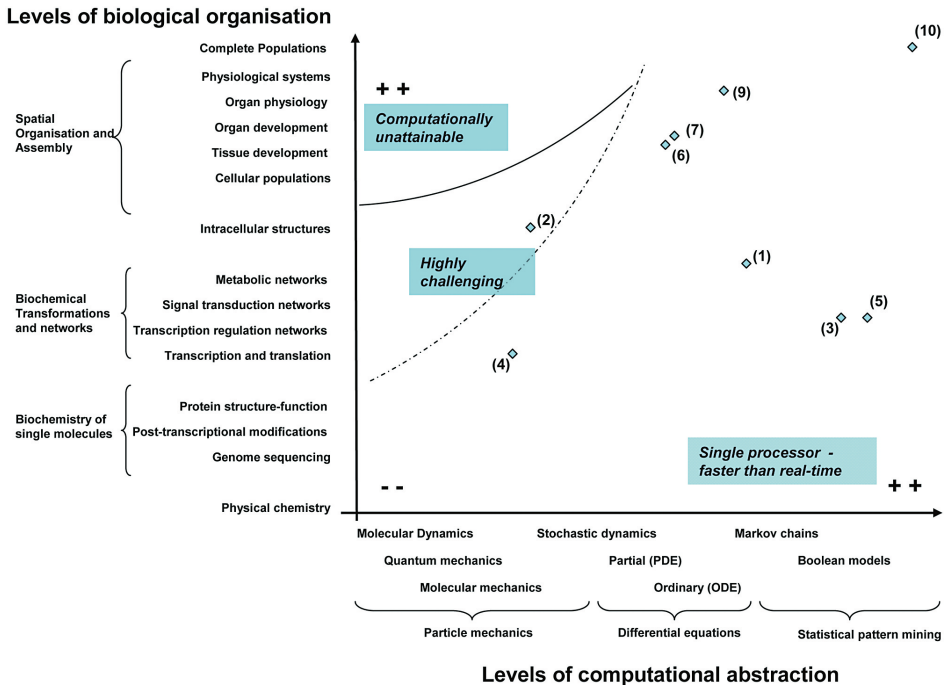
Summary

Compositional hierarchies in formalisms

explicit composition structures DEVS , Beta-binders , Bio Ambients

nesting of composition structures DEVS , Bio Ambients

Does this help us in describing systems at different levels of abstraction?



Different levels of abstraction

can be integrated implicitly, steps towards making explicit:

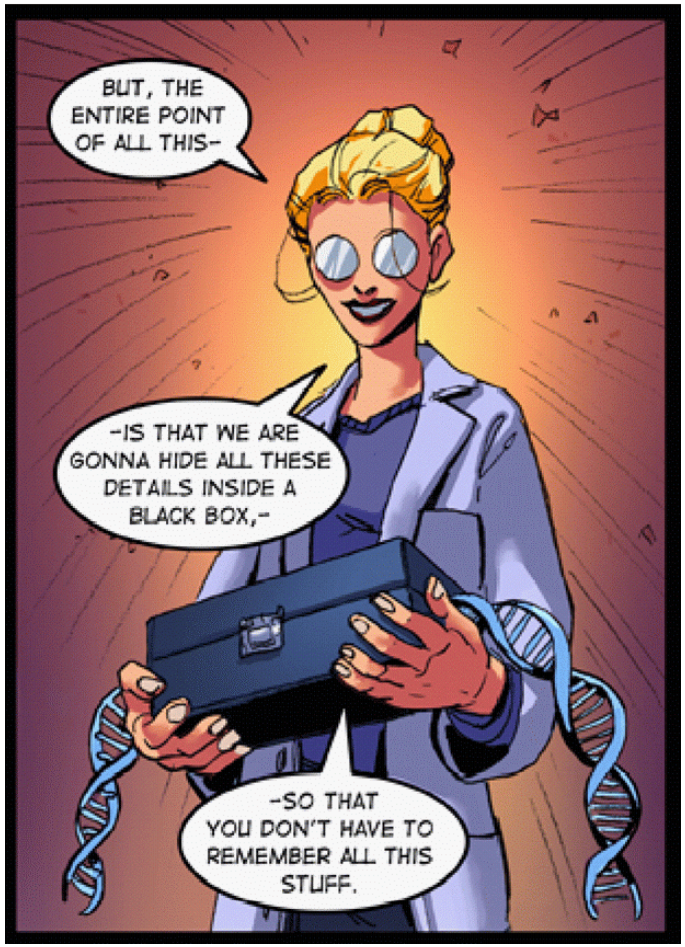
- dynamic composition structures: extensions of DEVS , bio-processes , Bio Ambients (introducing differences between “normal” dynamics and significant structure changes),
- explicit multi-level modeling: ML-DEVS, Bio Ambients (which refers to integrating and combining different abstractions, i.e. macro level and micro level perception)

Re-use for hierarchical, compositional modeling

supported by

- distinction between interface and model implementation
- use of different hierarchical relations
  - type hierarchies (syntax hierarchies, ontologies)
  - refinement between interfaces
  - composition hierarchies
- components in different modeling formalisms (and thus at possible different abstraction levels) can be combined

still a long way to go.



Adventures in Synthetic Biology, Drew Endy, 2005

THE END

Thank you!