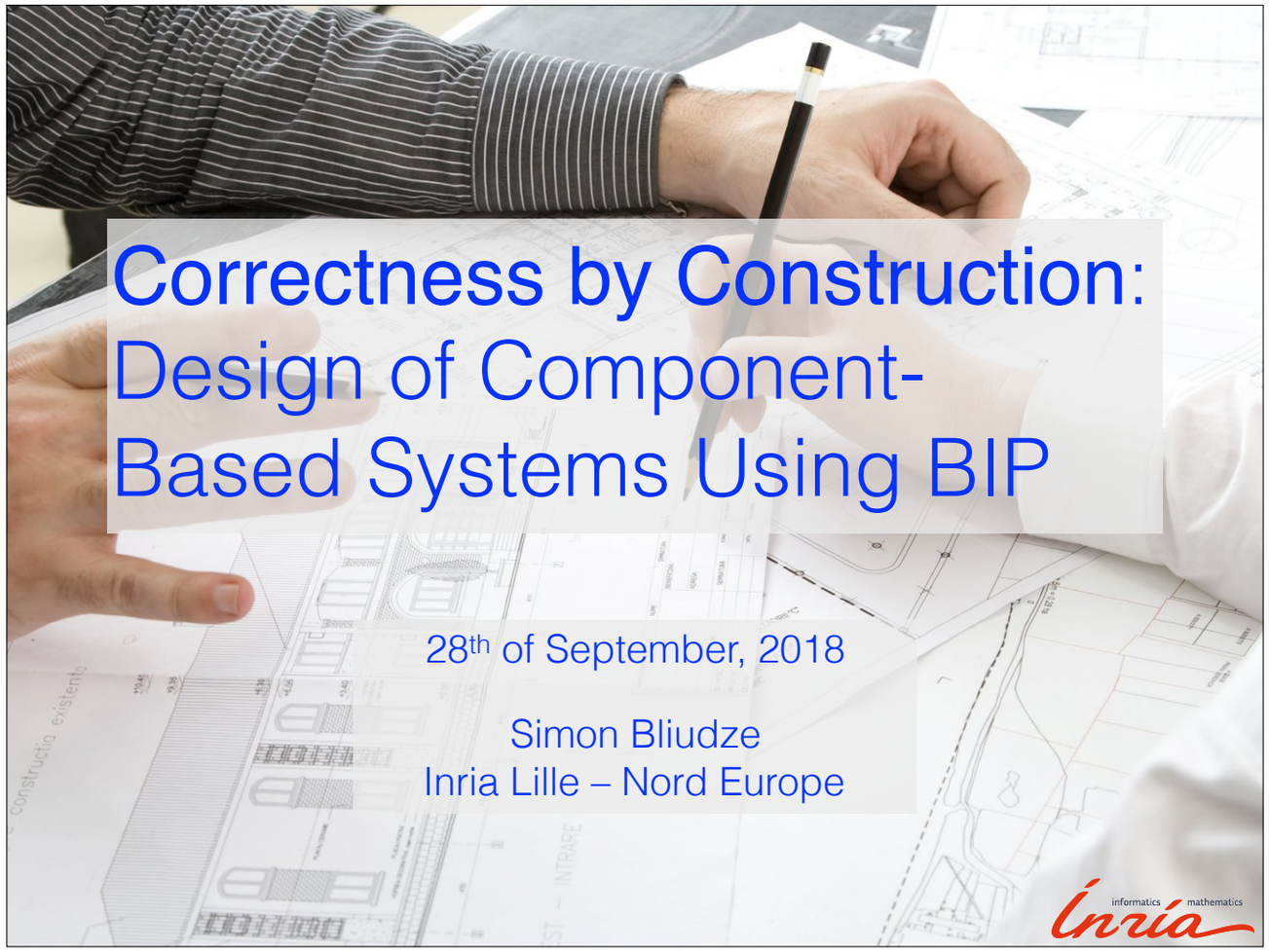
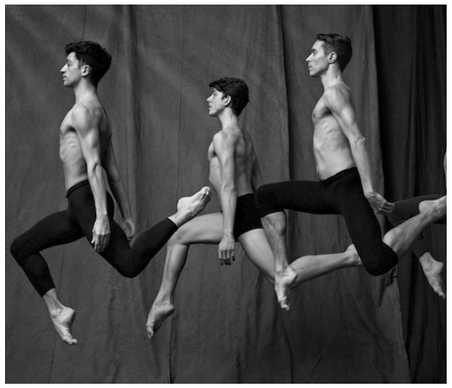


Correctness by Construction: Design of Component-Based Systems Using BIP

28th of September, 2018

Simon Bliudze
Inria Lille – Nord Europe

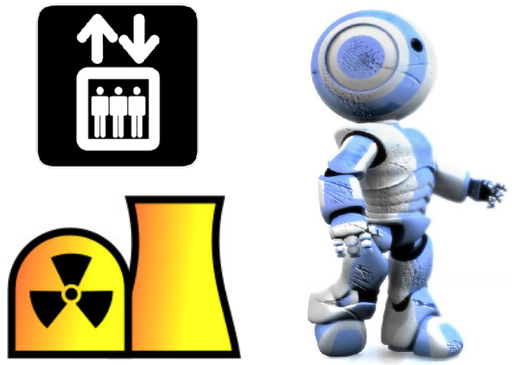
Concurrency...

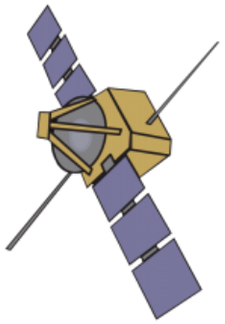


American cavalrymen charging up San Juan Hill, Cuba, 1 July 1898

Ruggeri, 2004

...is everywhere!





Embedded

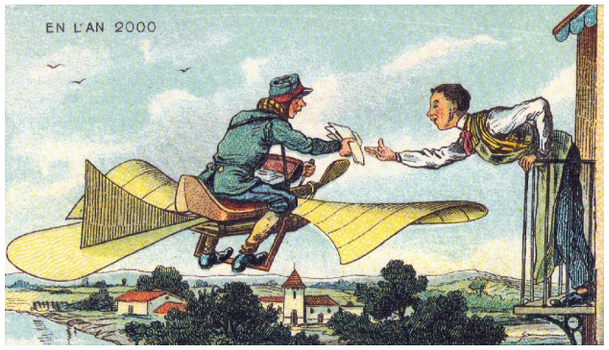
Infrastructure

Platform

Services

...you name it!

Coordination



The two views are complementary

Control-centric

- Synchronisation is primitive
- Locks, semaphores etc.
- Concurrent execution
- Critical systems

Data-centric

- Data exchange is primitive
- Messages, split-join etc.
- Distributed execution
- Data-intensive computation

Semaphores, locks, monitors, etc.

Coordination based on low-level primitives rapidly becomes unpractical.

Synchronisation

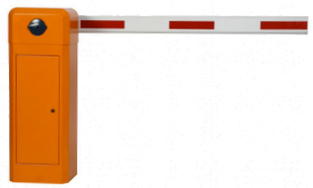
Process 1:

```
...
free(S1);
take(S2);
...
```

Process 2:

```
...
take(S1);
free(S2);
...
```

A simple synchronisation barrier



Synchronisation

Process 1:

```
...
free(S1);
free(S1);
take(S2);
take(S3);
...
```

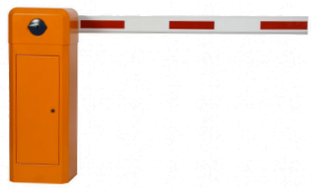
Process 2:

```
...
take(S1);
free(S2);
free(S2);
take(S3);
...
```

Process 3:

```
...
take(S1);
take(S2);
free(S3);
free(S3);
...
```

Three-way synchronisation barrier



Synchronisation with data transfer

Process 1:

```
x = f1(sh1,sh2);
free(S1);
take(S2);
sh1 = f2(sh1,x);
free(S1);
take(S2);
x = f3(sh1,sh2);
```

Process 2:

```
y = g1(sh1,sh2);
take(S1);
free(S2);
sh2 = g2(y,sh2);
take(S1);
free(S2);
y = g3(sh1,sh2);
```

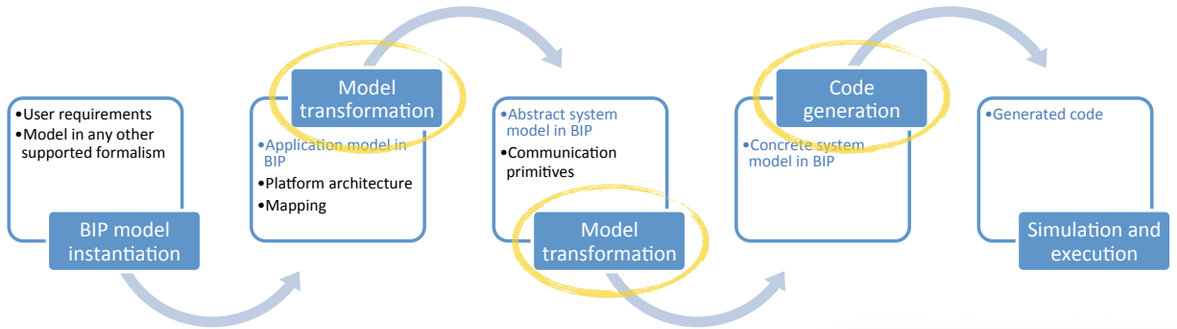
Coordination mechanisms mix up with computation and do not scale.
Code maintenance is a nightmare!

Objectives

Correct-by-construction concurrent systems

Separation of computation from coordination

Rigorous System Design flow

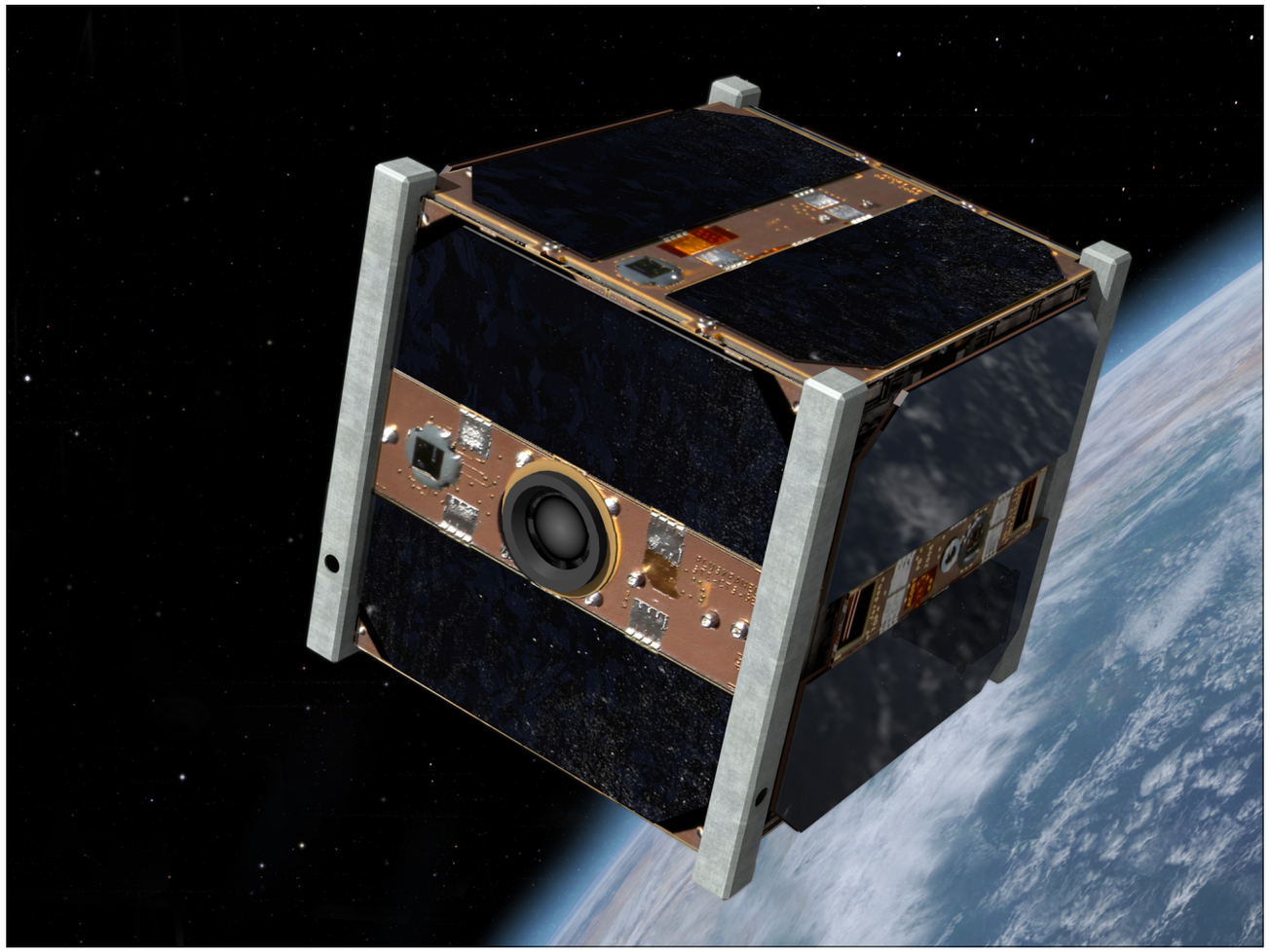


A series of semantics-preserving transformations

Correctness decomposed into
correctness of transformations
correctness of high-level models

Final implementation is **correct by construction**

- ☐ Unifying modelling framework
- ☐ Operational semantics
- ☐ Method(s) to design correct models



Satellite software design

A collaboration with the EPFL Space Engineering Center

Component-based design in BIP of the control software for a nano-satellite

Control and Data Management System (CDMS)

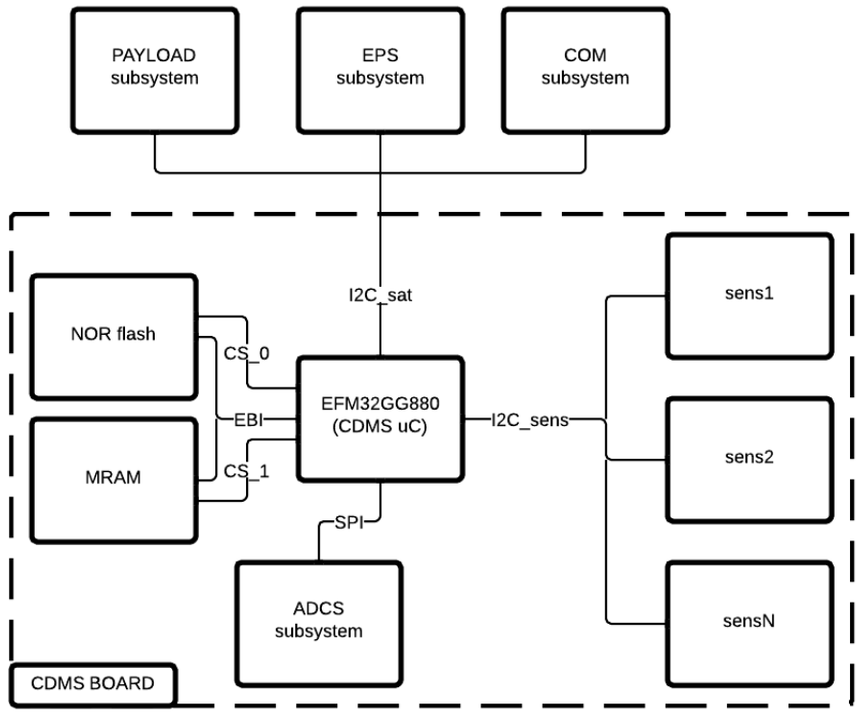
BIP = Behaviour-Interaction-Priority

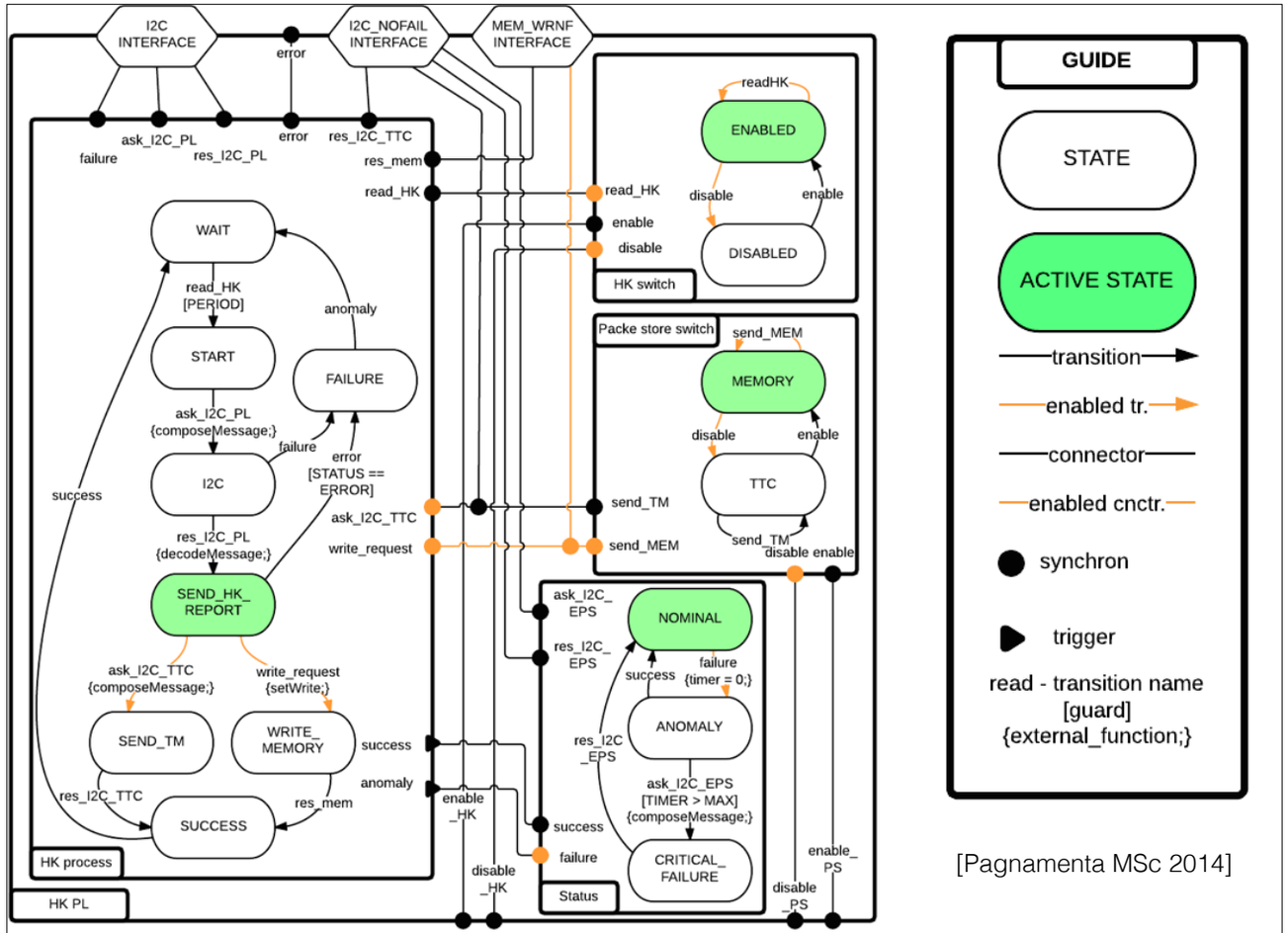
A collaboration with ThalesAlenia Space (France) and Aristotle University of Thessaloniki (Greece)

"Catalogue of System and Software Properties"

Funded by ESA

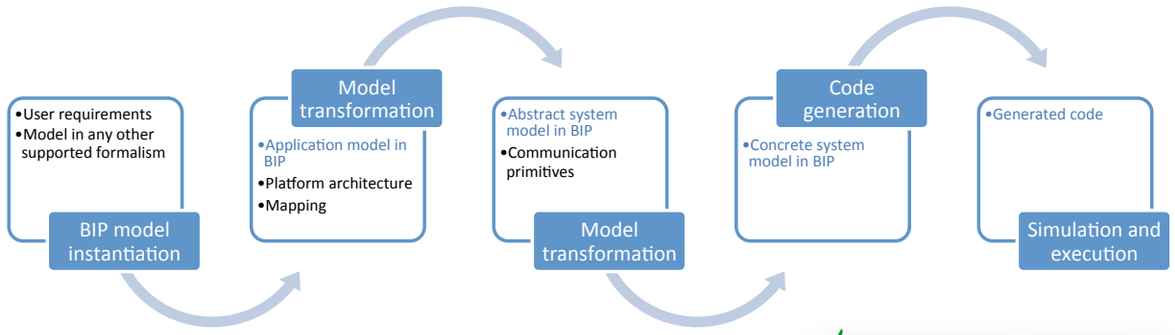
CubETH: CDMS architecture





[Pagnamenta MSc 2014]

Rigorous System Design flow

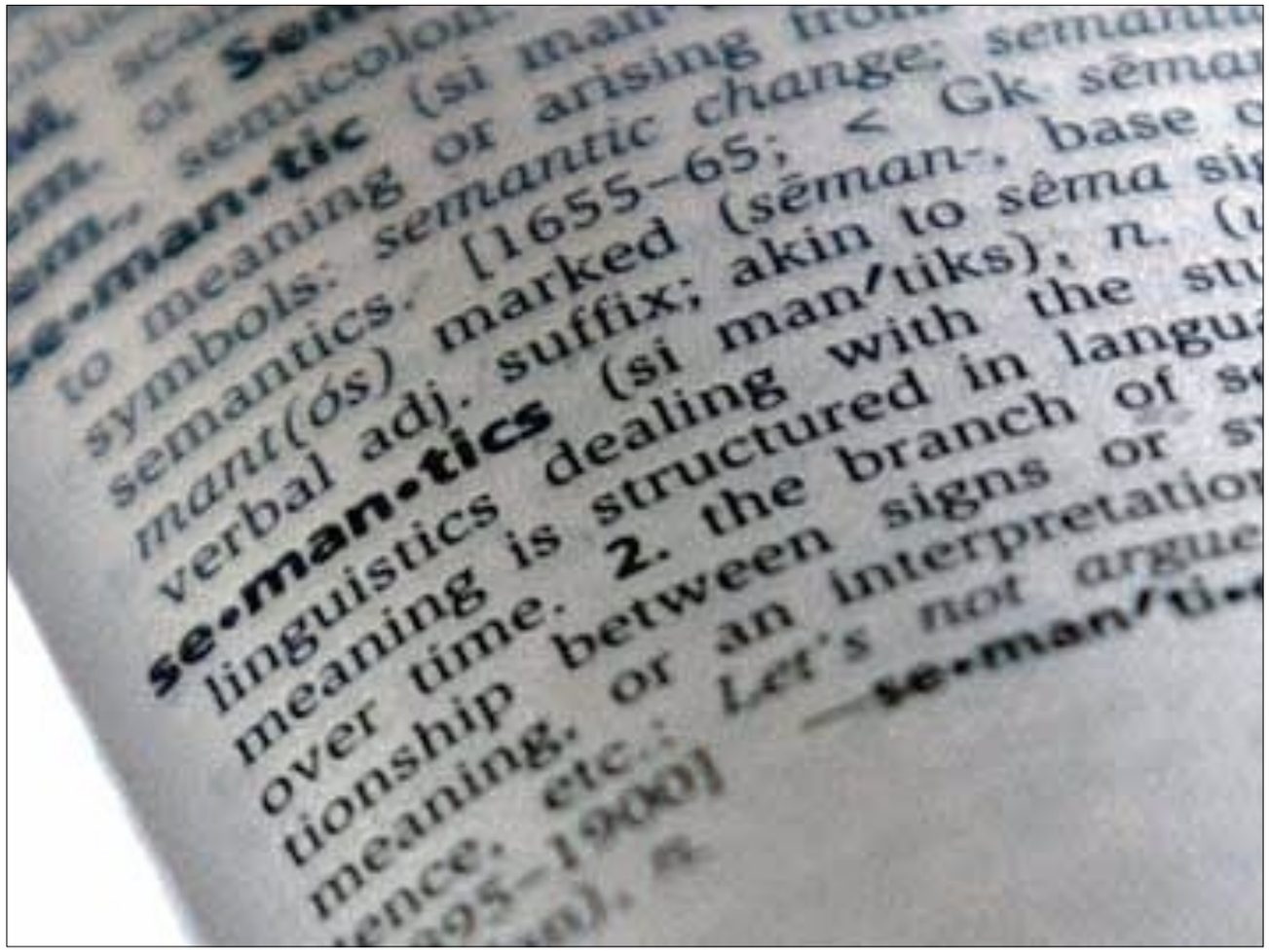


A series of semantics-preserving transformations

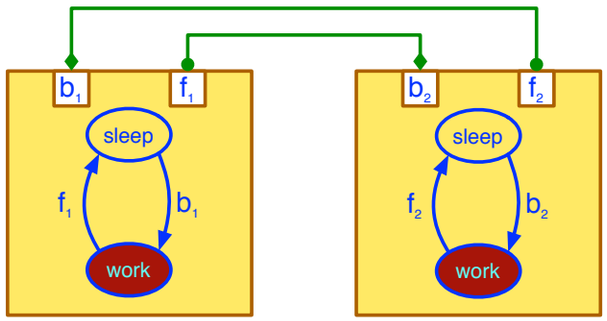
Correctness decomposed into
- correctness of transformations
- correctness of high-level models

Final implementation is **correct by construction**

- ☑ Unifying modelling framework
- ☐ Operational semantics
- ☐ Method(s) to design correct models


se·man·tic
(si man
meaning or arising
to meaning;
symbols; semantic change;
semantics. [1655–65; < Gk sēman
mant(ós) marked (sēman-, base
verbal adj. suffix; akin to sêma
se·man·tics (si man/tiks), n.
linguistics dealing with the
meaning is structured in
over time. 2. the branch
tionship between signs
meaning, or an interpretation
ence, etc.: Let's not
se·man/ti-

BIP by example: Mutual exclusion



Interaction model:

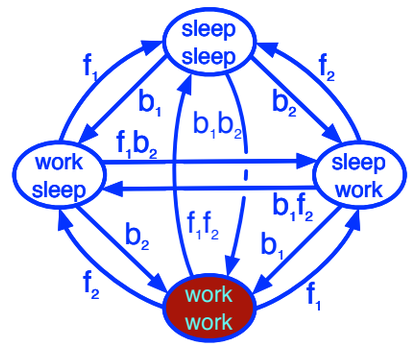
$\{b_1,\ f_1,\ b_2,\ f_2,\ b_1f_2,\ b_2f_1\}$

Maximal progress:

$b_1 < b_1f_2,\ b_2 < b_2f_1$

Design view

Semantic view



[Bliudze, Sifakis, EMSOFT '07]

An interaction specifies what transitions need to happen synchronously

Semantics: Interactions

$$B_i = (Q_i, P_i, \to_i), \qquad \to_i \subseteq Q_i \times 2^{P_i} \times Q_i$$

$$\gamma(B_1, \ldots, B_n) = (Q, P, \to) \qquad Q = \prod_{i=1}^{n} Q_i \qquad P = \bigcup\!\!\!\!\!\cdot_{\,i} P_i$$

Interaction model: $\gamma \subseteq 2^P$ — a set of allowed interactions

$$\frac{q_i \xrightarrow{a \cap P_i} q_i' \ (\text{if } a \cap P_i \neq \emptyset) \quad q_i = q_i' \ (\text{if } a \cap P_i = \emptyset)}{q_1 \ldots q_n \xrightarrow{a} q_1' \ldots q_n'}$$

for each $a \in \gamma$.

Semantics: Priority

$$B_i = (Q_i, P_i, \to_i), \qquad \to_i \subseteq Q_i \times 2^{P_i} \times Q_i \qquad P = \bigcup_i^{\cdot} P_i$$

Interaction model: $\gamma \subseteq 2^P$ — a set of allowed interactions

$$\frac{q_i \xrightarrow{a \cap P_i} q_i' \ (\text{if } a \cap P_i \neq \emptyset) \quad q_i = q_i' \ (\text{if } a \cap P_i = \emptyset)}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}$$

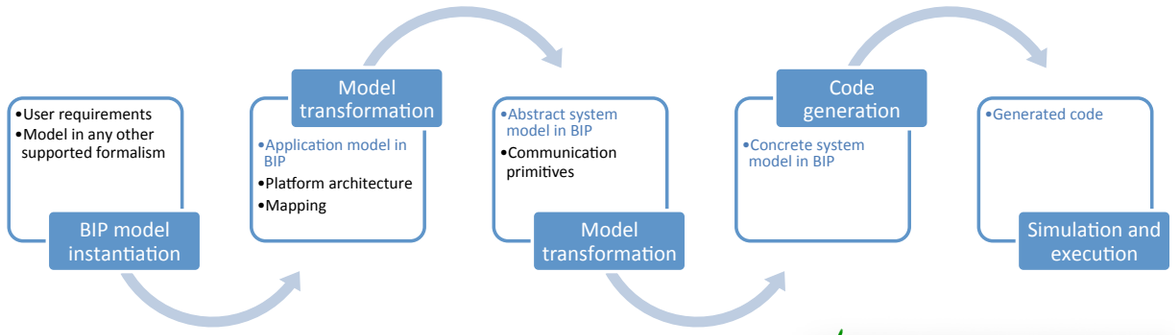
for each $a \in \gamma$.

Priority model: $\prec \subseteq 2^P \times 2^P$ — strict partial order

$$\frac{q \xrightarrow{a} q' \qquad \forall a \prec a', \ q \not\xrightarrow{a'}}{q \xrightarrow{a}_{\prec} q'}$$

for each $a \in 2^P$.

Rigorous System Design flow

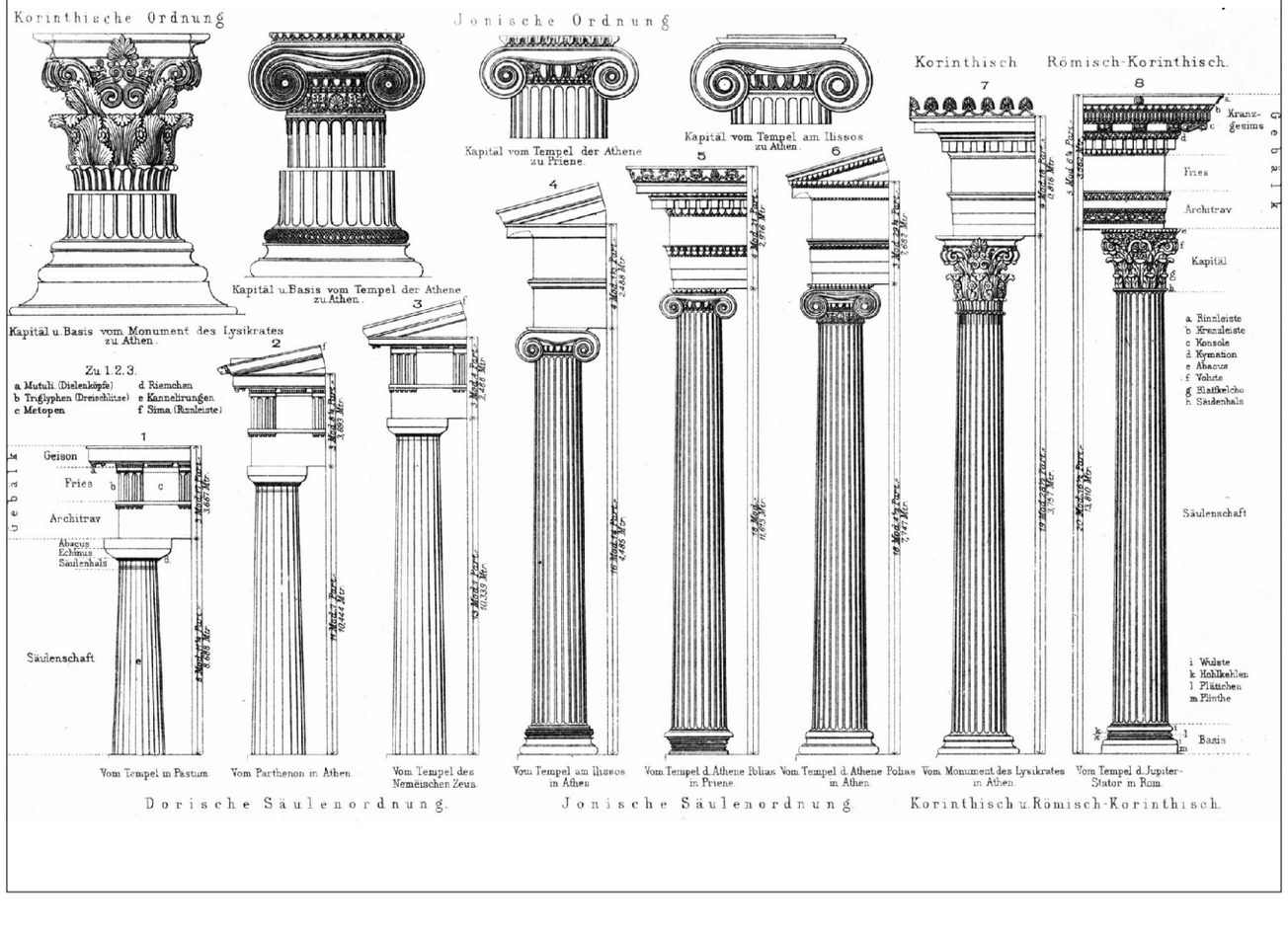


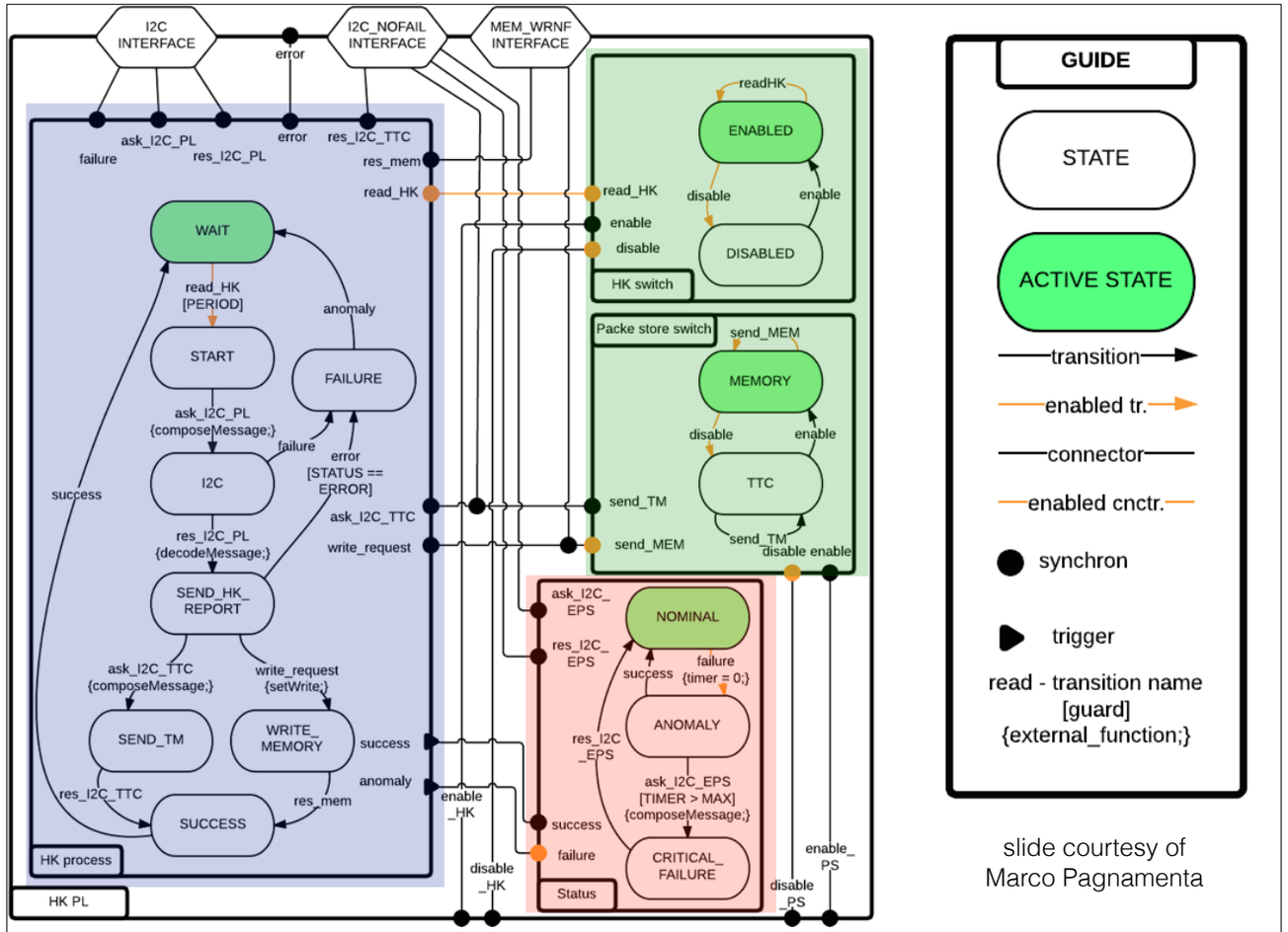
A series of semantics-preserving transformations

Correctness decomposed into
- correctness of transformations
- correctness of high-level models

Final implementation is **correct by construction**

- ☑ Unifying modelling framework
- ☑ Operational semantics
- ☐ Method(s) to design correct models


Korinthische Ordnung
Jonische Ordnung
Korinthisch
Römisch-Korinthisch.
Kapitäl u. Basis vom Monument des Lysikrates zu Athen.
Kapitäl u. Basis vom Tempel der Athene zu Athen.
Kapitäl vom Tempel der Athene zu Priene.
Kapitäl vom Tempel am Ilissos zu Athen.
Zu 1. 2. 3.
a Mutuli (Dielenköpfe)
b Triglyphen (Dreischlitze)
c Metopen
d Riemchen
e Kanneliruugen
f Sima (Rinnleiste)
Gebälk
Geison
Fries
Architrav
Abacus
Echinus
Säulenhals
Säulenschaft
Kapitäl
Gebälk
Fries
Architrav
a Rinnleiste
b Kranzleiste
c Konsole
d Kymation
e Abacus
f Voluta
g Blattkelche
h Saulenhals
Säulenschaft
i Wulste
k Hohlkehlen
l Plättchen
m Plinthe
Basis
Vom Tempel in Pastum
Vom Parthenon in Athen
Vom Tempel des Nemëischen Zeus
Vom Tempel am Ilissos in Athen
Vom Tempel d. Athene Polias in Priene
Vom Tempel d. Athene Polias in Athen
Vom Monument des Lysikrates in Athen
Vom Tempel d. Jupiter-Stator in Rom
Dorische Säulenordnung.
Jonische Säulenordnung.
Korinthisch u. Römisch-Korinthisch.



slide courtesy of
Marco Pagnamenta

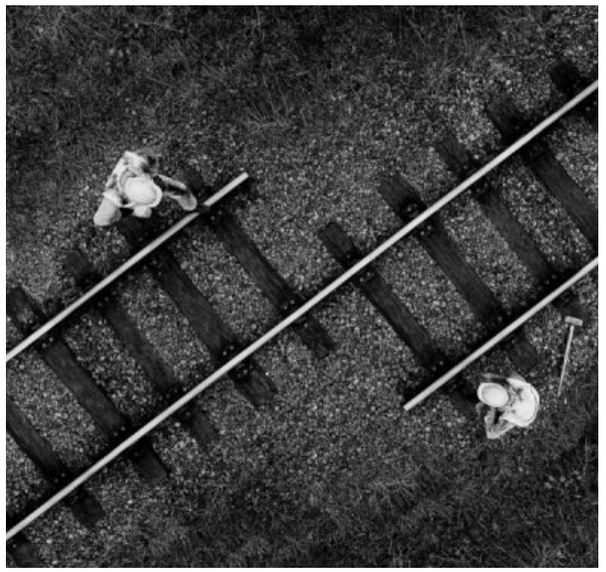
Theory of architectures

Design patterns for BIP

How to model?

How to combine?

How to specify?



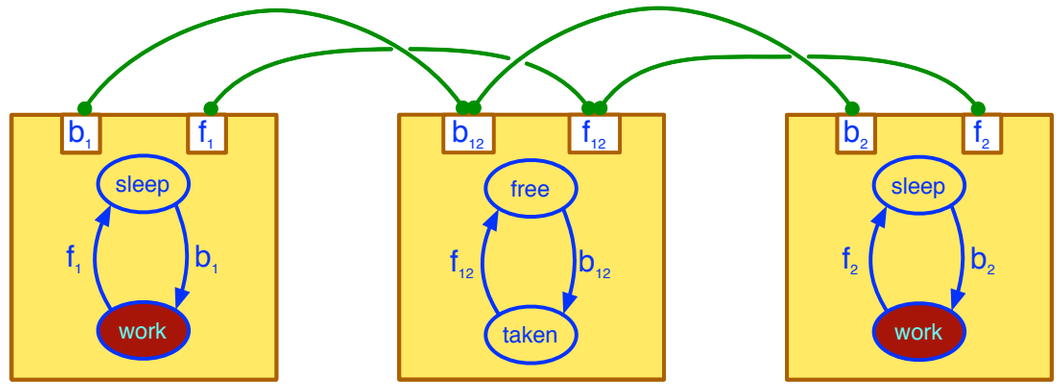
Architectures enforce characteristic properties. The crucial question is whether these are preserved by composition?

[Attie et al, SEFM '14]

We need theory for combining basic architectures and their characteristic properties to obtain an architecture meeting a given global property

How to model?

Example: Lock



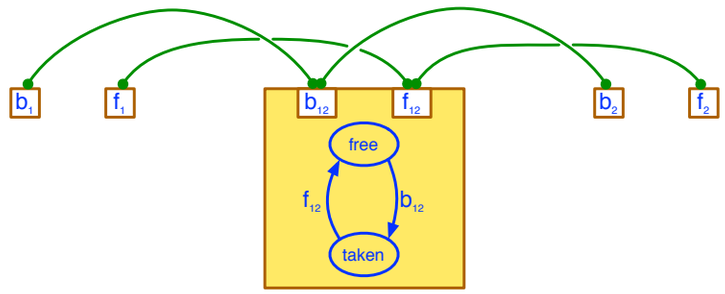
An architecture is...

$$A = (\mathcal{C}, P_A, \gamma)$$

Set of coordinating behaviours

Interface (ports)

Interaction model



...an operator...

$$A = (\mathcal{C}, P_A, \gamma)$$

...transforming

a set of components $\mathcal{B}$

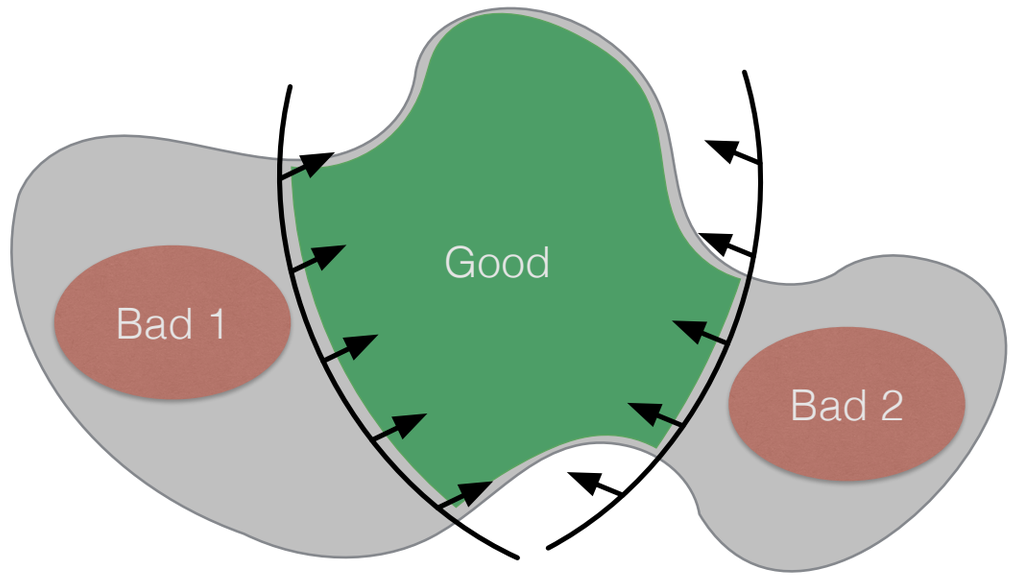
into a composed BIP system $A(\mathcal{B}) \stackrel{def}{=} \Big(\gamma \ltimes P\Big)(\mathcal{B} \cup \mathcal{C})$

where $P \stackrel{def}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B, \quad \gamma \ltimes P \stackrel{def}{=} \left\{ a \subseteq 2^P \,\middle|\, a \cap P_A \in \gamma \right\}$

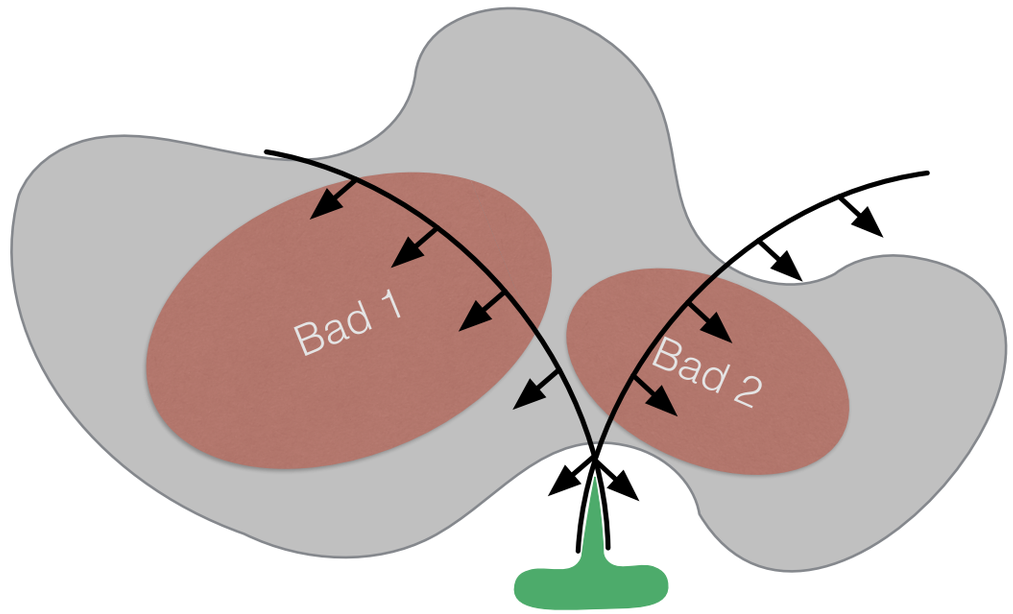
Partial application is not in the paper, but it is in the Tech Report

How to combine?

Constraints intuition



Limits of white magic



Formally

$$A_1 \oplus A_2 \stackrel{def}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma)$$

$$\begin{aligned}\gamma &\stackrel{def}{=} \left\{ a \subseteq 2^P \,\middle|\, a \cap P_1 \in \gamma_1 \ \wedge \ a \cap P_2 \in \gamma_2 \right\} \\ &= (\gamma_1 \ltimes P) \cap (\gamma_2 \ltimes P)\end{aligned}$$

Architecture composition is commutative and associative.

If coordinators are deterministic, it is also idempotent.

Main results: Safety

$$\left.\begin{array}{l} A_1(\mathcal{B}) \models \Phi_1 \\ A_2(\mathcal{B}) \models \Phi_2 \end{array}\right\} \implies \big(A_1 \oplus A_2\big)(\mathcal{B}) \models \Phi_1 \wedge \Phi_2$$

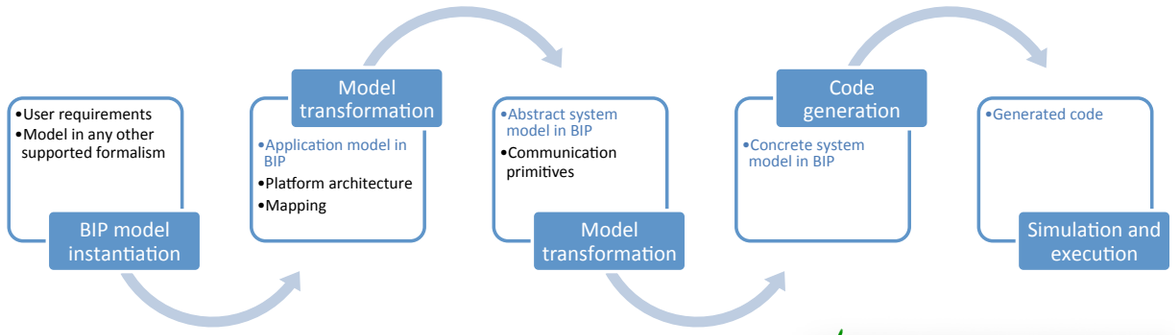
Safety = *"Bad states never occur"*

Main results: Liveness

$$\underbrace{\mathcal{A} \quad \left.\begin{array}{r}\text{live}\\ \text{pairwise non-interfering}\end{array}\right\}}_{\text{w.r.t. } \mathcal{B}} \implies \bigoplus \mathcal{A} \quad \text{live}$$

Liveness = *"Good states occur infinitely often"*

Rigorous System Design flow

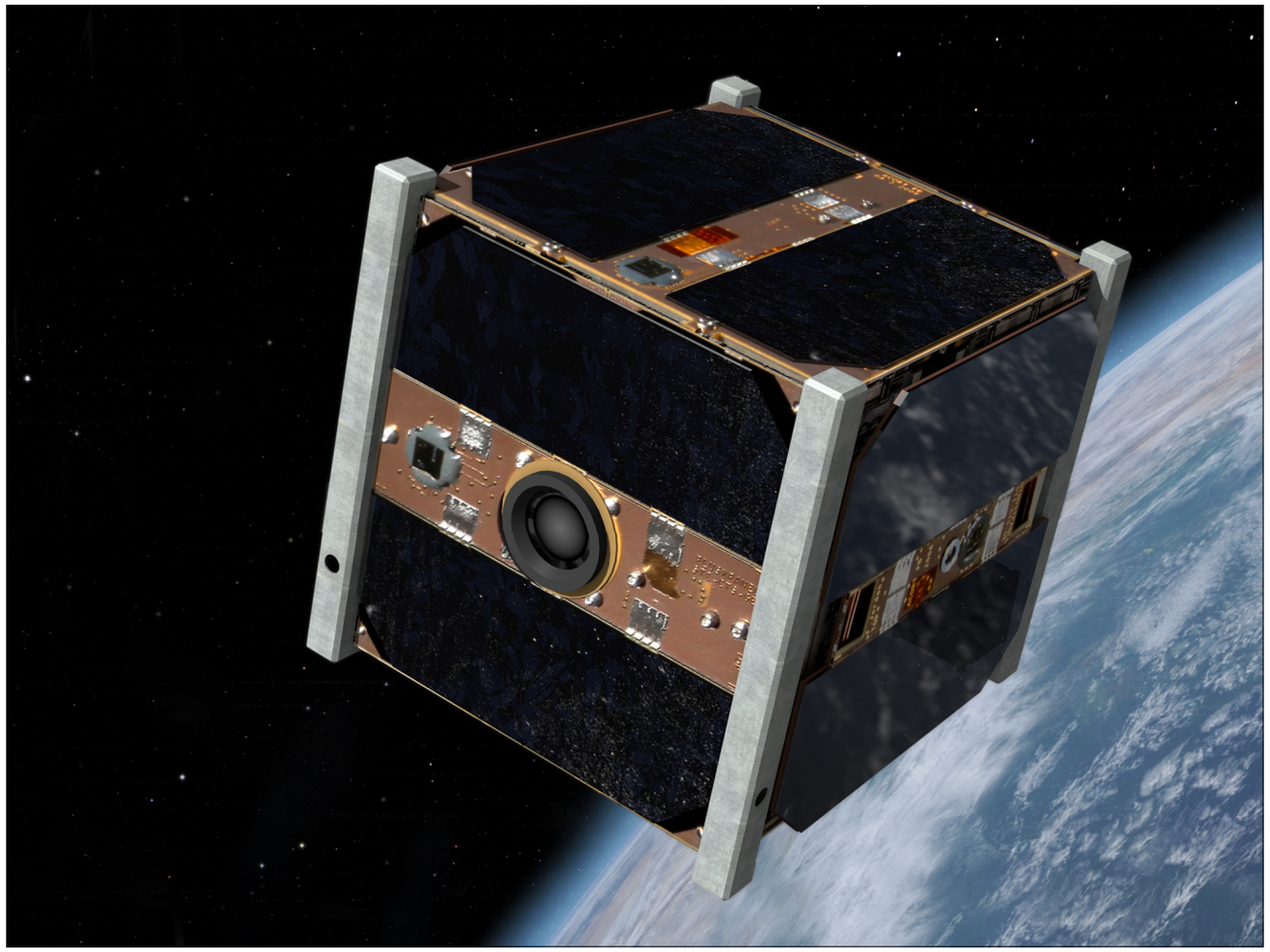


A series of semantics-preserving transformations

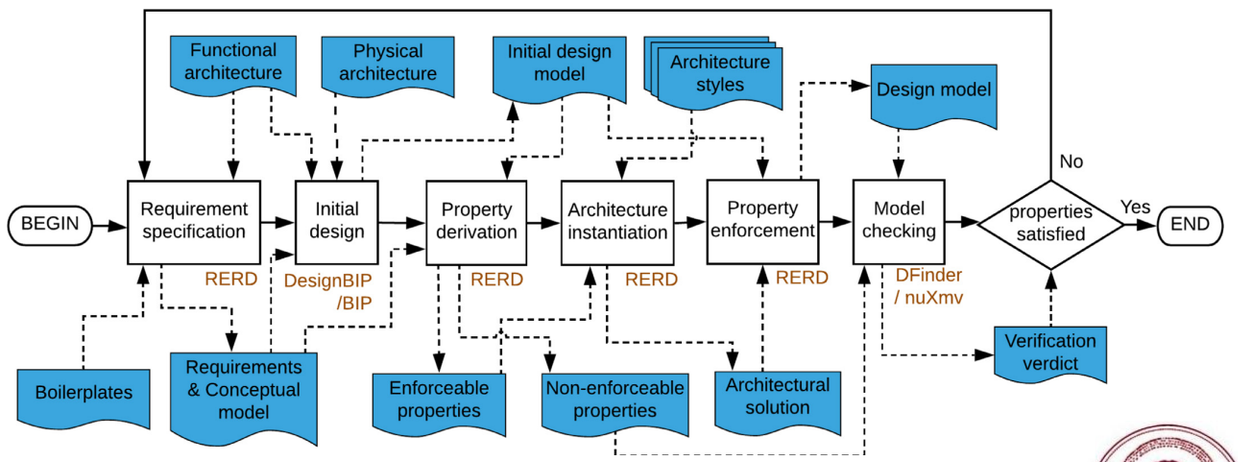
Correctness decomposed into
- correctness of transformations
- correctness of high-level models

Final implementation is **correct by construction**

- ☑ Unifying modelling framework
- ☑ Operational semantics
- ☑ Method(s) to design correct models



Requirements and design process



[Stachtiari et al, JSS '18]

ARISTOTLE UNIVERSITY OF THESSALONIKI

CubETH case study

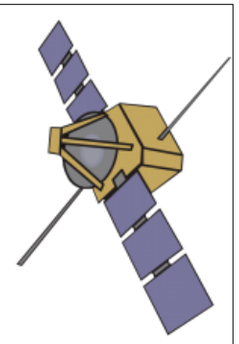


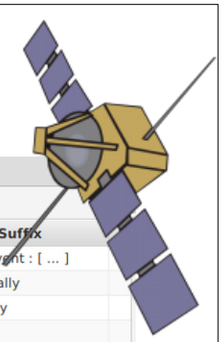
Table 1: Representative requirements for CDMS status and HK_PL

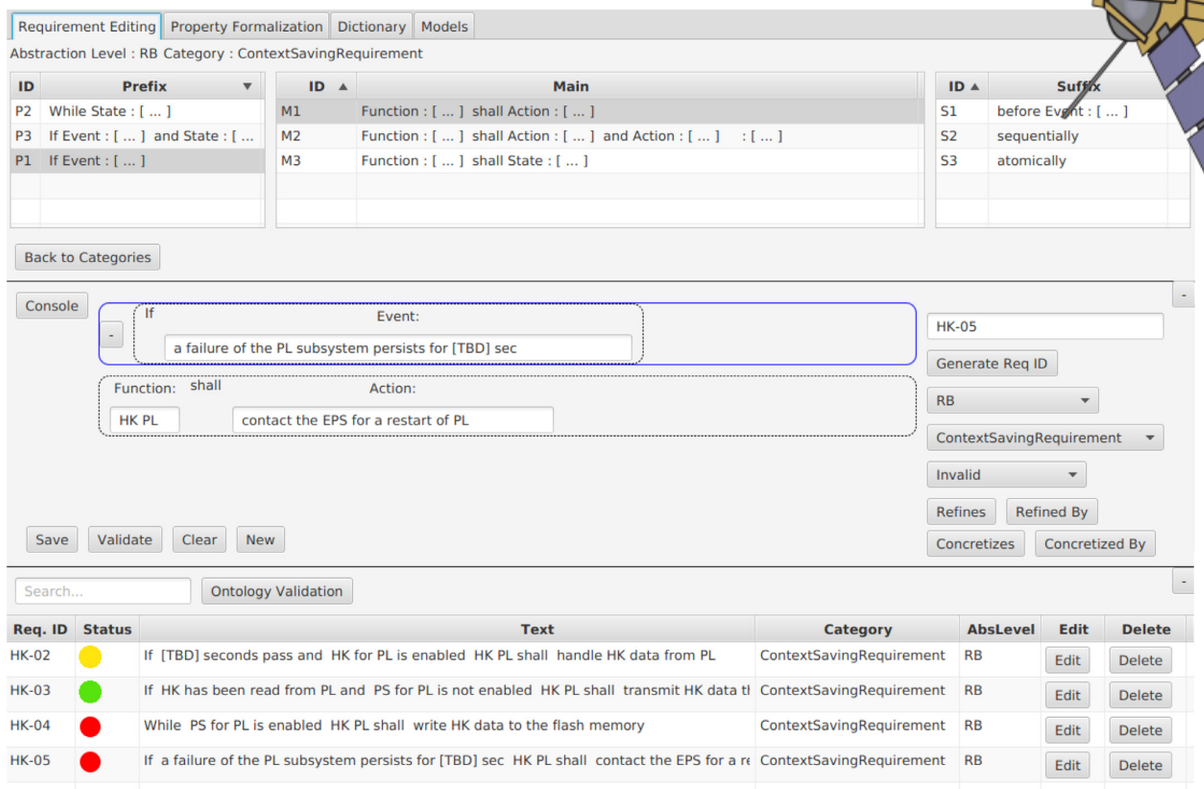
ID	Description
CDMS-007	The CDMS shall periodically reset both the internal and external watchdogs and contact the EPS subsystem with a “heartbeat”.
HK-001	The CDMS shall have a Housekeeping activity dedicated to each subsystem.
HK-003	When line-of-sight communication is possible, housekeeping information shall be transmitted through the COM subsystem.
HK-004	When line-of-sight communication is not possible, housekeeping information shall be written to the non-volatile flash memory.
HK-005	A Housekeeping subsystem shall have the following states: NOMINAL, ANOMALY and CRITICAL_FAILURE.

[Mavridou et al, FACS '16]

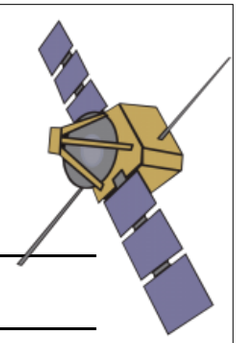
39 requirements in all — see the CubETH tech report for the (almost) complete list

RERD tool




Requirement Editing
Property Formalization
Dictionary
Models
Abstraction Level : RB Category : ContextSavingRequirement
ID
Prefix
P2 While State : [...]
P3 If Event : [...] and State : [...
P1 If Event : [...]
ID
Main
M1 Function : [...] shall Action : [...]
M2 Function : [...] shall Action : [...] and Action : [...] : [...]
M3 Function : [...] shall State : [...]
ID
Suffix
S1 before Event : [...]
S2 sequentially
S3 atomically
Back to Categories
Console
If
Event:
a failure of the PL subsystem persists for [TBD] sec
Function:
shall
Action:
HK PL
contact the EPS for a restart of PL
HK-05
Generate Req ID
RB
ContextSavingRequirement
Invalid
Refines
Refined By
Concretizes
Concretized By
Save
Validate
Clear
New
Search...
Ontology Validation
Req. ID
Status
Text
Category
AbsLevel
Edit
Delete
HK-02 If [TBD] seconds pass and HK for PL is enabled HK PL shall handle HK data from PL ContextSavingRequirement RB Edit Delete
HK-03 If HK has been read from PL and PS for PL is not enabled HK PL shall transmit HK data tl ContextSavingRequirement RB Edit Delete
HK-04 While PS for PL is enabled HK PL shall write HK data to the flash memory ContextSavingRequirement RB Edit Delete
HK-05 If a failure of the PL subsystem persists for [TBD] sec HK PL shall contact the EPS for a re ContextSavingRequirement RB Edit Delete

CubETH case study

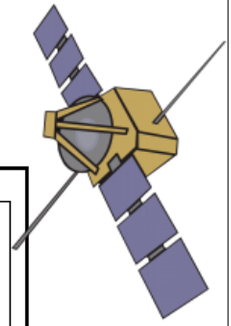


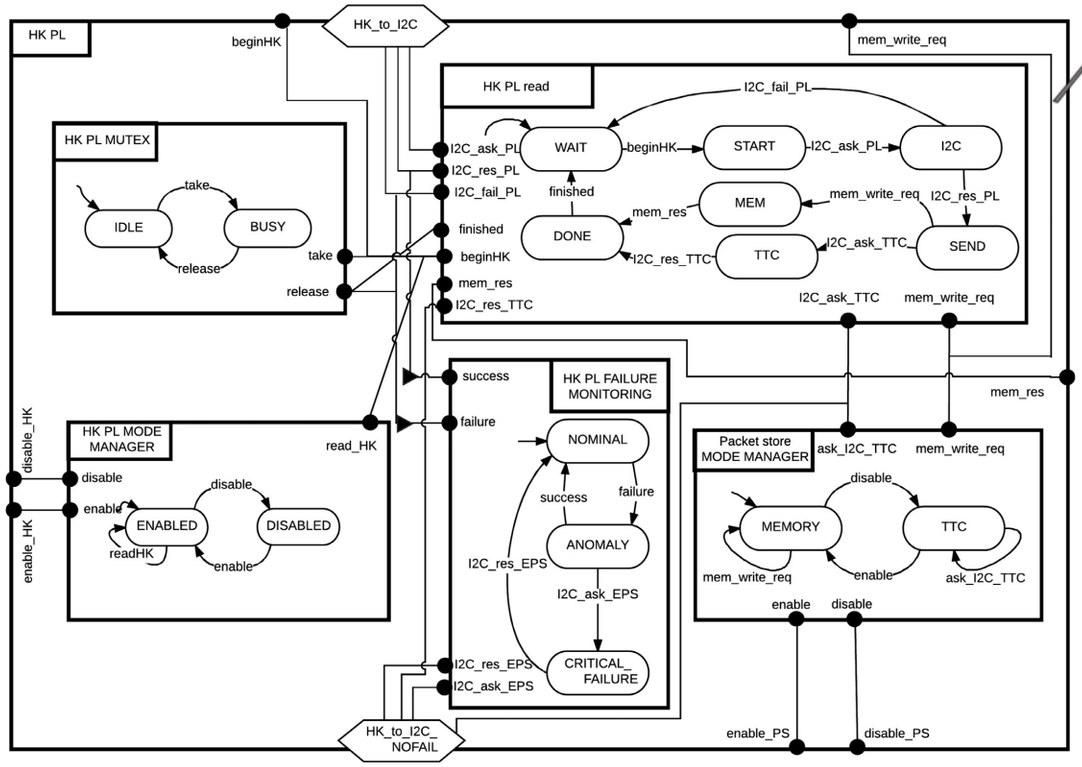
Requirements for the *HK PL* function.

ID	Requirement
HK-02	P2: if ⟨event-e003: [TBD] sec pass ⟩ and ⟨state-s003: HK collection is enabled for PL ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a004: handle HK data from the PL ⟩
HK-03	P3: if ⟨state-s002: PS[a] for PL is not enabled ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a002: transmit HK data through the TC/TM service ⟩
HK-04	P3: while ⟨state-s001: PS for PL is enabled ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a001: write HK data to the flash memory ⟩
HK-05	P1: if ⟨event-e004: a PL failure persists for [TBD] sec ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a003: contact the EPS for a restart of the PL ⟩

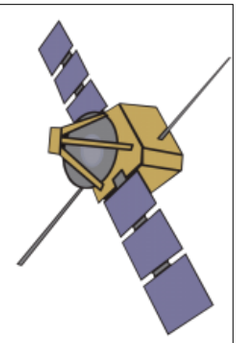
[a] PS stands for a packet store structure.

CubETH case study





CubETH case study



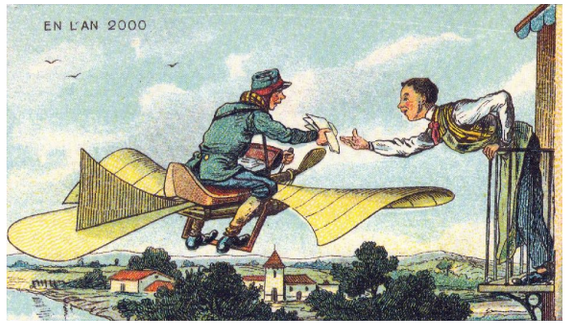
Durations and input sizes of the process steps.

Step	Duration	Input size
Requirement specification	8 h	38 requirements
Initial design	5 h	12 components
Architecture instantiation	3 h	47 enforced properties
Verification of deadlock freedom	12 s	46 components

Statistics of requirement formulation and property enforcement.

Model	Flow	Mode	Event	Mutex	Failure	Requir.	Deriv. Prop.	Assum. Prop.	Enforced
Payload	0	2	0	4	0	12	16	0	16
HK PL	0	2	1	1	1	4	6	0	6
HK EPS	0	2	1	1	1	4	6	0	6
HK COM	0	2	1	1	1	4	6	0	6
HK CDMS	0	2	1	1	0	3	4	0	4
Flash memory	0	1	0	1	0	8	13	4	3
CDMS status	1	0	0	0	0	1	3	0	3
Error logging	0	0	1	1	0	2	3	0	3
Total	1	11	5	10	3	38	57	4	47

Future work



BIP

Dynamicity, distribution, self-adaptation

Architectures

Case studies, case studies, case studies and taxonomies (libraries)

Real-time, Synthesis, Dynamicity

DSLs for usability

Verification and proof of architectures and architecture styles

Tool support

General purpose software

JavaBIP...

JavaBIP

Connectors

```
1 <require>
2   <effect id="give" specType="TCPReader"/>
3     <causes>
4       <port id="put" specType="CommandBuffer"/>
5   </causes>
6 </require>
7 <accept>
8   <effect id="give" specType="TCPReader"/>
9    <causes>
10      <port id="put" specType="CommandBuffer"/>
11   </causes>
12 </accept>
```

```
1 @Ports({
2   @Port(name="put", type=PortType.enforceabl
3   @Port(name="get", type=PortType.enforceabl
4 })
5
6 @ComponentType(initial="0", name="CommandBuf
7 public class CommandBuffer {
8   private LinkedList<Command> commandList;
9   private int bufferSize;
10
11   public CommandBuffer(int bufferSize){
12     this.bufferSize = bufferSize;
13     this.commandList = new LinkedList<Comman
14   }
15
16   @Transition(name="get", source="0", target="0", guard="notEmpty")
17   public void get() {
18     commandList.remove();
19   }
20
21   @Guard(name="notEmpty")
22   public boolean notEmpty() { return !commandList.isEmpty(); }
23
24   @Transition(name="put", source="0", target="0", guard="notFull")
25   public void put(@Data(name="input") Command cmd) {
26     commandList.add(cmd);
27   }
28
29   @Guard(name="notFull")
30   public boolean notFull() { return commandList.size() < bufferSize; }
31
32   @Data(name="command")
33   public Command getNextCommand() { return commandList.get(0); }
34 }
```

Transitions

States

Data

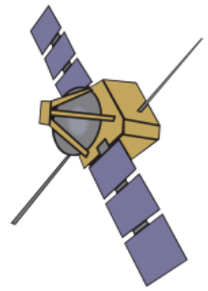
[Bliudze et al, SPE '17]

Conclusion

Powerful theoretical tools to build systems that are correct by construction

Going from theory to practice requires a lot of effort and cross-domain collaborations

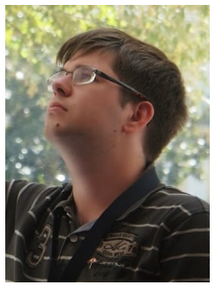
Bigger challenge yet: taking these methods to less constrained application domains

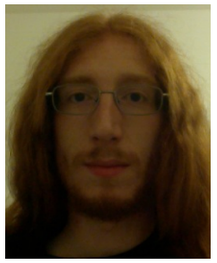


Thanks!



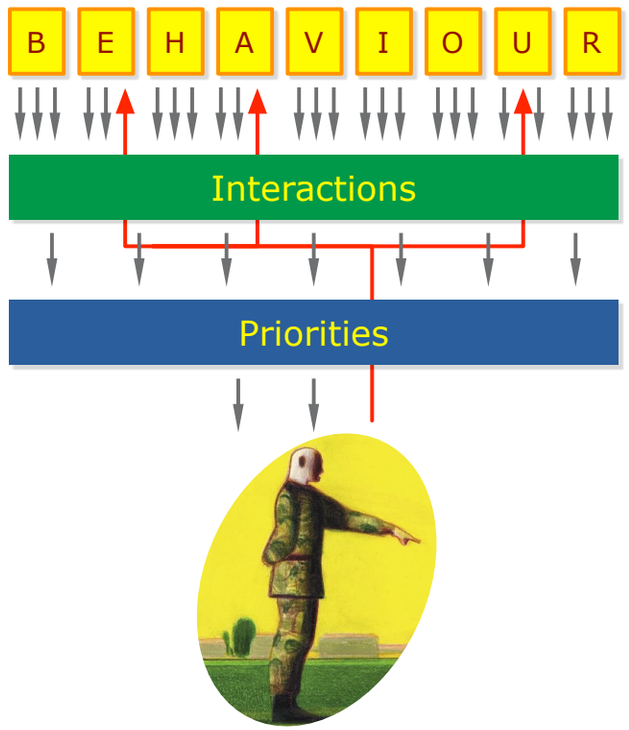





Appendices

Engine-based execution



1. Components notify the Engine about enabled transitions.

2. The Engine picks an interaction and instructs the components about next actions to take.

Various Engine implementations:

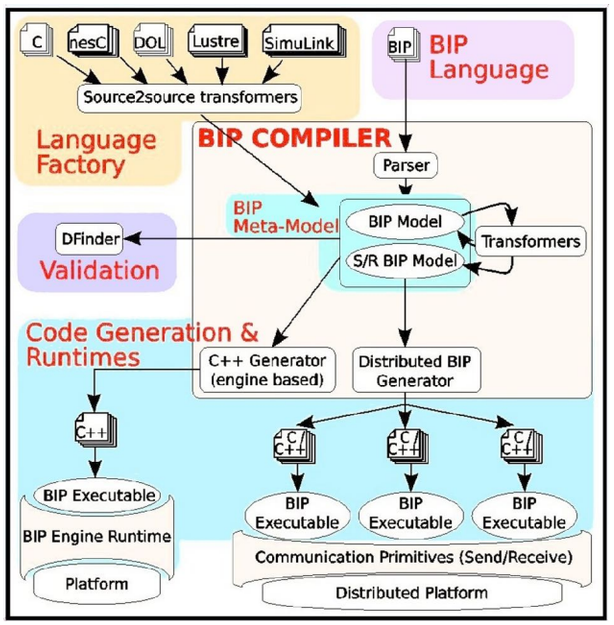
Centralised (enumerative & symbolic)

Distributed (two protocol layers)

Real-Time (timed automata)

Dynamic (evolving topology)

Core BIP tool-set @ Verimag



Ecore meta-model

Model-checking tools

C++ code generation

Main idea

Characteristic predicate for $\gamma \subseteq 2^P$

$$\varphi_\gamma : \mathbb{B}^P \to \mathbb{B} \qquad \varphi_\gamma \triangleq \bigvee_{a \in \gamma} \left(\bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p} \right)$$

Interaction models to predicates and back

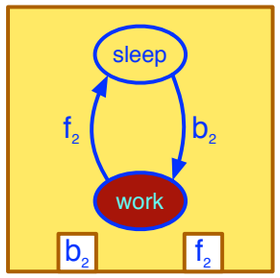
$$v : P \to \mathbb{B}, \qquad \varphi(v) = \mathtt{tt} \iff \{p \in P \mid v(p) = \mathtt{tt}\} \in \gamma$$

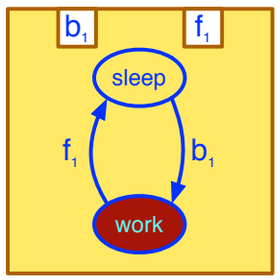
$$A_1 \oplus A_2 \stackrel{def}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma_\varphi) \qquad \varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$$

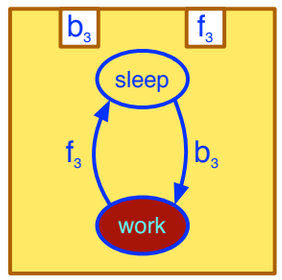
Architecture composition is commutative and associative.

If coordinators are deterministic, it is also idempotent.

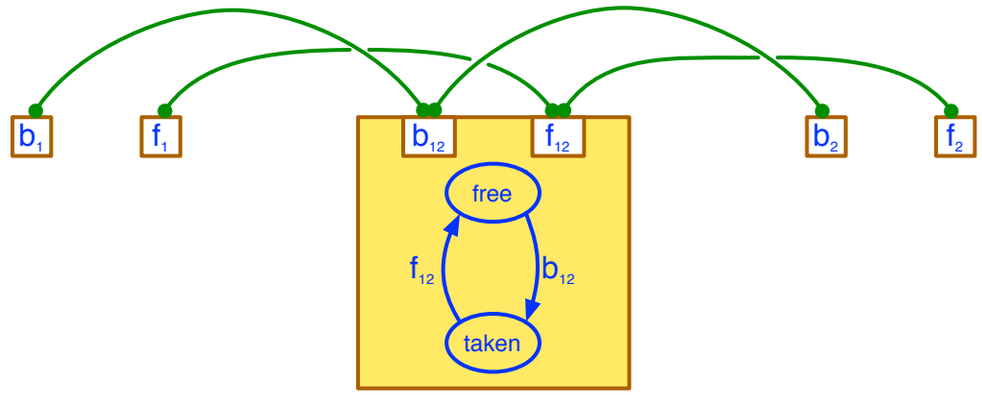
Example continued

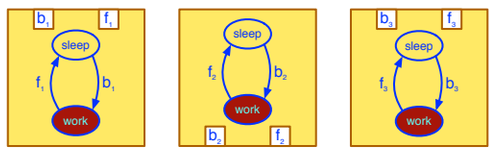






Example continued



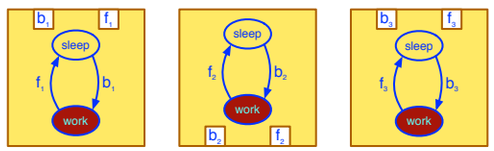


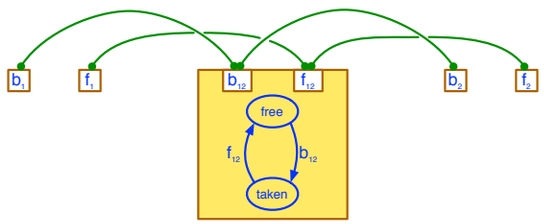
Example continued

$$\varphi_{\gamma_{12}} \equiv (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \wedge (b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})$$

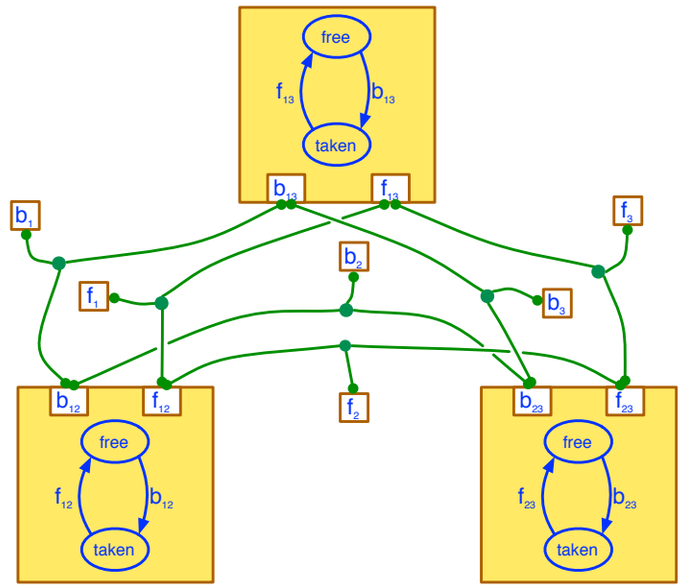
$$\begin{array}{lllll} b_1 \Rightarrow b_{12} \wedge b_{13}\,, & f_1 \Rightarrow f_{12} \wedge f_{13}\,, & b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2\,, & f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2\,, & b_{12} \Rightarrow \overline{f_{12}} \\ b_2 \Rightarrow b_{12} \wedge b_{23}\,, & f_2 \Rightarrow f_{12} \wedge f_{23}\,, & b_{13} \Rightarrow b_1 \,\texttt{XOR}\, b_3\,, & f_{13} \Rightarrow f_1 \,\texttt{XOR}\, f_3\,, & b_{13} \Rightarrow \overline{f_{13}} \\ b_3 \Rightarrow b_{13} \wedge b_{23}\,, & f_3 \Rightarrow f_{13} \wedge f_{23}\,, & b_{23} \Rightarrow b_2 \,\texttt{XOR}\, b_3\,, & f_{23} \Rightarrow f_2 \,\texttt{XOR}\, f_3\,, & b_{23} \Rightarrow \overline{f_{23}} \end{array}$$

$$\{\emptyset\,,\ b_1 b_{12} b_{13}\,,\ f_1 f_{12} f_{13}\,,\ b_2 b_{12} b_{23}\,,\ f_2 f_{12} f_{23}\,,\ b_3 b_{13} b_{23}\,,\ f_3 f_{13} f_{23}\}$$





Example continued



$$\{\emptyset\,,\ b_1b_{12}b_{13}\,,\ f_1f_{12}f_{13}\,,\ b_2b_{12}b_{23}\,,\ f_2f_{12}f_{23}\,,\ b_3b_{13}b_{23}\,,\ f_3f_{13}f_{23}\}$$

Enforcing properties (safety)

Consider a behaviour $B = (Q, q^0, P, \rightarrow)$

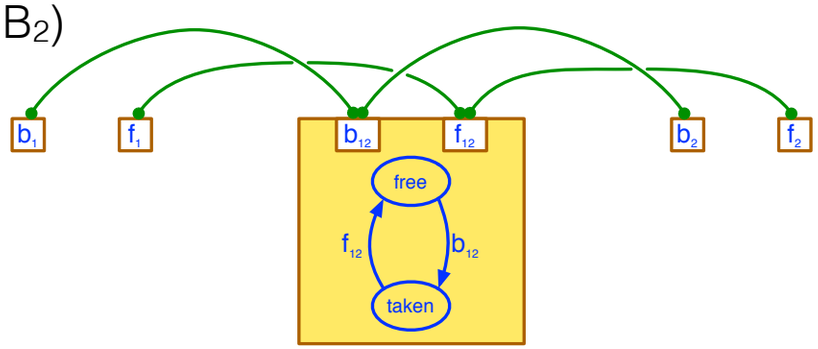
A property: $\Phi \subseteq Q$

An architecture A imposes a property Φ on $\mathcal{B}$ if Φ contains the projection of all the reachable states of $A(\mathcal{B})$ onto $\mathcal{B}$

$$A(\mathcal{B}) \models \Phi$$

Properties

MUX (B_1, B_2)



Assumed:

AG (f_1 => **A** [$state_1$ ≠ work **W** b_1])

AG (f_2 => **A** [$state_2$ ≠ work **W** b_2])

Characteristic:

AG ($state_1$ ≠ work v $state_2$ ≠ work)

Nice properties

Under suitable conditions

Architectures can be composed before applying

$$A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B})$$

Architecture application can be restricted

$$A_2(A_1(\mathcal{B}_1, \mathcal{B}_2)) = A_2(A_1(\mathcal{B}_1), \mathcal{B}_2)$$

Architecture can be applied partially

$$A(\mathcal{B}_1, \mathcal{B}_2) = A[\mathcal{B}_1](\mathcal{B}_2)$$

BIP coordination for Java…

SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. (2017)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2495

Exogenous coordination of concurrent software components with JavaBIP

Simon Bliudze[1,*,†], Anastasia Mavridou[2], Radoslaw Szymanek[3] and Alina Zolotukhina[1]

[1]*Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland*
[2]*Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA*
[3]*Crossing-Tech S.A., EPFL Innovation Park, 1015 Lausanne, Switzerland*

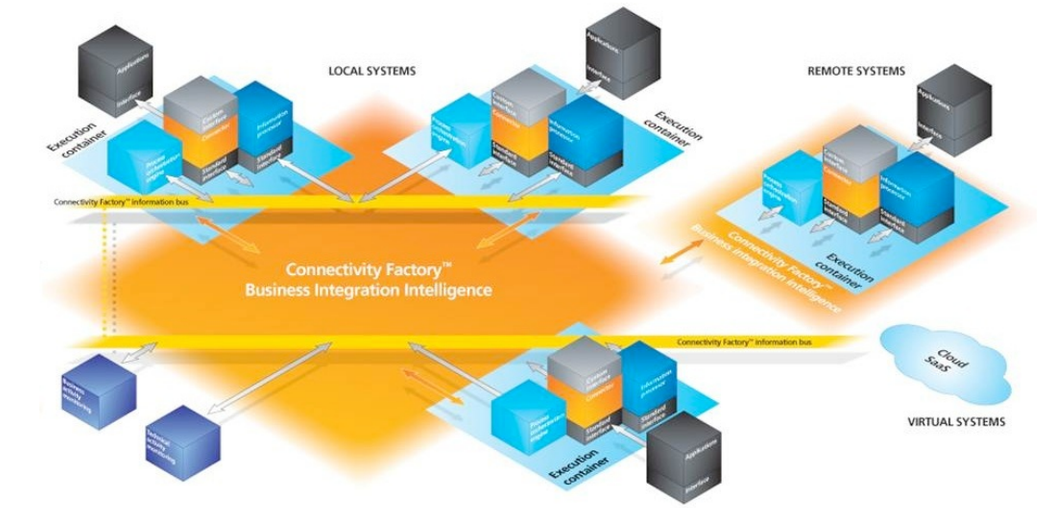
SUMMARY

A strong separation of concerns is necessary in order to make the design of domain-specific functional components independent from cross-cutting concerns, such as concurrent access to the shared resources of the execution platform. Native coordination mechanisms, such as locks and monitors, allow developers to address these issues. However, such solutions are not modular; they are complex to design, debug, and main-

Entities/components code that needs coordination often can not be changed (even by AOP).

Entities or the existing managers of those entities provide API to control those entities.

Use case: Camel Routes



Many independent routes share memory

We have to control the memory usage

e.g., by limiting to only a safe number of routes simultaneously

Lack of coordination can easily result in fatal errors, like JVM running out of memory, or bad behaviour like too large

JavaBIP

```
 1 @Ports({
 2  @Port(name = "add", type = Port
 3   @Port(name = "rm", type = Port
 4 })
 6 @ComponentType(initial = "on", r
 7 public class MemoryMonitor {
 8
 9   final private int memoryLimit;
10   private int currentCapacity =
11
12   public MemoryMonitor(int memor
13       this.memoryLimit = memoryL
14       }
15
16   @Transition(name = "add", sour
17               guard = "hasCapaci
18   public void addRoute(@Data("me
19     currentCapacity += deltaMemo
20   }
21
22   @Transition(name = "rm", source = "on", target = "on", guard = "")
23   public void removeRoute(@Data(name="memoryUsage") Integer deltaMemory) {
24     currentCapacity -= deltaMemory;
25   }
26
27   @Guard(name = "hasCapacity")
28   public boolean hasCapacity(@Data("memoryUsage") Integer memoryUsage) {
29     return currentCapacity + memoryUsage < memoryLimit;
30   }
31 }
```