Revisiting the Glue of BIP

Jacques Combaz

Verimag

MeTRiD 2019
Component-Based Design

- Complex systems built by assembling **components**:
  - digital circuits from gates
  - mechanical object from smaller parts (car, ...)
  - backhoe loader from Lego blocks
  - ETAPS from conferences, workshops, tutorials, ...
  - ...

- Advantages:
  - managing complexity: divide-and-conquer
  - modularity: replace / upgrade components
  - reuse of existing components.
Assembling Components

- Components **interfaces**
  - assembly options

- Systems:
  - what are the components?
  - how are they assembled (architecture)?
“Glues” assemble *software components* based on their interfaces:

- coordination / synchronization: joint actions, barriers, precedence, . . .
- communication: exchange of data
- hierarchy: construction of software components from simpler ones.
Interesting Properties of Glues

Interesting notions for glues:

- **expressivity**
  \[
  \forall \quad \approx \quad \exists
  \]

- **flattening**
  \[
  \rightarrow
  \]

- **compositionality**
  \[
  \models \Phi_1 \quad \models \Phi_2 \quad \models \Phi_3
  \]

- **composability**
  \[
  \models \Phi_1 \models \Phi_2 \models \Phi_1 \land \Phi_2
  \]

- **synthesis**
  \[
  \models \Phi
  \]

Studied in the **BIP** framework (Sifakis et al.):

1. **A Notion of Glue Expressiveness for Component-Based Systems**, CONCUR 2008
1. Components and Glues

2. A Notion of Interactions

3. A Notion of Connectors

4. Conclusion
1 Components and Glues

2 A Notion of Interactions

3 A Notion of Connectors

4 Conclusion
Components = private data (internal state) + interface (set of ports):
- ports convey data (in and out)
- transformation of internal state = execution on a port.
**Components**

**Components and Glues**

**(Atomic) Components**

A component is an LTS $B = (Q, P, \rightarrow)$ having:

- **(internal) states** $Q$ (possibly infinite)
- **an interface** $P$, i.e. finite set of **ports** exposing values in domain $\mathcal{D}$
- **interface state** (at state $q$) given by $x : P \rightarrow (\mathbb{B} \times \mathcal{D})$
- **transitions** $\rightarrow : Q \times P \times \mathcal{D} \times Q$:
  - notation: $q \xrightarrow{p \downarrow y} q'$ for $(q, p, y, q') \in \rightarrow$
  - $y \in \mathcal{D}$: value received by $B$ through $p$
  - $p$ is **enabled** at $q$ iff $q \xrightarrow{p \downarrow y} q'$.

**Assumptions:**

- $x(p) = (e, v)$ at state $q$ if $p$ exposes $v$ and if $p$ enabled $\iff e = \text{true}$
- **completeness**: $\exists y \in \mathcal{D} . q \xrightarrow{p \downarrow y} q' \Rightarrow \forall y' \in \mathcal{D} . \exists q'' \in Q . q \xrightarrow{p \downarrow y'} q''$.  

Jacques Combaz

Revisiting the Glue of BIP

MeTRiD 2019
(Atomic) Components

\[ \mathcal{D} = \{1, 2, 3, 4, 5\} \]

Assumptions:
- \( x(p) = (e, v) \) at state \( q \) if \( p \) exposes \( v \) and if \( p \) enabled \( \iff e = \text{true} \)
- \textbf{completeness}: \( \exists y \in \mathcal{D} \cdot q \xrightarrow{p'y} q' \Rightarrow \forall y' \in \mathcal{D} \cdot \exists q'' \in Q \cdot q \xrightarrow{p'y'} q'' \).
(Atomic) Components without Data

Special case: ports without data:

- interface state: \( x : P \to \mathbb{B} \)
- transitions are of the form \( q \xrightarrow{p} q' \).
Example of (Atomic) Component: Aircraft Flaps

port type Port()
atom type Flaps()
  export port Port extend(), retract()
place EXTENDED, RETRACTED
  initial to EXTENDED
  on retract from EXTENDED to RETRACTED
  on extend from RETRACTED to EXTENDED
end

extend retract
Flaps
retractextend
RETRACTED
EXTENDED

Jacques Combaz
Composition of Components

(Parallel) Composition of Components \( \{ B_i = (Q_i, P_i, \rightarrow_i) \}_{i=1...n} \)

- States: \((q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n\)
- Execution: subsets of component transitions \( \{ q_i \xrightarrow{p_i \downarrow y_i}_i q_i' \}_{i \in I} \):
  \[
  (q_1, \ldots, q_n) \xrightarrow{\{p_i \downarrow y_i\}_{i \in I}} (q_1', \ldots, q_n')
  \]
  - for \( i \in I \) we have \( q_i \xrightarrow{p_i \downarrow y_i}_i q_i' \) (execution in \( B_i \))
  - for \( i \notin I \) we have \( q_i' = q_i \) (no execution in \( B_i \)).
Composition of Components

(Parallel) Composition of Components \( \{ B_i = (Q_i, P_i, \rightarrow_i) \} \) \( i=1 \ldots n \)

- **States:** \((q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n\)

- **Execution:** partial function \( y : P_1 \cup \ldots \cup P_n \rightarrow D \), \( \text{dom}(y) \neq \emptyset \):
  \[
  (q_1, \ldots, q_n) \xrightarrow{y} (q'_1, \ldots, q'_n)
  \]
  - either \( \text{dom}(y) \cap P_i = \{p_i\} \) and \( q_i \xrightarrow{p_i \downarrow y(p_i)} q'_i \) (execution in \( B_i \))
  - or \( \text{dom}(y) \cap P_i = \emptyset \) and \( q'_i = q_i \) (no execution in \( B_i \)).
(Parallel) Composition of Components \( \{ B_i = (Q_i, P_i, \rightarrow_i) \}_{i=1}^{n} \)

- **States:** \((q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n\)
- **Execution:** partial function \(y : P_1 \cup \ldots \cup P_n \rightarrow D, \ dom(y) \neq \emptyset\):
  \[(q_1, \ldots, q_n) \xrightarrow{y} (q'_1, \ldots, q'_n)\]
  - either \(\text{dom}(y) \cap P_i = \{p_i\}\) and \(q_i \xrightarrow{p_i \downarrow y(p_i)} q'_i\) (execution in \(B_i\))
  - or \(\text{dom}(y) \cap P_i = \emptyset\) and \(q'_i = q_i\) (no execution in \(B_i\)).
Composition of Components without Data

(Parallel) Composition of Components \( \{ B_i = (Q_i, P_i, \rightarrow_i) \} \) \( i = 1 \ldots n \)

- **States:** \( (q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n \)

- **Execution:** subset of ports \( y \subseteq P_1 \cup \ldots \cup P_n, y \neq \emptyset \):
  \[
  (q_1, \ldots, q_n) \downarrow^y (q'_1, \ldots, q'_n)
  \]
  - either \( y \cap P_i = \{p_i\} \) and \( q_i \xrightarrow{p_i} q_i' \) (execution in \( B_i \))
  - or \( y \cap P_i = \emptyset \) and \( q_i' = q_i \) (no execution in \( B_i \)).
Notations (components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1...n}$)

Given a subset of ports $P \subseteq P_1 \cup \ldots \cup P_n$:

- $[P]^\uparrow$: states of interface $P$, i.e. $x : P \rightarrow (\mathbb{B} \times \mathcal{D})$
- $[P]^\downarrow$: executions over $P$, i.e. $y : P \rightarrow \mathcal{D}$ s.t. $\forall i . |\text{dom}(y) \cap P_i| \leq 1$
- $[P]^\downarrow$: executions with domain $P$, i.e. $y \in [P]^\downarrow$ s.t. $\text{dom}(y) = P$. 
Notations without Data

Notations (components \( \{ B_i = (Q_i, P_i, \rightarrow_i) \}_{i=1}^{n} \) without data)

Given a subset of ports \( P \subseteq P_1 \cup \ldots \cup P_n \):

- \( [P]^\uparrow \): states of interface \( P \), i.e. \( x : P \rightarrow \mathbb{B} \)
- \( [P]^\downarrow \): executions over \( P \), i.e. \( y \subseteq P \) s.t. \( \forall i . \ |y \cap P_i| \leq 1 \)
- \( [P]^\downarrow \): executions with domain \( P \), i.e. \( [P]^\downarrow = \{ P \} \) if \( \forall i . \ |P \cap P_i| \leq 1 \), \( \emptyset \) otherwise.
A glue $gl$ restricts executions $\downarrow y$ in a composition, based (only) on interface state $x \in [P]^\uparrow$ of ports $P = P_1 \cup \ldots \cup P_n$: 

$$gl : [P]^\uparrow \rightarrow 2^{[P]^\downarrow}$$

s.t. $\forall x . \forall y \in gl(x) . \forall p \in dom(y) . x(p) \in \{\text{true}\} \times D$ (i.e. all ports executed by $y$ are enabled in $x$).
**Glues**

A glue $gl$ restricts executions $\downarrow y$ in a composition, based (only) on interface state $x \in [P]^\uparrow$ of ports $P = P_1 \cup \ldots \cup P_n$:

$$gl : [P]^\uparrow \rightarrow 2^{[P]^\downarrow}$$

s.t. $\forall x . \forall y \in gl(x) . \forall p \in \text{dom}(y) . x(p) \in \{\text{true}\} \times D$ (i.e. all ports executed by $y$ are enabled in $x$).
Glue for a composition of components \( \{ B_i = (Q_i, P_i, \rightarrow_i) \}_{i=1}^{n} \)

A glue \( gl \) restricts executions \( \downarrow^y \) in a composition, based (only) on interface state \( x \in [P]^\uparrow \) of ports \( P = P_1 \cup \ldots \cup P_n \):

\[
\forall x . \forall y \in gl(x) . \forall p \in \text{dom}(y) . x(p) \in \{\text{true}\} \times \mathcal{D} \quad \text{(i.e. all ports executed by } y \text{ are enabled in } x)\.
\]
Example of Glue: Flight Control ($FC$) + Flaps ($F$)

- Synchronize both ports retract (resp. extend) if they are enabled.
Example of Glue: Flight Control ($FC$) + Flaps ($F$)

$gl(x) \subseteq \{y, y'\}$

$y = \{FC\. retract, F\. retract\}$

$y' = \{FC\. extend, F\. extend\}$

$y \in gl(x)$ iff $x(FC\. retract) \land x(FC\. retract)$

$y' \in gl(x)$ iff $x(FC\. extend) \land x(FC\. extend)$

- Synchronize both ports retract (resp. extend) if they are enabled.
1 Components and Glues

2 A Notion of Interactions

3 A Notion of Connectors

4 Conclusion
A Notion of Interactions

**Proposal for Interaction**

An interaction $\alpha = (g_\alpha, f_\alpha)$ is a synchronization between a subset of ports $P_\alpha \neq \emptyset$ of $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$ s.t. $|P_\alpha \cap P_i| \leq 1$ and

- the guard $g_\alpha : [P_\alpha \cup P_\alpha^o]^\uparrow \rightarrow \mathbb{B}$ sat. $g_\alpha(x) \Rightarrow x(P_\alpha) \subseteq \{\text{true}\} \times X$
- $f_\alpha : [P_\alpha \cup P_\alpha^o]^\uparrow \rightarrow [P_\alpha]^\downarrow$ is the transfer function

$P^o$ are the ports observed by interaction $\alpha$, possibly $P_\alpha \cup P_\alpha^o = P$. 

**Interactions**

\[ g_\alpha(x_1, x_2, x_3, x_4) \]
\[ (y_1, y_2, y_3) := f_\alpha(x_1, x_2, x_3, x_4) \]

\[ P_\alpha = \{p_1, p_2, p_3\} \]
\[ P_\alpha^o = \{p_4\} \]

\[ g_\alpha(x_1, x_2, x_3, x_4) \]
\[ (y_1, y_2, y_3) := f_\alpha(x_1, x_2, x_3, x_4) \]
A set of interactions $\Gamma$ corresponds to the glue $gl$ satisfying:

$$gl(x) = \{ f_\alpha(x) \mid \alpha \in \Gamma \land g_\alpha(x) \}.$$
Example: Flight Control + Flaps

\[
[\text{FC.extend} \land \text{F.extend}]
\]

\[
[\text{FC.retract} \land \text{F.retract}]
\]
Example: Flight Control + Flaps

\[ FC.\text{extend} \land F.\text{extend} \]
\[ FC.\text{retract} \land F.\text{retract} \]
Example: Flight Control + Flaps (cont’d)

FlightControl

retract extend noFlaps

TAKEOFF

FLIGHT

retract extend noFlaps

LANDING

LANDING W/O FL.

[¬F.extend]

Flaps

extend retract

EXTENDED

FAIL

RETRACTED

Jacques Combaz

Revisiting the Glue of BIP

MeTRiD 2019
Universality of Interactions

Theorem (Universality)

For any glue $gl$ on a composition $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1...n}$ there exists an equivalent set of interactions $\Gamma$ s.t.:

$$|\Gamma| \leq 2^{|P|-1}|gl|$$

where $|gl| = \max_{x \in [P]^\uparrow} |gl(x)|$.

Sketch of proof for $|gl| = K < +\infty$:

For $P' \subseteq P$ s.t. $\forall i \ |P' \cap P_i| \leq 1$ we build interactions $\{\alpha_j\}_{j=1..K}$ s.t.:

- $\forall j$ we have $P_{\alpha_j} = P'$ and $P_{\alpha_j} = P_1 \cup \ldots \cup P_n$
- $\forall x$ we have $\{ y \in gl(x) \ | \ \text{dom}(y) = P' \} = \{ y_j \}_{j=1..L}$, $L \leq K$, and:
  - $g_{\alpha_j}(x) \iff 1 \leq j \leq L$
  - $f_{\alpha_j}(x) = y_j$ for $j = 1..L$, any value otherwise.
Universality but not Minimalism
Universality but not Minimalism
Universality but not Minimalism

[A diagram showing interactions between FlightControl, Flaps, and ManualControl with logical expressions and state transitions.]

(For detailed expressions and transitions, refer to the original document.)
1. Components and Glues

2. A Notion of Interactions

3. A Notion of Connectors

4. Conclusion
A Notion of Connectors

Connectors: compact representations for subsets interactions $\Gamma$ s.t.:

- guards are mutually exclusive (max. 1 enabled interaction at a state):
  \[ \forall \alpha \neq \beta \in \Gamma . \ \forall x . \neg (g_\alpha(x) \land g_\beta(x)) \]

- at most one participating port per component:
  \[ |P_i \cap \bigcup_{\alpha \in \Gamma} P_\alpha| \leq 1. \]

Remark: for such sets $\Gamma$, for any $p \in \bigcup_{\alpha \in \Gamma} P_\alpha$ we can define:

- $g_p : \left[ \bigcup_{\alpha \in \Gamma} P_\alpha \cup P_\alpha^o \right]^\uparrow \to \mathbb{B}$ s.t. $g_p = \bigvee_{\alpha \in \Gamma} g_\alpha \land p \in P_\alpha$

- $f_p : \left[ \bigcup_{\alpha \in \Gamma} P_\alpha \cup P_\alpha^o \right]^\uparrow \to \mathcal{D}$ s.t. $f_p(x) = f_\alpha(x)(p)$ if $p \in P_\alpha$ and $g_\alpha(x)$. 
Connectors

A connector $C$ is defined by

- a set of potentially participating ports $P_C$ s.t. $\forall i . |P_C \cap P_i| \leq 1$
- a set of observed ports $P^o_C$
- guards $\{g_p : [P_C \cup P^o_C]^\uparrow \rightarrow \mathbb{B}\}_{p \in P_C}$ s.t. $g_p(x) \Rightarrow p$ is enabled in $x$
- transfer functions $\{f_p : [P_C \cup P^o_C]^\uparrow \rightarrow \mathcal{D}\}_{p \in P_C}$. 
Connectors

A connector \( C \) corresponds to interactions \( \{ \alpha \mid P_\alpha \neq \emptyset \land P_\alpha \subseteq P_C \} \) and:

- \( P^o_\alpha = (P_C \setminus P_\alpha) \cup P^o_C \)
- \( \forall x . g_\alpha(x) = \bigwedge_{p \in P_\alpha} g_p(x) \land \bigwedge_{p \in P_C \setminus P_\alpha} \neg g_p(x) \)
- \( \forall x . f_\alpha(x) : p \mapsto f_p(x) \).
Glue for Connectors

A set of connectors $C$ corresponds to the glue $gl$ satisfying $y \in gl(x)$ iff $\exists C \in C$ s.t.:

- $\text{dom}(y) = \{ p \in P_C \mid g_p(x) \} \neq \emptyset$
- $\forall p \in \text{dom}(y). \ y(p) = f_p(x)$.

$\rightarrow$ computing $gl$ is linear in number of connectors and part. ports.
Example: Flight Control + Flaps + Manual Control

- **Flight Control**
  - retract
  - extend
  - noFlaps

- **Flaps**
  - extend
  - retract

- **Manual Control**
  - retract
  - extend

Logic:
- \([F.\text{extend} \lor F.\text{retract}]\)
- \([\neg F.\text{extend} \land \neg F.\text{retract}]\)
- \([MC.\text{extend}]\)
- \([MC.\text{retract}]\)
- \([F.\text{retract}]\)
Contributions to the BIP framework:

- formalization of glues taking into account data
- proposed interactions: expressive enough to encode any glue $g_l$

\[
g_l \cong p_1 p_2 p_3 p_4 + \ldots
\]

- proposed connectors:
  - compact representations of interactions (incl. guards / data transfers)

\[
\cong p_1 p_2 p_3 p_4 + p_1 p_2 p_3 p_4 + p_1 p_2 p_3 p_4 + \ldots
\]

- runtime evaluation of linear complexity.
Thank you.
Implementation Steps

- **Source2source transformers**
- **BIP LANGUAGE**
- **BIP COMPILER**
  - Parser
  - BIP Meta-Model
  - S/R BIP Model
    - Transformers
    - Distributed BIP Generator
    - C++ Generator (engine based)
    - Distributed BIP Generator

- **Language Factory**
  - SMC-BIP
  - DFinder

- **Validation**
- **Code Generation & Runtimes**
  - C++
  - BIP Executable
  - BIP Engine Runtime
  - Platform

- **Communication Primitives (Send/Receive)**
  - BIP Executable
  - BIP Executable
  - BIP Executable

- **Platforms**
  - nesC
  - DOL
  - SimuLink
  - C
  - Lustre

- **Transformers**
  - C

- **Jacques Combaz**
  - Revisiting the Glue of BIP
  - MeTRiD 2019
Implementation Steps

1. IR: added new connectors
2. translation: old glue to new glue
3. generation of the new glue (direct, no engine)

Jacques Combaz
Revisiting the Glue of BIP
MeTRiD 2019 34 / 33
Implementation Steps

1. IR: added new connectors
2. translation: old glue to new glue
3. generation of the new glue (direct, no engine)
4. extend the language for new connectors

Jacques Combaz
Revisiting the Glue of BIP
BIP offers a rich language for assembling components using:

- **interactions** = multiparty rendez-vous + data transfer
- **connectors** = compact representation of sets of interactions: *rendez-vous, broadcast, atomic broadcast, causal chain, ...*
- **priorities** = partial order on interactions
An interaction $\alpha = (g_\alpha, f_\alpha)$ is a synchronization between a subset of ports $P_\alpha \neq \emptyset$ of $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1}^n$ s.t. $|P_\alpha \cap P_i| \leq 1$ and

- the guard $g_\alpha : [P_\alpha]^\uparrow \rightarrow \mathbb{B}$ satisfies $g_\alpha(x) \Rightarrow \forall p \in P_\alpha . \mathcal{E}_p(x)$
- $f_\alpha : [P_\alpha]^\uparrow \rightarrow [P_\alpha]^\uparrow$ is the transfer function.

The above functions can be applied to any $x \in [P]^\uparrow$ by taking $x|_{P_\alpha}$.
Priorities

Priorities is a set of rules that define a strict partial order over a set of interactions $\Gamma$. A special case is maximal progress over a subset $\Gamma'$:

$$\alpha \subsetneq_{\Gamma'} \beta \iff \alpha, \beta \in \Gamma' \land P_\alpha \subsetneq P_\beta.$$
Semantics of Interactions + Priorities

\[ \alpha < \beta \]
\[ \alpha < \gamma \]

Glue for Interactions + Priorities

Interactions \( \Gamma \) under priorities \( \prec \) corresponds to the glue \( gl \) satisfying:

\[
gl(x) = \{ f_\alpha(x) \mid \alpha \in \Gamma \land g_\alpha(x) \land \forall \alpha \prec \beta . \neg g_\beta(x) \}.
\]
Connectors

$P_\alpha = \{p_1\}, \quad P_\beta = \{p_1, p_2, p_3\}$

$g_\alpha \ f_\alpha \\ g_\beta \ f_\beta$

$\alpha < \beta$

$p_1 \quad p_2 \quad p_3$

Connectors = subsets of interactions + priorities, defined using trigger ports (▲) and synchron ports (●) (possibly hierarchically):

- ports $P_\alpha$ of interactions contain at least one ▲ or all the ports
- guards $g_\alpha$ and transfer functions $f_\alpha$: given by enumeration
- priorities = maximal progress ($\subseteq$).
Connectors: Recursive Characterization of Subsets of Ports $P_\alpha$

- $P_\alpha \models \{c_1, c_2, \ldots, c_n\}$ iff $P_\alpha = \bigcup_{i=1}^{n} P_i$ and $\forall i \cdot P_i \models C_i$
- $P_\alpha \models \{c_1, c_2, \ldots, c_j, \ldots, c_n\}$ iff $P_\alpha = \bigcup_{i \in \{j\} \cup I} P_i$ and $\forall i \cdot P_i \models C_i$
- $P_i \models p_i$ iff $P_i = \{p_i\}$ for a (component) port $p_i$. 

$P_\alpha = \{p_1\}, \; P_\beta = \{p_1, p_2, p_3\}$

$g_\alpha f_\alpha \quad g_\beta f_\beta$

$\alpha < \beta$
### Example of Connectors

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rendez-vous</strong></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>broadcast</strong></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>atomic broadcast</strong></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>causality chain</strong></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
</tbody>
</table>

#### Interactions (Priorities = $\emptyset$)

- **rendez-vous**
  - $\{p_1, p_2, p_3, p_4\}$

- **broadcast**
  - $\{p_1, p_2\}$
  - $\{p_1, p_2, p_3\}$
  - $\{p_1, p_2, p_4\}$
  - $\{p_1, p_2, p_3, p_4\}$

- **atomic broadcast**
  - $\{p_1\}$
  - $\{p_1, p_2, p_3, p_4\}$

- **causality chain**
  - $\{p_1\}$
  - $\{p_1, p_2\}$
  - $\{p_1, p_2, p_3\}$
  - $\{p_1, p_2, p_3, p_4\}$
### Example of Connectors: Concrete Syntax

<table>
<thead>
<tr>
<th>Connector Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RendezVous (rendez-vous)</td>
<td>connector type RendezVous(IPort p1, IPort p2, IPort p3, IPort p4)</td>
</tr>
<tr>
<td></td>
<td>define p1 p2 p3 p4</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 p3 p4 provided ( /* guard <em>/ ) down { /</em> transfer function */ }</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>Broadcast (broadcast)</td>
<td>connector type Broadcast(IPort p1, IPort p2, IPort p3, IPort p4)</td>
</tr>
<tr>
<td></td>
<td>define p1’ p2 p3 p4</td>
</tr>
<tr>
<td></td>
<td>on p1 provided (...) down { ... }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 provided (...) down { p2.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p3 provided (...) down { p3.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p4 provided (...) down { p3.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 p3 provided (...) down { p2.x = p1.x; p3.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 p4 provided (...) down { p2.x = p1.x; p4.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p3 p4 provided (...) down { p3.x = p1.x; p4.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 p3 p4 provided (...) down { p2.x = p1.x; p3.x = p1.x; p4.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>AtomicBroadcast (atomic broadcast)</td>
<td>connector type AtomicBroadcast(IPort p1, IPort p2, IPort p3, IPort p4)</td>
</tr>
<tr>
<td></td>
<td>define p1’ (p2 p3 p4)</td>
</tr>
<tr>
<td></td>
<td>on p1 provided (...) down { ... }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 p3 p4 provided (...) down { p2.x = p1.x; p3.x = p1.x; p4.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>CausalityChain (causality chain)</td>
<td>connector type CausalityChain(IPort p1, IPort p2, IPort p3, IPort p4)</td>
</tr>
<tr>
<td></td>
<td>define ((p1’ p2’) p3’) p4</td>
</tr>
<tr>
<td></td>
<td>on p1 provided (...) down { ... }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 provided (...) down { p2.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 p3 provided (...) down { p2.x = p1.x; p3.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>on p1 p2 p3 p4 provided (...) down { p2.x = p1.x; p3.x = p1.x; p4.x = p1.x; }</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>
## Example of Connectors: Concrete Syntax (No Data)

<table>
<thead>
<tr>
<th>Connector Type</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| *rendez-vous*   | ```
connector type RendezVous(Port p1, Port p2, Port p3, Port p4)
    define p1 p2 p3 p4
end
``` |
| *broadcast*     | ```
connector type Broadcast(Port p1, Port p2, Port p3, Port p4)
    define p1’ p2 p3 p4
end
``` |
| *atomic broadcast* | ```
connector type AtomicBroadcast(Port p1, Port p2, Port p3, Port p4)
    define p1’ (p2 p3 p4)
end
``` |
| *causality chain* | ```
connector type CausalityChain(Port p1, Port p2, Port p3, Port p4)
    define ((p1’ p2’) p3’) p4
end
``` |
Example: Flight Control + Flaps

- Synchronize ports `extend` (resp. `retract`) of `FlightControl` and `Flaps`
Example: Flight Control + Flaps + Manual Control

- Can always execute port extend / retract in FlightControl
Glue in BIP = Connectors + Priorities

In BIP, the glue $gl$ is defined by connectors $C_1, \ldots, C_m$ and priorities $\prec_{\text{user}}$ and corresponds to interactions $\Gamma$ and priorities $\prec$ s.t.:

- $\Gamma = \bigcup_{i=1 \ldots m} \Gamma_j$ where $\Gamma_j$ is the set of interactions of $C_j$
- $\prec = \prec_{\text{user}} \oplus \bigoplus_{j=1 \ldots m} \lessdot \Gamma_j$ where $\prec_{\text{user}}$ must be compatible with $\lessdot \Gamma_j$. 
Implementation of Connectors/Interactions + Priorities

Implementation

For interactions $\Gamma$ and priorities $\prec$, an implementation evaluate $gl(x)$ by searching for one/all interaction(s) $\alpha = (g_\alpha, f_\alpha) \in \Gamma$ s.t.:

$$g_\alpha(x) \land \bigwedge_{\alpha \prec \beta} \alpha \prec \beta \Rightarrow \neg g_\beta(x).$$

Optimization #1: filter by $g_\alpha$, then apply $\prec$

1. Compute $\Delta = \{ \alpha \mid g_\alpha(x) \}$.
2. Remove any $\alpha$ from $\Delta$ if $\exists \beta \in \Delta$ s.t. $\alpha \prec \beta$.

Optimization #2: follow priority order $\prec$

Compute $\text{search}(\Gamma)$ where $\top(\Delta) = \{ \alpha \in \Delta \mid \forall \beta \in \Delta . \alpha \not\prec \beta \}$ and:

- $\text{search}(\emptyset) = \emptyset$
- $\text{search}(\Delta) = \{ \alpha \in \top(\Delta) \mid g_\alpha(x) \} \cup \text{search}(\{ \alpha \in \top(\Delta) \mid \neg g_\alpha(x) \})$. 
Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal “glue”!
Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal “glue”!
  - execute *extend* if *extend* or *retract* is enabled in *Flaps*
  - execute *noFlaps* if both *extend* and *retract* are disabled in *Flaps*
Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal "glue"!
Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal “glue”!

- flattening of hierarchical structures: not always possible

![Diagram showing flattening of hierarchical structures](image)
Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal “glue”!
- flattening of hierarchical structures: not always possible
- not compact w.r.t. data (exponential enumeration)

```
connector type Broadcast(IPort p1, IPort p2, IPort p3, ..., IPort pn)
  define p1’ p2 p3 ... pn
  on p1        provided (...) down { ... }     // 1
  on p1 p2     provided (...) down { p2.x = p1.x; }  // 2
  on p1 p3     provided (...) down { p3.x = p1.x; }  // 3
  ...
  on p1 p2 p3 ... pn provided (...) down { p2 = p1.x; p2 = p1.x; ... pn = p1.x; }  // 2^n-1
end
```
Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal “glue”!
- flattening of hierarchical structures: not always possible
- not compact w.r.t. data (exponential enumeration)
- runtime complexity

<table>
<thead>
<tr>
<th>WC complexity</th>
<th>searching</th>
<th>memory allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rendez-vous</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>flat + (1)</td>
<td>$O(n)$</td>
<td>1</td>
</tr>
<tr>
<td>flat</td>
<td>$O(2^n)$</td>
<td>$O\left(\binom{n}{n/2}\right)$</td>
</tr>
<tr>
<td>hierarchy + (1) + (2)</td>
<td>$O(n)$</td>
<td>1</td>
</tr>
<tr>
<td>hierarchy + (2)</td>
<td>$O(2^n)$</td>
<td>$O\left(\binom{n}{n/2}\right)$</td>
</tr>
<tr>
<td>hierarchy</td>
<td>$O(2^n)$</td>
<td>$O(2^n)$</td>
</tr>
</tbody>
</table>

(1): guards s.t. $g_\alpha \land g_\beta \Rightarrow g_\gamma$ if $P_\alpha \cup P_\beta \subseteq P_\gamma$
(2): guards are independent from lower levels
Encoding Priorities in Interactions

Priorities $\prec$ corresponds to modifying interactions $\alpha$ of a set $\Gamma$ s.t.:

1. $P'_\alpha = P_\alpha$
2. $P'^{lo}_\alpha = P^o_\alpha \cup \bigcup_{\alpha \prec \beta} P_\beta \cup P^o_\beta$
3. $g'_\alpha : [P_\alpha \cup P'^{lo}_\alpha]^\uparrow \rightarrow \mathbb{B}$ s.t. $g'_\alpha(x) = g_\alpha(x) \land \bigwedge_{\alpha \prec \beta} \neg g_\beta(x)$
4. $f'_\alpha : [P_\alpha \cup P'^{lo}_\alpha]^\uparrow \rightarrow [P_\alpha]^\uparrow$ s.t. $f'_\alpha(x) = f_\alpha(x)$.
Encoding Priorities in Interactions

Priorities \( \prec \) corresponds to modifying interactions \( \alpha \) of a set \( \Gamma \) s.t.:

- \( P'_\alpha = P_\alpha \)
- \( P'^{\circ}_{\alpha} = P_\alpha^\circ \cup \bigcup_{\alpha \prec \beta} P_\beta \cup P_\beta^\circ \)
- \( g'_\alpha : [P_\alpha \cup P'^{\circ}_{\alpha}]^\uparrow \rightarrow \mathbb{B} \) s.t. \( g'_\alpha(x) = g_\alpha(x) \land \bigwedge_{\alpha \prec \beta} \neg g_\beta(x) \)
- \( f'_\alpha : [P_\alpha \cup P'^{\circ}_{\alpha}]^\uparrow \rightarrow [P_\alpha]^\uparrow \) s.t. \( f'_\alpha(x) = f_\alpha(x) \).
Example of (Atomic) Component: Aircraft Flaps

port type Port(const int n)

atom type Flaps(int MAX_CYCLES)
  data int n
  export port Port extend(nbCycles),
  retract(nbCycles)

place EXTENDED, RETRACTED

initial to EXTENDED
  do { nbCycles = 0; }

on retract from EXTENDED to RETRACTED
  provided (nbCycles < MAX_CYCLES)

on extend from RETRACTED to EXTENDED
  do { nbCycles++; }
end