The π-Calculus for SoS: Novel π-Calculus for the Formal Modeling of Software-intensive Systems-of-Systems

Flavio Oquendo

flavio.oquendo@irisa.fr

http://people.irisa.fr/Flavio.Oquendo/

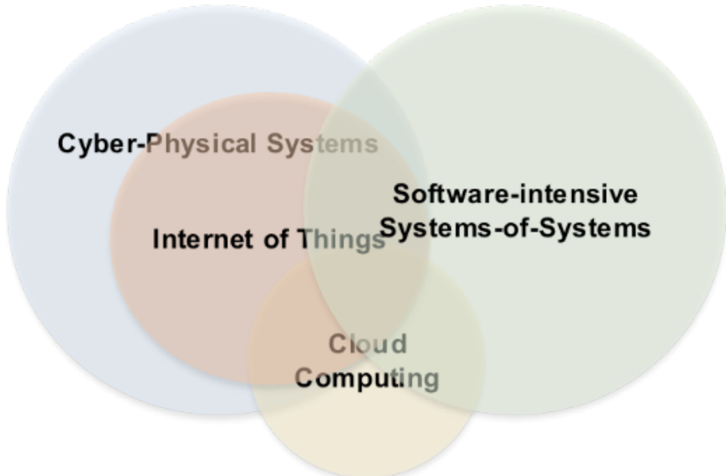
Communicating Process Architectures 2016 (CPA 2016)

Outline

- Introduction: Motivation to conceive the π-Calculus for SoS
  - Need of formal description techniques to model SoS architectures
  - Limitations of current formal description techniques
- Problematics
  - Needs for a novel process calculus for SoS
- Formal Approach for Conceiving the π-Calculus for SoS
  - Novel process calculus meeting SoS needs: The π-Calculus for SoS
- Formal Definition of the π-Calculus for SoS
  - Formal transition system defining the π-Calculus for SoS
- Validating the Formal Operational Semantics of the π-Calculus for SoS
- Conclusion

Introduction: Software-intensive System-of-Systems

- Software-intensive Systems-of-Systems (SoS)
  - Systems are independently developed, operated, managed, evolved and eventually retired
  - Increasingly, networks make communication and cooperation possible among these independent systems
  - These networked systems evolved to form Systems-of-Systems
  - Systems-of-Systems are evolutionary developed from independent systems to achieve missions not possible by a constituent system alone
    - SoS creates emergent behavior
  - Systems-of-Systems have evolutionary architectures



Introduction: System-of-Systems Architecture

- Software-intensive Systems
  - were simple and became complicated: needs engineering
  - are becoming complex as SoS: needs architecture
    - complexity poses the need for separation of concerns between architecture and engineering
    - architecture: focus on reasoning about interactions of parts and their emergent properties
- Issues:
  - Do the process calculi constituting the formal foundations of ADLs for single systems provide enough expressive power for modeling SoS architectures?
  - Beyond the process calculi underlying single system ADLs, are there other process calculi that would be suitable for describing SoS architectures?

Limitations of the state-of-the-art ADLs for describing SoS Architectures

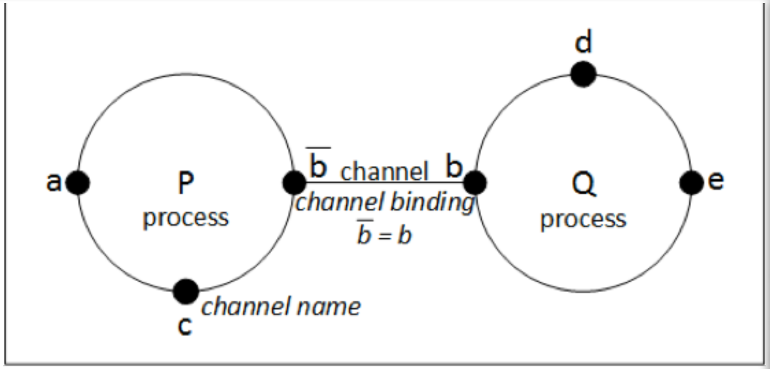
- **Software Architecture Description Language (ADL)**
  - Subject of intensive research in the last 20 years
  - Proposal of several ADLs for formally describing Software Architecture (see IFIP/IEEE ICSA, ECSA, QoSA…; IEEE TSE, ACM TOSEM, JSS, FGCS, IEEE Software…)

- **ADLs for Single Systems**
  - None of those ADLs has the expressive power to describe the Software Architecture of a Software-intensive SoS
    - Formal foundations of these ADLs are too limited to describe SoS Architectures
- **A novel formal foundation is needed for representing, analyzing and evolving SoS Architectures**
  - Need of a novel formal foundation to describe SoS Architectures

Formal Foundations of ADLs for Single Systems: Process Calculi

- Formal foundations for describing the Architecture of Single Systems are mostly based on Process Calculi
  - FSP: the formal foundation of Darwin ADL
  - CSP: the formal foundation of Wright ADL
  - π-Calculus: the formal foundation of π-ADL
- Process Calculi
  - Mathematical theory for formally modeling concurrent communicating systems
    - provide a formalism for the description of communicating processes
    - provide algebraic laws that allow process descriptions to be manipulated and analyzed
    - enable formal reasoning about equivalences between processes
  - The Process Calculus of reference
    - The π-Calculus (ACM Turing Award for Robin Milner in 1991)

Formal Foundations of ADLs for Single Systems: The π-Calculus



- π-Calculus
  - Basic concepts
    - Processes (single and composite processes)
    - Channels (interaction points) – channels support the binding of interaction points in concurrent processes
    - Names (including channel names)
    - Mobility (channels are used to send and receive names that may be channels)
- π-Calculus has shown to be a suitable formal foundation for describing and analyzing the architecture of software-intensive single systems
- However, π-Calculus as well as other process calculi, e.g. FSP/CSP, are too limited to cope with SoS architecture needs

Formal Foundations of ADLs for Single Systems: Process Calculi

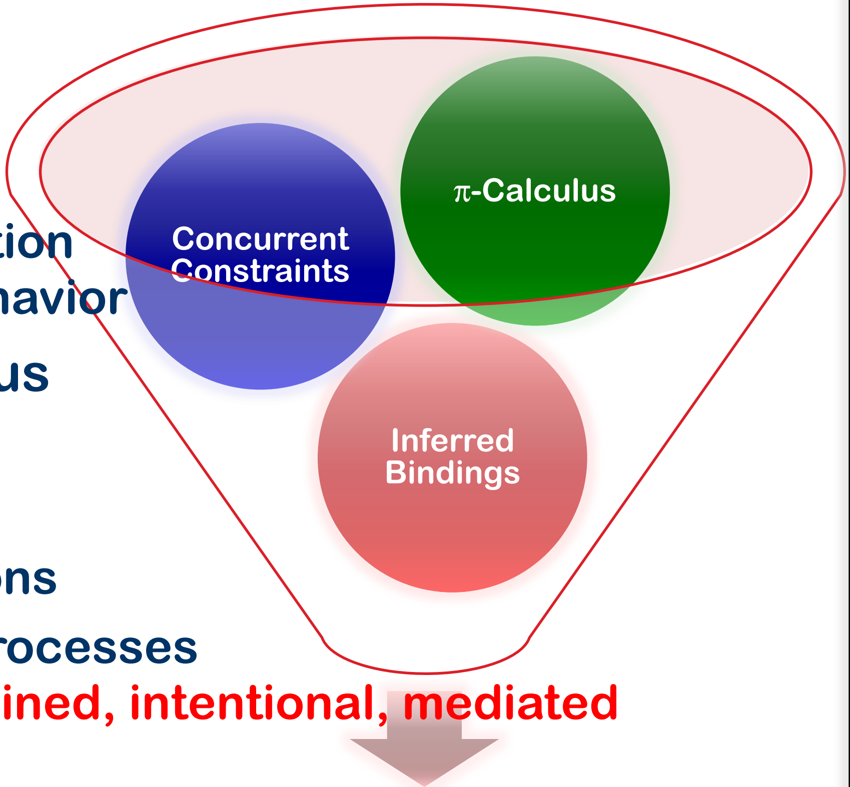
- Different process calculi were applied for formally describing the architecture of single software-intensive systems
  - Including different variants of the π-Calculus
- Bindings in all these process calculi for the architecture description of single software-intensive systems are:
  - endogenously decided at design-time
  - extensionally declared at design-time
  - unconstrained by local environments
  - unmediated between constituents
- Expressive power of these process calculi based on design-time decisions do not cope with SoS defining characteristics
- Research question:
  - How to enhance the π-Calculus for formally describing SoS architectures?

Differences of Description Needs between Single Systems and Systems-of-Systems

- None of the existing π-Calculi provides a suitable basis for formally describing and analyzing SoS architectures
- Needs related to SoS Architecture Description
  - Representing systems as processes
  - Representing mediators between communicating processes via inferred channel bindings
    - In SoS, the binding between channels must be exogenous
      - Problem: In the π-Calculus binding is endogenous
    - In SoS, the binding must be constrained by local contexts
      - Problem: In the π-Calculus binding is unconstrained
    - In SoS, the binding between channels must be intentional
      - Problem: In the π-Calculus binding is extensional
    - In SoS, the binding between channels must be mediated
      - Problem: In the π-Calculus binding is unmediated

Formal Approach for Describing SoS Architectures: The π-Calculus for SoS

- **Design decisions for the π-Calculus for SoS**
  - Generalization of the π-Calculus with mediated constraints
    - Subsuming the original π-Calculus
    - Coping with uncertainty
      - In SoS, partial information contributes to uncertainty, in addition to the uncertainty of emergent behavior
  - Definition of an enhanced π-Calculus based on
    - Concurrent interacting processes
    - Concurrent constraints on interactions
    - Inferred bindings from concurrent processes and constraints: exogenous, constrained, intentional, mediated
  - Emergent behavior
    - Drawn from constrained interactions



π-Calculus for SoS

Formal Approach for Describing SoS Architectures: The π-Calculus for SoS

- **The π-Calculus for SoS**: meeting the needs of SoS architecture description
  - the π-Calculus for SoS generalizes the π-Calculus with the notion of computing with partial information based on concurrent constraints
    - A **constraint** represents partial information on the state of the environment as perceived by mediated constituent systems
    - During the computation, the current state of the environment is specified by a set of told constraints
    - Processes can change the state of the environment by telling information
      - **tell** new constraints or **untell** existing constraints
    - Processes can synchronize by entailing information from the environment
      - **ask** whether a given constraint can be inferred from the told constraints in the environment

Abstract Syntax of the π-Calculus for SoS

- The formal definition of the π-Calculus for SoS encompasses its formal abstract syntax and formal semantics
  - formal operational semantics of π-Calculus for SoS is defined by means of a formal transition system, expressed by labelled transition rules

Transition rule:

$$\frac{P_1 \xrightarrow{\alpha_1} P_1' \ \ldots \ P_n \xrightarrow{\alpha_n} P_n'}{C \xrightarrow{\alpha} C'}$$

where *side conditions*

Abstract syntax of π-Calculus for SoS

constrainedBehavior ::= behavior$_1$
  | restriction$_1$ **.** constrainedBehavior$_1$ -- *Constrained Behavior*
  | **behavior** name$_1$ **(** value$_0$ …**,** value$_n$ **) is {** behavior$_1$ **}** -- *Definition*
  | **constraint** name$_1$ **is {** constraint$_1$ **}** -- *Constraint Definition*
  | **compose {** constrainedBehavior$_0$ … **and** constrainedBehavior$_n$ **}**
behavior ::= baseBehavior$_1$
  | restriction$_1$ **.** behavior$_1$ -- *Unconstrained Behavior*
  | **repeat {** behavior$_1$ **}** -- *Repeat*
  | **apply** name$_1$ **(** value$_0$ …**,** value$_n$ **)** -- *Application*
  | **compose {** behavior$_0$ … **and** behavior$_n$ **}** -- *Composition*
baseBehavior ::= action$_1$ **.** behavior$_1$ -- *Sequence*
  | **choose {** action$_0$ **.** baseBehavior$_0$ -- *Choice*
  or action$_1$ **.** baseBehavior$_1$ … **or** action$_n$ **.** baseBehavior$_n$ **}**
  | **if** constraint$_1$ **then {** baseBehavior$_1$ **} else {** baseBehavior$_2$ **}**
  | **done** -- *Termination*
action ::= baseAction$_1$
  | **tell** constraint$_1$ -- *Tell*
  | **untell** constraint$_1$ -- *Unsaid*
  | **check** constraint$_1$ -- *Check*
  | **ask** constraint$_1$ -- *Ask*
baseAction ::= **via** connection$_1$ **send** value$_0$ -- *Output*
  | **via** connection$_1$ **receive** name$_0$ **:** type$_0$ -- *Input*
  | **unobservable** -- *Unobservable*
connection ::= **connection** name$_1$
restriction ::= **value** name$_1$ **=** value$_0$ | connection$_1$

Formal Semantics of Actions in the π-Calculus for SoS

Actions:

- **send** value via connection
- **receive** value via connection
- **unobservable** internal actions
- **tell** constraint to local environment
- **untell** constraint from local environment
- **check** if constraint is consistent with local environment
- **ask** if constraint can be entailed from local environment

Formal semantics of π-Calculus for SoS: labeled transition rules for actions

Output:

$$\frac{}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ (\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1\,.\,\text{behavior}_1)\end{array}\right\} \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1} \textbf{compose}\left\{\text{constraint}_{0..n}\ \textbf{and}\ \text{behavior}_1\right\}}$$

Input:

$$\frac{}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ (\textbf{via}\ \text{connection}_1\ \textbf{receive}\ \text{value}\,.\,\text{behavior}_1)\end{array}\right\} \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{receive}\ \text{value}_1} \textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ (\text{value} = \text{value}_1)\\ \textbf{and}\ \text{behavior}_1\end{array}\right\}}$$

where (constraint$_{0..n}$ **and** (value = value$_1$)) *is consistent, i.e. binding* (value = value$_1$) *can be consistently asserted together with* constraint$_{0..n}$

Unobservable:

$$\frac{}{\textbf{compose}\left\{\text{constraint}_{0..n}\ \textbf{and}\ (\textbf{unobservable}\,.\,\text{behavior}_1)\right\} \xrightarrow{\tau} \textbf{compose}\left\{\text{constraint}_{0..n}\ \textbf{and}\ \text{behavior}_1\right\}}$$

Tell:

$$\frac{}{\textbf{compose}\left\{\text{constraint}_{0..m}\ \textbf{and}\ (\textbf{tell}\ \text{constraint}_n\,.\,\text{behavior}_1)\right\} \xrightarrow{\tau} \textbf{compose}\left\{\text{constraint}_{0..m}\ \textbf{and}\ \text{constraint}_n\ \textbf{and}\ \text{behavior}_1\right\}}$$

where (constraint$_{0..m}$ **and** constraint$_n$) *is consistent, i.e.* constraint$_n$ *can be consistently asserted with* constraint$_{0..m}$

Untell:

$$\frac{}{\textbf{compose}\left\{\text{constraint}_{0..n}\ \textbf{and}\ (\textbf{untell}\ \text{constraint}_m\,.\,\text{behavior}_1)\right\} \xrightarrow{\tau} \textbf{compose}\left\{(\text{constraint}_{0..n} - \text{constraint}_m)\ \textbf{and}\ \text{behavior}_1\right\}}$$

where (constraint$_{0..n}$ – constraint$_m$) *is consistent, i.e.* constraint$_m$ *can be consistently retracted from* constraint$_{0..n}$

Check:

$$\frac{}{\textbf{compose}\left\{\text{constraint}_{0..n}\ \textbf{and}\ (\textbf{check}\ \text{constraint}_m\,.\,\text{behavior}_1)\right\} \xrightarrow{\tau} \textbf{compose}\left\{\text{constraint}_{0..n}\ \textbf{and}\ \text{behavior}_1\right\}}$$

where (constraint$_{0..n}$ **and** constraint$_m$) *is consistent, i.e.* constraint$_m$ *is checked to be consistent with* constraint$_{0..n}$

Ask:

$$\frac{}{\textbf{compose}\left\{\text{constraint}_{0..m}\ \textbf{and}\ (\textbf{ask}\ \text{constraint}_n\,.\,\text{behavior}_1)\right\} \xrightarrow{\tau} \textbf{compose}\left\{\text{constraint}_{0..m}\ \textbf{and}\ \text{behavior}_1\right\}}$$

where constraint$_{0..m}$ |- constraint$_n$, *i.e.* constraint$_n$ *can be derived from* constraint$_{0..m}$

Formal Semantics of Behaviors in π-Calculus for SoS

Behaviors:

- **restriction** of value to local behavior
- **communication** of value via connection between behaviors
  - synchronization between send and receive
  - equality constraint
- **extrusion** of value to another behavior (open restriction & close communication)
- **nondeterministic choice** among behaviors
- **conditional** choice between behaviors
- **repetition** of behavior
- **composition** of concurrent behaviors

Formal semantics of π-Calculus for SoS: labeled transition rules for behaviors

Restriction:

$$\frac{\text{constrainedBehavior}_1 \xrightarrow{\text{action}_1} \text{constrainedBehavior}_1'}{\textbf{value}\ \text{value}_1\,.\,\text{constrainedBehavior}_1 \xrightarrow{\text{action}_1} \textbf{value}\ \text{value}_1\,.\,\text{constrainedBehavior}_1'}$$

where $\text{value}_1 \notin \mathit{names}(\text{action}_1)$, *i.e.* value_1 *is not among the names used in* action_1

Communication:

$$\frac{\text{behavior}_1 \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1} \text{behavior}_1' \qquad \text{behavior}_2 \xrightarrow{\textbf{via}\ \text{connection}_2\ \textbf{receive}\ \text{value}} \text{behavior}_2'}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ (\text{connection}_1 = \text{connection}_2)\\ \textbf{and}\ \text{behavior}_1\ \textbf{and}\ \text{behavior}_2\end{array}\right\} \xrightarrow{\tau} \textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ (\text{connection}_1 = \text{connection}_2)\\ \textbf{and}\ (\text{value} = \text{value}_1)\ \textbf{and}\ \text{behavior}_1'\ \textbf{and}\ \text{behavior}_2'\end{array}\right\}}$$

where $\text{connection}_1 = \text{connection}_2$, *i.e.* $(\text{connection}_1 = \text{connection}_2)$ *is a binding resulting from an extrusion or unification*

Restriction-Open:

$$\frac{\text{constrainedBehavior}_1 \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1} \text{constrainedBehavior}_1'}{\textbf{value}\ \text{value}_1\,.\,\text{constrainedBehavior}_1 \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1} \text{constrainedBehavior}_1'}$$

where $\text{value}_1 \neq \text{connection}_1$, *i.e.* value_1 *cannot be used for connection as it is restricted*

Communication-Close:

$$\frac{\text{behavior}_1 \xrightarrow{\textbf{value}\ \text{connection}\,.\,\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{connection}} \text{behavior}_1' \qquad \text{behavior}_2 \xrightarrow{\textbf{via}\ \text{connection}_2\ \textbf{receive}\ \text{value}} \text{behavior}_2'}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ (\text{connection}_1 = \text{connection}_2)\\ \textbf{and}\ \text{behavior}_1\ \textbf{and}\ \text{behavior}_2\end{array}\right\} \xrightarrow{\tau} \textbf{value}\ \text{connection}\,.\,\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ (\text{connection}_1 = \text{connection}_2)\\ \textbf{and}\ (\text{value} = \text{connection})\\ \textbf{and}\ \text{behavior}_1'\ \textbf{and}\ \text{behavior}_2'\end{array}\right\}}$$

where $\text{value} \notin \mathit{free}(\text{behavior}_2)$, *i.e.* value *is not restricted in* behavior_2 *while* connection *is restricted in* behavior_1

Choice:

$$\frac{\text{constraint}_{0..n}\ \textbf{and}\ (\text{action}_i\,.\,\text{behavior}_i') \xrightarrow{\text{action}_i} \text{constraint}_{0..n}'\ \textbf{and}\ \text{behavior}_i'}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\\ \textbf{and}\ \textbf{choose}\{\text{action}_0\,.\,\text{behavior}_0' \ldots\ \textbf{or}\ \ \text{action}_m\,.\,\text{behavior}_m'\}\end{array}\right\} \xrightarrow{\text{action}_i} \textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}'\\ \textbf{and}\ \text{behavior}_i'\end{array}\right\}}$$

where $i \in 0..m$, *i.e. only one of the actions* $\text{action}_{0..m}$ *is performed*

Conditional-Then:

$$\frac{\text{behavior}_1 \xrightarrow{\text{action}_1} \text{behavior}_1' \qquad \text{constraint} \equiv \text{true}}{\textbf{compose}\{\text{constraint}_{0..n}\ \textbf{and}\ (\textbf{if}\ \text{constraint}\ \textbf{then}\ \text{behavior}_1\ \textbf{else}\ \text{behavior}_2)\} \xrightarrow{\text{action}_1} \textbf{compose}\{\text{constraint}_{0..n}\ \textbf{and}\ \ \text{behavior}_1}$$

Conditional-Else:

$$\frac{\text{behavior}_2 \xrightarrow{\text{action}_2} \text{behavior}_2' \qquad \text{constraint} \equiv \text{false}}{\textbf{compose}\{\text{constraint}_{0..n}\ \textbf{and}\ (\textbf{if}\ \text{constraint}\ \textbf{then}\ \text{behavior}_1\ \textbf{else}\ \text{behavior}_2)\} \xrightarrow{\text{action}_1} \textbf{compose}\{\text{constraint}_{0..n}\ \textbf{and}\ \text{behavior}_2'}$$

Repetition:

$$\frac{\text{behavior}_1 \xrightarrow{\text{action}_1} \text{behavior}_1'}{\textbf{repeat}\{\text{behavior}_1\} \xrightarrow{\text{action}_1} \text{behavior}_1'\ .\ \textbf{repeat}\{\text{behavior}_1\}}$$

where $\text{behavior}_1'\ .\ \text{behavior}_1$ *is a sequential composition, i.e.* $\text{behavior}_1'$ *must be performed before* behavior_1

Composition:

$$\frac{\text{constrainedBehavior}_i \xrightarrow{\text{action}_i} \text{constrainedBehavior}_i'}{\textbf{compose}\left\{\begin{array}{l}\text{constrainedBehavior}_0 \ldots\\ \textbf{and}\ \text{constrainedBehavior}_i\\ \textbf{and}\ \text{constrainedBehavior}_n\end{array}\right\} \xrightarrow{\text{action}_i} \textbf{compose}\left\{\begin{array}{l}\text{constrainedBehavior}_0 \ldots\\ \textbf{and}\ \text{constrainedBehavior}_i'\\ \textbf{and}\ \text{constrainedBehavior}_n\end{array}\right\}}$$

where $i \in 1..n$ **and** $\mathit{bound}(\text{action}_i) \cap \mathit{free}(\text{constrainedBehavior}_{0..n-i}) = \varnothing$,
i.e. restricted names in action_i *are not restricted elsewhere*

Understanding the Semantics of the π-Calculus for SoS

Communication:

$$\frac{\text{behavior}_1 \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1} \text{behavior}_1' \qquad \text{behavior}_2 \xrightarrow{\textbf{via}\ \text{connection}_2\ \textbf{receive}\ \text{value}} \text{behavior}_2'}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\ \textbf{and}\ (\text{connection}_1 = \text{connection}_2) \\ \textbf{and}\ \text{behavior}_1\ \textbf{and}\ \text{behavior}_2\end{array}\right\} \xrightarrow{\tau} \textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\ \textbf{and}\ (\text{connection}_1 = \text{connection}_2) \\ \textbf{and}\ (\text{value} = \text{value}_1)\ \textbf{and}\ \text{behavior}_1'\ \textbf{and}\ \text{behavior}_2'\end{array}\right\}}$$

Output:

$$\frac{}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n} \\ \textbf{and}\ (\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1 . \text{behavior}_1)\end{array}\right\} \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{send}\ \text{value}_1} \textbf{compose}\{\text{constraint}_{0..n}\ \textbf{and}\ \text{behavior}_1\}}$$

Input:

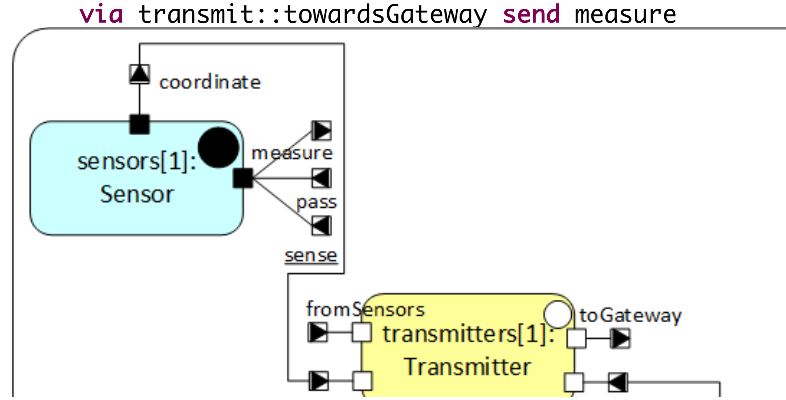
$$\frac{}{\textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n} \\ \textbf{and}\ (\textbf{via}\ \text{connection}_1\ \textbf{receive}\ \text{value} . \text{behavior}_1)\end{array}\right\} \xrightarrow{\textbf{via}\ \text{connection}_1\ \textbf{receive}\ \text{value}_1} \textbf{compose}\left\{\begin{array}{l}\text{constraint}_{0..n}\ \textbf{and}\ (\text{value} = \text{value}_1) \\ \textbf{and}\ \text{behavior}_1\end{array}\right\}}$$

where ($\text{constraint}_{0..n}$ **and** ($\text{value} = \text{value}_1$)) *is consistent, i.e. binding* ($\text{value} = \text{value}_1$) *can be consistently asserted together with* $\text{constraint}_{0..n}$

- **Communication**

```
Sensors[1] : system Sensor(lps=Coordinate::(10,10)) is { …
  behavior sensing is {
    value sensorcoordinate is Coordinate = lps
    tell sensorlocation is {sensorcoordinate = lps}
    via location::coordinate send sensorcoordinate
    via energy::threshold receive powerthreshold
    repeat {
      via energy::power receive powerlevel
      if (powerlevel > powerthreshold) then {
        tell powering is {powerlevel > powerthreshold}
        choose{
          via measurement::sense receive data
          via measurement::measure send
            tuple{coordinate=lps,depth=data::convert()}
        } or {
          via measurement::pass receive data
          via measurement::measure send data
        }
      }
    }
  }
}
```

```
transmitters[1] : mediator
  Transmitter(distancebetweengates:Distance) is { …
   behavior transmitting is {
     via location::fromCoordinate receive sendercoordinate
     via location::toCoordinate receive receivercoordinate
     ask sendercoordinate::distance(receivercoordinate)
        < distancebetweengates
     repeat {
          via transmit::fromSensors receive measure
          via transmit::towardsGateway send measure
     }
   }
}
```



- **Equality from coalition**

```
constraint {sensors[1]::location::coordinate = transmitters[1]::location::fromCoordinate}
```

Understanding the Semantics of the π-Calculus for SoS

Communication:

$$\frac{\text{behavior}_1 \xrightarrow{\textbf{via } \text{connection}_1 \textbf{ send } \text{value}_1} \text{behavior}_1' \qquad \text{behavior}_2 \xrightarrow{\textbf{via } \text{connection}_2 \textbf{ receive } \text{value}} \text{behavior}_2'}{\textbf{compose}\begin{Bmatrix}\text{constraint}_{0..n} \textbf{ and } (\text{connection}_1 = \text{connection}_2) \\ \textbf{and } \text{behavior}_1 \textbf{ and } \text{behavior}_2\end{Bmatrix} \xrightarrow{\tau} \textbf{compose}\begin{Bmatrix}\text{constraint}_{0..n} \textbf{ and } (\text{connection}_1 = \text{connection}_2) \\ \textbf{and } (\text{value} = \text{value}_1) \textbf{ and } \text{behavior}_1' \textbf{ and } \text{behavior}_2'\end{Bmatrix}}$$

Output:

$$\frac{}{\textbf{compose}\begin{Bmatrix}\text{constraint}_{0..n} \\ \textbf{and } (\textbf{via } \text{connection}_1 \textbf{ send } \text{value}_1 . \text{behavior}_1)\end{Bmatrix} \xrightarrow{\textbf{via } \text{connection}_1 \textbf{ send } \text{value}_1} \textbf{compose}\{\text{constraint}_{0..n} \textbf{ and } \text{behavior}_1\}}$$

Input:

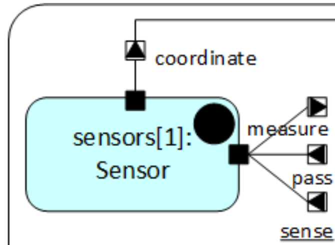
$$\frac{}{\textbf{compose}\begin{Bmatrix}\text{constraint}_{0..n} \\ \textbf{and } (\textbf{via } \text{connection}_1 \textbf{ receive } \text{value} . \text{behavior}_1)\end{Bmatrix} \xrightarrow{\textbf{via } \text{connection}_1 \textbf{ receive } \text{value}_1} \textbf{compose}\begin{Bmatrix}\text{constraint}_{0..n} \textbf{ and } (\text{value} = \text{value}_1) \\ \textbf{and } \text{behavior}_1\end{Bmatrix}}$$

where ($\text{constraint}_{0..n}$ **and** ($\text{value} = \text{value}_1$)) *is consistent, i.e. binding* ($\text{value} = \text{value}_1$) *can be consistently asserted together with* $\text{constraint}_{0..n}$

- **Communication**

```
Sensors[1] : system Sensor(lps=Coordinate::(10,10)) is { …
  behavior sensing is {
     value sensorcoordinate is Coordinate = lps
     tell sensorlocation is {sensorcoordinate = lps}
     via location::coordinate send sensorcoordinate
     via energy::threshold receive powerthreshold
     repeat {
        via energy::power receive powerlevel
        if (powerlevel > powerthreshold) then {
           tell powering is {powerlevel > powerthreshold}
           choose{
              via measurement::sense receive data
              via measurement::measure send
                 tuple{coordinate=lps,depth=data::convert()}
           } or {
              via measurement::pass receive data
              via measurement::measure send data
           }
        }
     }
  }
}
```

```
transmitters[1] : mediator
  Transmitter(distancebetweengates:Distance) is { …
   behavior transmitting is {
      via location::fromCoordinate receive sendercoordinate
      via location::toCoordinate receive receivercoordinate
      ask sendercoordinate::distance(receivercoordinate)
         < distancebetweengates
      repeat {
            via transmit::fromSensors receive measure
            via transmit::towardsGateway send measure
      }
    }
  }
```



- **Equality from communication**

```
constraint {transmitters[1]::sendercoordinate = Coordinate::(10,10)}
```

- **Equality from coalition**

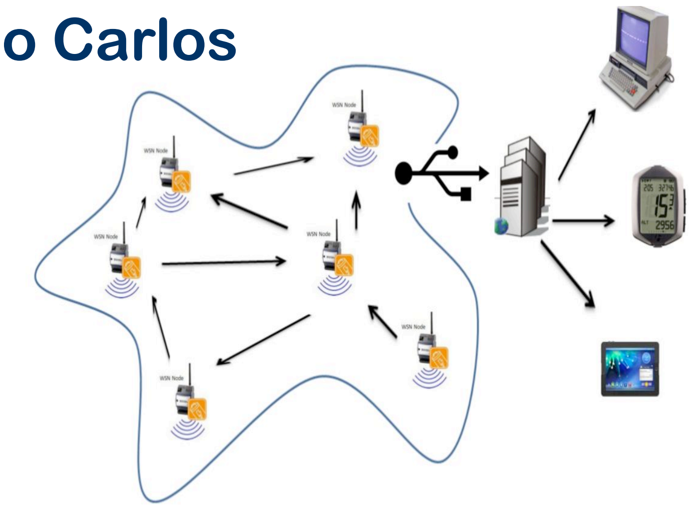
```
constraint {sensors[1]::location::coordinate = transmitters[1]::location::fromCoordinate}
```

Formal Definition of the π-Calculus for SoS

UMR IRISA

Validating the Formal Operational Semantics of SosADL: WSN-based Urban River Monitoring SoS

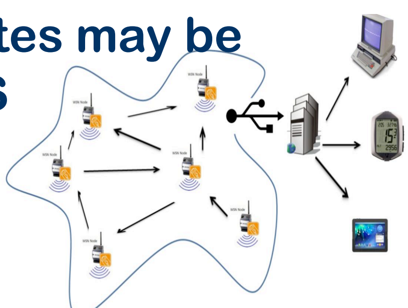
- Monjolinho river crossing the city of Sao Carlos



- The Urban River Monitoring SoS is based on two kinds of constituent systems:
  - wireless river sensors (for measuring river level depth via pressure physical sensing)
  - a gateway base station (for analyzing variations of river level depths and warning on the risk of flash flood)

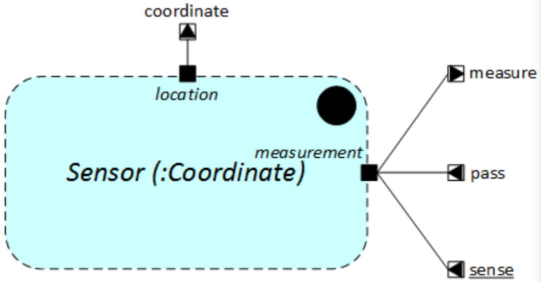
Applying π-Calculus for SoS: Urban River Monitoring

- Sensor motes are operated by different City Councils in the Urban area
- Operational independence of constituent systems
  - Each sensor mote operates in a way that is independent of other sensor motes (which may belong to different organizations and have different missions, e.g. pollution control, water supply, …)
- Managerial independence of constituent systems
  - Each sensor mote has its own strategy for transmission vs. energy consumption
- Geographical distribution of constituent systems
  - Sensor motes are geographically distributed along the river
- Evolutionary development of system-of-systems
  - New sensor motes may be installed, existing sensor motes may be changed or uninstalled without any control from the SoS
- Emergent behavior of system-of-systems
  - Sensor motes together, with the gateway, will make emerge the behavior of flood detection

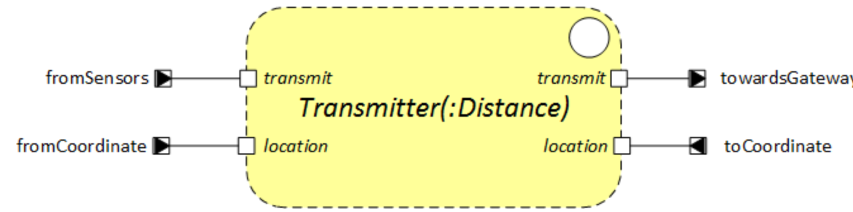


Illustrating the Formal Operational Semantics of SosADL: WSN-based Urban River Monitoring SoS

```
system Sensor(lps: Coordinate) is { …
  behavior sensing is {
     value sensorcoordinate is Coordinate = lps
     tell sensorlocation is {sensorcoordinate = lps}
     via location::coordinate send sensorcoordinate
     via energy::threshold receive powerthreshold
     repeat {
        via energy::power receive powerlevel
        if (powerlevel > powerthreshold) then {
           tell powering is {powerlevel > powerthreshold}
           choose{
              via measurement::sense receive data
              via measurement::measure send
                 tuple{coordinate=lps,depth=data::convert()}
           } or {
              via measurement::pass receive data
              via measurement::measure send data
           }
        }
     }
  }
}
```

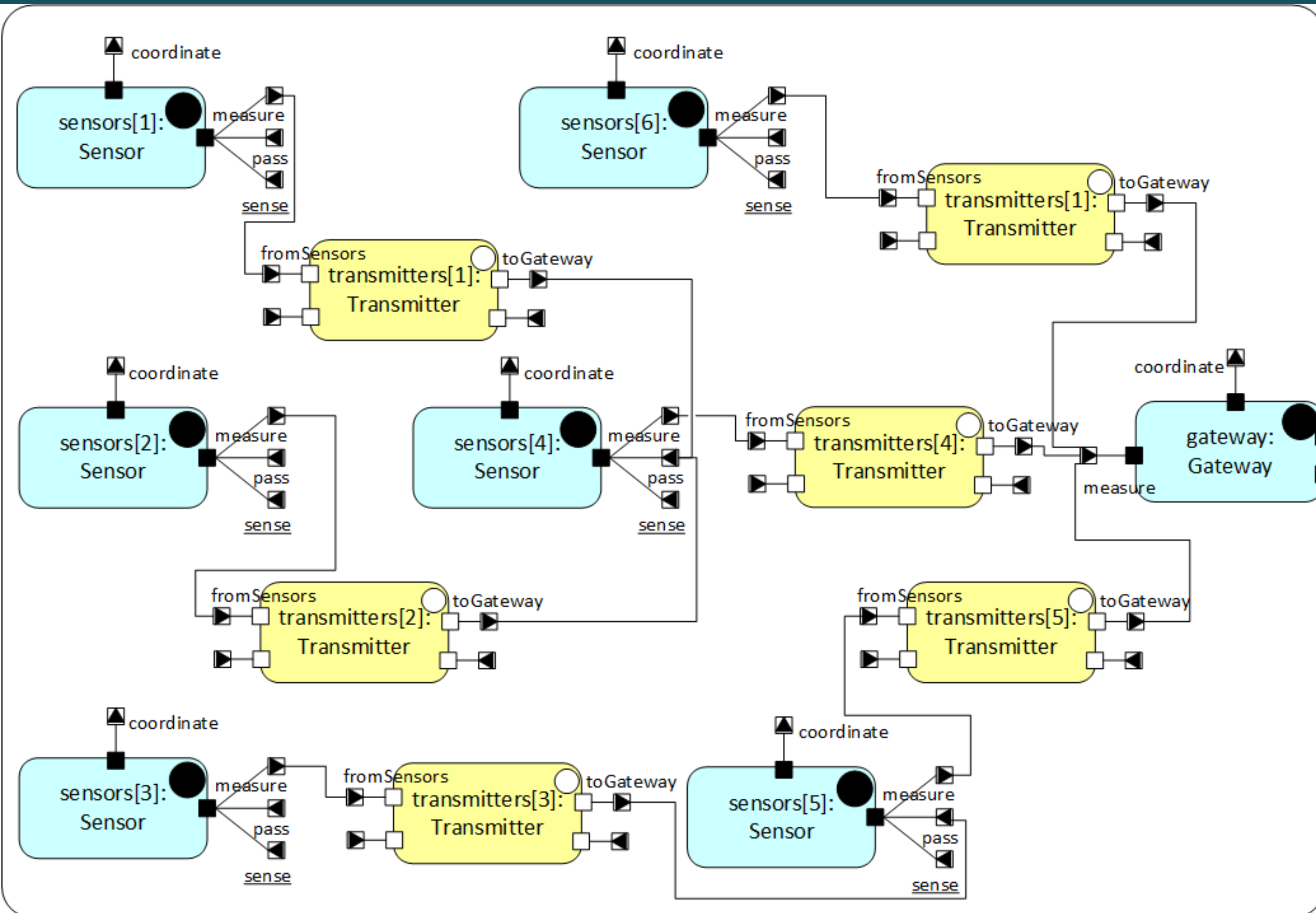


Illustrating the Formal Operational Semantics of SosADL: WSN-based Urban River Monitoring SoS



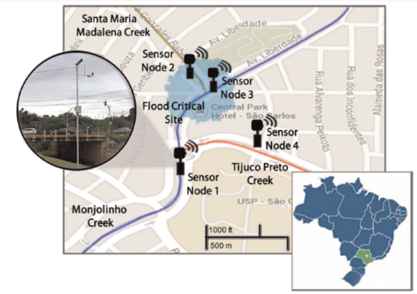

```
mediator Transmitter(distancebetweengates:Distance) is { …
  behavior transmitting is {
      via location::fromCoordinate receive sendercoordinate
      via location::toCoordinate receive receivercoordinate
      ask sendercoordinate::distance(receivercoordinate)
          < distancebetweengates
      repeat {
            via transmit::fromSensors receive measure
            via transmit::towardsGateway send measure
      }
  }
}
```

Urbain River Monitoring SoS Architecture: Concretion (snapshot)

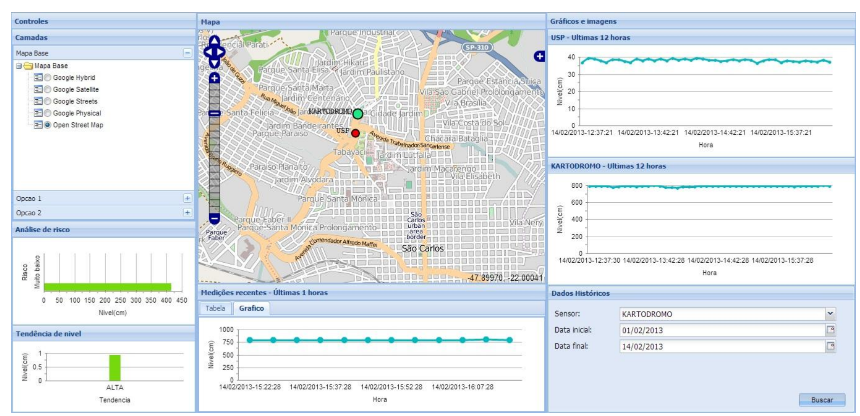


Validation through Real Application Cases

- Urban River Monitoring SoS
  - Monjolinho river crossing the city of Sao Carlos
    - XBee motes, ZigBee transmissions, Solar panels…

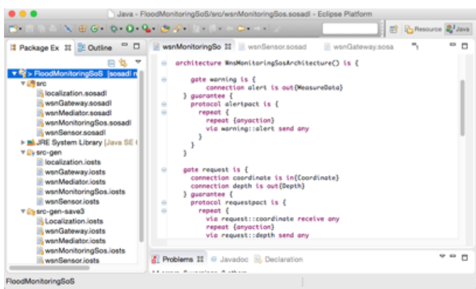


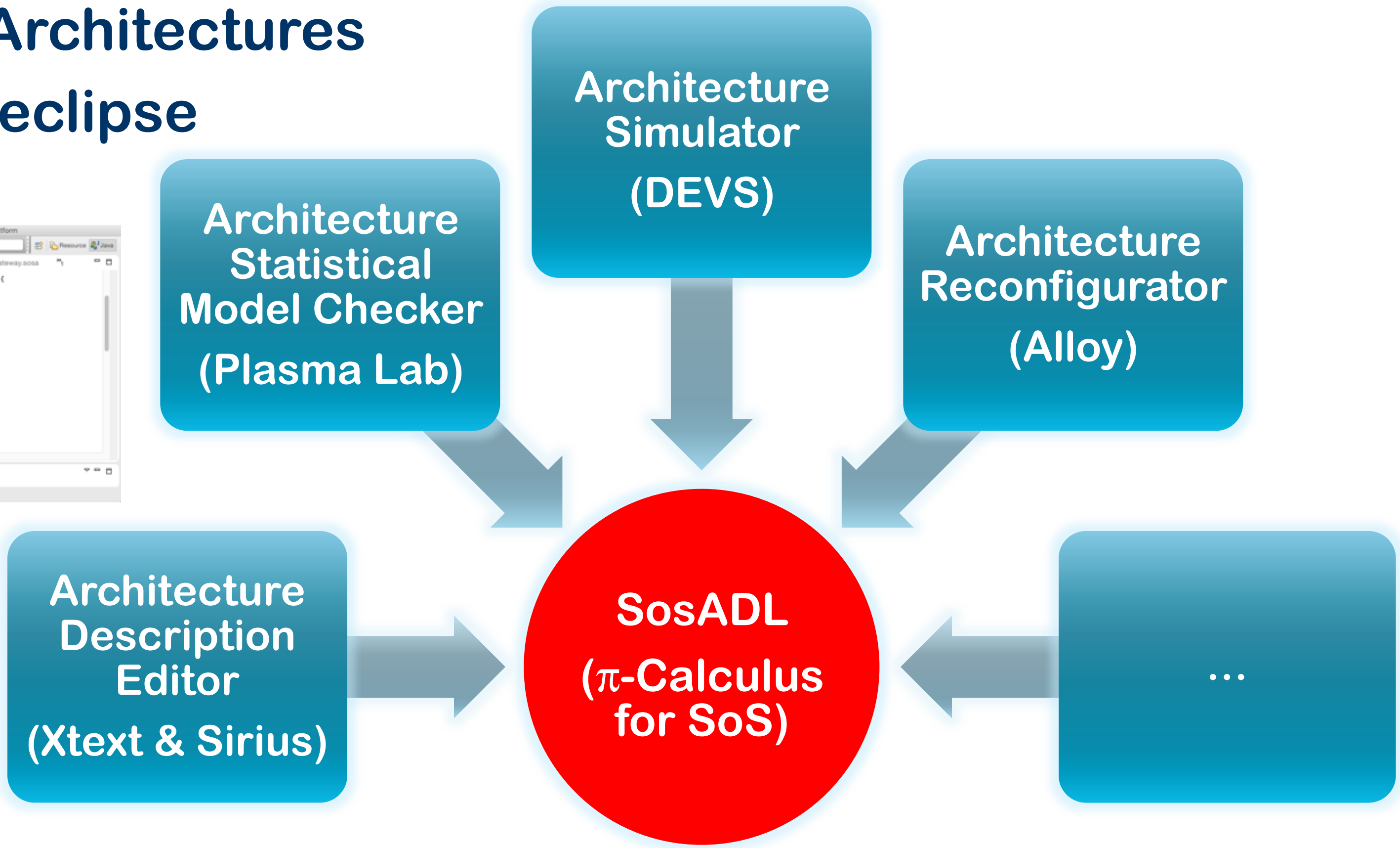
- Flood Monitoring and Emergency Response SoS
  - Wireless River Sensors
  - Telecommunication Gateways
  - Unmanned Aerial Vehicles (UAVs)
  - Vehicular Ad Hoc Networks (VANETs)
  - Meteorological Centers
  - Fire and Rescue Services
  - Hospital Centers
  - Police Departments
  - Short Message Service Centers
  - Social Networks



Toolset for π-Calculus for SoS

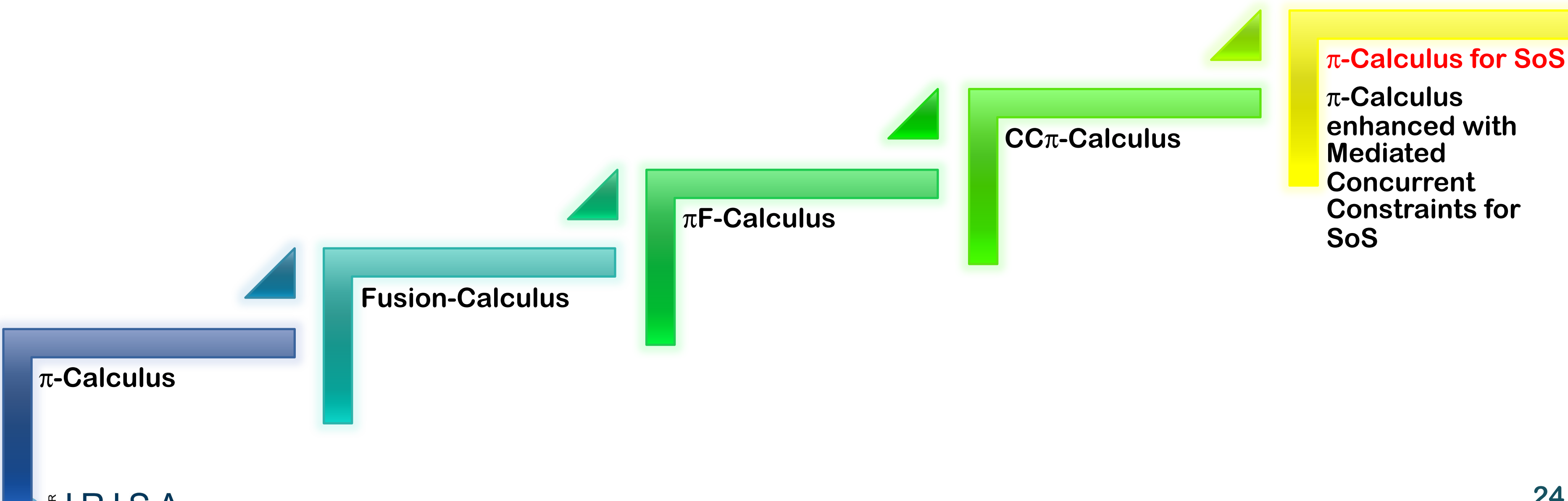
- SosADE (SoS Architecture Development Environment) for supporting the application of SosADL based on the π-Calculus for SoS for description and analysis of SoS Software Architectures
  - Plugins eclipse





Conclusion: Novel π-Calculus coping with SoS needs

- π-Calculus for SoS
  - Enhances the expressiveness of the π-Calculus with Mediated Concurrent Constraints for coping with SoS characteristics
    - exogenous, intentional, constrained and mediated channel bindings subject to uncertainty
  - Provides a novel π-Calculus as formal foundation for SosADL



Conclusion

Conclusion: Novel π-Calculus coping with SoS needs

- **π-Calculus for SoS** provides a formal foundation having the expressiveness to address the challenge of describing architectures of Software-intensive SoSs
  - The π-Calculus for SoS supports **automated verification of correctness properties of SoS architectures**
  - The π-Calculus for SoS supports **validation through executable specifications**
    - Including **simulation to validate and discover emergent behaviors**
- π-Calculus for SoS provided the formal foundation of a novel ADL for SoS: SosADL
- It was applied for architecting a Flood Monitoring and Emergency Response SoS in the Monjolinho river crossing the City of Sao Carlos
- Several new applications are on the way with DCNS, IBM, ICMC, SEGULA… for formal modeling SoS Architectures

Thank You

Questions?

The π-Calculus for SoS:

Novel π-Calculus for the Formal Modeling of Software-intensive Systems-of-Systems

Flavio Oquendo

flavio.oquendo@irisa.fr

http://people.irisa.fr/Flavio.Oquendo/