The $\pi$-Calculus for SoS:
Novel $\pi$-Calculus for the Formal Modeling of Software-intensive Systems-of-Systems

Flavio Oquendo
flavio.oquendo@irisa.fr
http://people.irisa.fr/Flavio.Oquendo/
Introduction: Motivation to conceive the $\pi$-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 $\pi$-Calculus for SoS
- Novel process calculus meeting SoS needs: The $\pi$-Calculus for SoS

Formal Definition of the $\pi$-Calculus for SoS
- Formal transition system defining the $\pi$-Calculus for SoS

Validating the Formal Operational Semantics of the $\pi$-Calculus for SoS

Conclusion
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
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
  - $\pi$-Calculus: the formal foundation of $\pi$-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 $\pi$-Calculus (ACM Turing Award for Robin Milner in 1991)
Formal Foundations of ADLs for Single Systems: The $\pi$-Calculus

- **$\pi$-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)
  
  - $\pi$-Calculus has shown to be a suitable formal foundation for describing and analyzing the architecture of software-intensive single systems

- However, $\pi$-Calculus as well as other process calculi, e.g. FSP/CSP, are too limited to cope with SoS architecture needs
Different process calculi were applied for formally describing the architecture of single software-intensive systems

- Including different variants of the $\pi$-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 $\pi$-Calculus for formally describing SoS architectures?
None of the existing $\pi$-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 $\pi$-Calculus binding is endogenous
  - In SoS, the binding must be constrained by local contexts
    - Problem: In the $\pi$-Calculus binding is unconstrained
  - In SoS, the binding between channels must be intentional
    - Problem: In the $\pi$-Calculus binding is extensional
  - In SoS, the binding between channels must be mediated
    - Problem: In the $\pi$-Calculus binding is unmediated
Formal Approach for Describing SoS Architectures: The $\pi$-Calculus for SoS

- **Design decisions for the $\pi$-Calculus for SoS**
  - Generalization of the $\pi$-Calculus with mediated constraints
    - Subsuming the original $\pi$-Calculus
    - Coping with uncertainty
      - In SoS, partial information contributes to uncertainty, in addition to the uncertainty of emergent behavior
  - Definition of an enhanced $\pi$-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

- **$\pi$-Calculus for SoS**
The $\pi$-Calculus for SoS: meeting the needs of SoS architecture description

- the $\pi$-Calculus for SoS generalizes the $\pi$-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 $\pi$-Calculus for SoS

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

Transition rule:

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

where side conditions
Formal Semantics of Actions in the $\pi$-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 $\pi$-Calculus for SoS: labeled transition rules for actions**

**Output:**

$\text{compose} \{\text{constraint}_0 . \text{n} \land (\text{via} \\text{connection}_1 \\text{send} \text{value}_1 . \text{behavior}_1)\} \xrightarrow{\text{via} \\text{connection}_1 \text{send} \text{value}_1} \text{compose} \{\text{constraint}_0 . \text{n} \land \text{behavior}_1\}$

**Input:**

$\text{compose} \{\text{constraint}_0 . \text{n} \land (\text{via} \\text{connection}_1 \text{receive} \text{value}_1 . \text{behavior}_1)\} \xrightarrow{\text{via} \\text{connection}_1 \text{receive} \text{value}_1} \text{compose} \{\text{constraint}_0 . \text{n} \land \text{behavior}_1\}$

where (constraint$\text{m}_0 . \text{n}$ and (value = value$1$)) is consistent, i.e. binding (value = value$1$) can be consistently asserted together with constraint$\text{m}_0 . \text{n}$

**Unobservable:**

$\text{compose} \{\text{constraint}_0 . \text{n} \land (\text{unobservable} . \text{behavior}_1)\} \xrightarrow{\tau} \text{compose} \{\text{constraint}_0 . \text{n} \land \text{behavior}_1\}$

**Tell:**

$\text{compose} \{\text{constraint}_0 . \text{m} \land (\text{tell} \text{constraint}_n . \text{behavior}_1)\} \xrightarrow{\tau} \text{compose} \{\text{constraint}_0 . \text{m} \land \text{constraint}_n \land \text{behavior}_1\}$

where (constraint$\text{m}_0 . \text{m}$ and constraint$\text{m}_n$) is consistent, i.e. constraint$\text{m}_n$ can be consistently asserted with constraint$\text{m}_0 . \text{m}$

**Untell:**

$\text{compose} \{\text{constraint}_0 . \text{n} \land (\text{untell} \text{constraint}_\text{m} . \text{behavior}_1)\} \xrightarrow{\tau} \text{compose} \{(\text{constraint}_0 . \text{n} \land \text{constraint}_\text{m}) \land \text{behavior}_1\}$

where (constraint$\text{m}_0 . \text{n}$ − constraint$\text{m}_\text{m}$) is consistent, i.e. constraint$\text{m}_\text{m}$ can be consistently retracted from constraint$\text{m}_0 . \text{n}$

**Check:**

$\text{compose} \{\text{constraint}_0 . \text{n} \land (\text{check} \text{constraint}_\text{m} . \text{behavior}_1)\} \xrightarrow{\tau} \text{compose} \{\text{constraint}_0 . \text{n} \land \text{behavior}_1\}$

where (constraint$\text{m}_0 . \text{n}$ and constraint$\text{m}_\text{m}$) is consistent, i.e. constraint$\text{m}_\text{m}$ is checked to be consistent with constraint$\text{m}_0 . \text{n}$

**Ask:**

$\text{compose} \{\text{constraint}_0 . \text{m} \land (\text{ask} \text{constraint}_n . \text{behavior}_1)\} \xrightarrow{\tau} \text{compose} \{\text{constraint}_0 . \text{m} \land \text{behavior}_1\}$

where constraint$\text{m}_0 . \text{m} \vdash$ constraint$\text{n}$, i.e. constraint$\text{n}$ can be derived from constraint$\text{m}_0 . \text{m}$
Formal Semantics of Behaviors in $\pi$-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 $\pi$-Calculus for SoS: labeled transition rules for behaviors

Restriction:
- $\text{constrainedBehavior}_1 \xrightarrow{\text{action}} \text{constrainedBehavior}_2$
- $\text{value} \text{value}_1 \cdot \text{constrainedBehavior}_1 \xrightarrow{\text{action}} \text{value} \text{value}_1 \cdot \text{constrainedBehavior}_1$
- where $\text{value}_1 \notin \text{names(\text{action}_1)}$, i.e. $\text{value}_1$ is not among the names used in $\text{action}_1$

Communication:
- $\text{behavior}_1 \xrightarrow{\text{via connection: send value}} \text{behavior}_1'$
- $\text{behavior}_2 \xrightarrow{\text{via connection: receive value}} \text{behavior}_2'$
- $\text{compose} \text{and (connection}_1 = \text{connection}_2) \xrightarrow{\text{action}} \text{compose} \text{and (connection}_1 = \text{connection}_2)$
- $\text{and (behavior}_1 \text{ and behavior}_2)$
- $\text{and (value = value}_1 \text{ and behavior}_1' \text{ and behavior}_2')$
- where $\text{connection}_1 = \text{connection}_2$, i.e. (connection$=\text{connection}_2$) is a binding resulting from an extrusion or unification

Restriction-Open:
- $\text{constrainedBehavior}_1 \xrightarrow{\text{via connection: send value}} \text{constrainedBehavior}_1'$
- $\text{value} \text{value}_1 \cdot \text{constrainedBehavior}_1 \xrightarrow{\text{via connection: send value}} \text{constrainedBehavior}_1'$
- where $\text{value}_1 \notin \text{connection}_1$, i.e. $\text{value}_1$ cannot be used for connection as it is restricted

Communication-Close:
- $\text{behavior}_1 \text{value} \text{connection} \xrightarrow{\text{via connection: send connection}} \text{behavior}_1'$
- $\text{behavior}_2 \xrightarrow{\text{via connection: receive value}} \text{behavior}_2'$
- $\text{compose} \text{and (connection}_1 = \text{connection}_2) \xrightarrow{\text{action}} \text{compose} \text{and (connection}_1 = \text{connection}_2)$
- $\text{and (behavior}_1 \text{ and behavior}_2)$
- $\text{and (value = connection}_1 \text{ and behavior}_1' \text{ and behavior}_2')$
- where $\text{value}_1 \notin \text{free(behavior}_2)$, i.e. $\text{value}_1$ is not restricted in $\text{behavior}_2$ while connection is restricted in $\text{behavior}_1$

Choice:
- $\text{constraint}_0 \text{and (action}_i \text{. behavior}_i) \xrightarrow{\text{action}_i} \text{constraint}_0 \text{and behavior}_i$
- $\text{compose} \text{and choose (action}_1 \text{. behavior}_1 \text{ or action}_n \text{. behavior}_n) \xrightarrow{\text{action}_i} \text{compose} \text{and behavior}_i$
- where $\text{constraint}_0 \in \text{0..n}$, i.e. only one of the actions $\text{action}_0 \text{..n}$ is performed

Conditional-Then:
- $\text{behavior}_1 \text{action}_i \xrightarrow{\text{action}_i} \text{behavior}_1'$
- $\text{constraint} = \text{true}$
- $\text{compose} \{\text{constraint}_0 \text{and (if constraint then behavior}_1 \text{else behavior}_2}) \xrightarrow{\text{action}_i} \text{compose} \{\text{constraint}_0 \text{and behavior}_1$

Conditional-Else:
- $\text{behavior}_2 \xrightarrow{\text{action}_i} \text{behavior}_2'$
- $\text{constraint} = \text{false}$
- $\text{compose} \{\text{constraint}_0 \text{and (if constraint then behavior}_1 \text{else behavior}_2}) \xrightarrow{\text{action}_i} \text{compose} \{\text{constraint}_0 \text{and behavior}_2$

Repetition:
- $\text{behavior}_1 \text{action}_i \xrightarrow{\text{action}_i} \text{behavior}_1'$
- $\text{repeat} \{\text{behavior}_1} \xrightarrow{\text{action}_i} \text{behavior}_1' \text{. repeat} \{\text{behavior}_1$
- where $\text{behavior}_1'$, $\text{behavior}_1$ is a sequential composition, i.e. $\text{behavior}_1'$ must be performed before $\text{behavior}_1$

Composition:
- $\text{constrainedBehavior}_1 \xrightarrow{\text{action}_i} \text{constrainedBehavior}_1'$
- $\text{compose} \text{and constrainedBehavior}_1 \xrightarrow{\text{action}_i} \text{compose} \text{and constrainedBehavior}_1'$
- $\text{and constrainedBehavior}_1 \xrightarrow{\text{action}_i} \text{and constrainedBehavior}_1'$
- $\text{and constrainedBehavior}_1 \xrightarrow{\text{action}_i} \text{and constrainedBehavior}_1'$
- $\text{where} \text{constraint}_0 \in \text{1..n and bound(\text{action}) \cap free(\text{constrainedBehavior}_0 \text{. n . i}) = \emptyset}$, i.e. restricted names in action are not restricted elsewhere
## 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
                }
            }
        }
    }
}

Transmitter[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 \( \pi \)-Calculus for SoS

### Communication

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

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

### Equality from communication

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

### Equality from coalition

\[
\text{constraint} \{ \text{sensors[1]::location::coordinate} = \text{transmitters[1]::location::fromCoordinate} \}
\]
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)
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
        }
      }
    }
  }
}
mediator Transmitter(distancebetweenengates:Distance) is 
  behavior transmitting is 
    via location::fromCoordinate receive sendercoordinate 
    via location::toCoordinate receive receivercoordinate 
    ask sendercoordinate::distance(receivercoordinate) < distancebetweenengates 
    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 $\pi$-Calculus for SoS

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

![Diagram showing the relationships between SosADE, Architecture Description Editor, Architecture Statistical Model Checker, Architecture Simulator, Architecture Reconfigurator, and SosADL.](Image)
Conclusion: Novel $\pi$-Calculus coping with SoS needs

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

- $\pi$-Calculus for SoS provides a formal foundation having the expressiveness to address the challenge of describing architectures of Software-intensive SoSs
  - The $\pi$-Calculus for SoS supports automated verification of correctness properties of SoS architectures
  - The $\pi$-Calculus for SoS supports validation through executable specifications
    - Including simulation to validate and discover emergent behaviors
- $\pi$-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 $\pi$-Calculus for SoS: Novel $\pi$-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)