
Flavio Oquendo
flavio.oquendo@irisa.fr
http://people.irisa.fr/Flavio.Oquendo/

ECSA 2023
Mon 18 - Fri 22 September 2023
Yeditepe University, Istanbul, Turkey.

Assistant/Associate Professor
University of Grenoble, France

M.Sc., Ph.D., and H.D.R.
from Univ. of Grenoble, France

B.Eng. from ITA, São José dos Campos, SP, Brazil

R&D Engineer at G.I.E. Emeraude [R&D group on advanced Software Engineering created by BULL, THALES IS, AT&T Europe], Paris, France

Full Professor from 1995 to 2005 at the University of Savoy, and since 2005 at the University of South Brittany, France

Research Direction on Software Architecture / Distinguished Full Professor (Professeur des universités de classe exceptionnelle)
IRISA – The Computer Science Research Institute of Brittany, France

850 Research Staff
- 350 Research Academics
- 350 PhD Students & Post-docs
- 100 Supporting Staff

http://www.irisa.fr/en

I. Key notions and problematics of architecting Software-intensive Systems-of-systems (SoS)

II. Proposed solution for architecting SoS raising Collective Intelligence (CI)

III. Enhancing the proposed solution for raising Collective Intelligence under Uncertainty

IV. Summing up and takeaway message
I. Key Notions and Problematics of architecting Software-intensive Systems-of-systems (SoS)
1. What are Software-intensive Systems-of-Systems (SoS)
1.1a The advent of Software-intensive Systems-of-Systems: From Single Systems (SiS) to Systems-of-Systems (SoS)

- The past and present of Software-intensive Systems
  - They were stand-alone single systems in the past
  - They are often part of connected and automated systems in the present

- Systems (with no software)
- Systems (assisted by software)
- Systems (automated by software)
- Connected systems
  - Connected and automated driving systems
  - Automated driving systems
  - Advanced driver-assistance systems
The upcoming future of Software-intensive Systems

- They are increasingly becoming systems of systems in the upcoming future
  - A system whose constituents are systems and which are developed to achieve missions not possible by a constituent system alone
    - e.g. Vehicle platooning

Source: https://theconversation.com/coming-soon-to-a-highway-near-you-truck-platooning-87748
A System-of-Systems (SoS) is a system that delivers unique capabilities formed by integrating independent useful systems [ISO/IEC/IEEE 24765:2010]

- In its more evolved form, it can be perceived as a complex system in which its constituents, i.e. themselves systems, are possibly discovered, selected, and composed at run-time to fulfill a specific mission.

- Note that constituent systems fulfill valid purposes in their own right and continue to operate to fulfill those purposes if disassembled from the encompassing SoS.

- They are managed, in part, for their own purposes rather than the purposes of the whole.
1.2b Defining Characteristics of Systems-of-Systems (SoS)

- Characteristics of SoS constituents
  1. Operational independence
  2. Managerial independence
  3. Geographical distribution

- Characteristics of an SoS as a whole
  4. Evolutionary development
  5. Emergent behavior

- Characteristics of SoS operational environments
  - Uncertainty

- SoS by its very nature has characteristics that are hard to address, in particular, emergent behavior
1.3 The Importance of Architecting Software-intensive Systems-of-Systems (SoS)

- Architecting Software-intensive Systems-of-Systems has been identified as a major issue in the upcoming years
  - Systems Engineering Vision 2035 by the International Council on Systems Engineering (INCOSE)
  - Today, SoS architecting practices are often ad-hoc and do not effectively integrate required architectural concerns
  - By 2035, SoS architecting practices shall enable to master the complexity of SoS architectural design and evolutionary development, achieving open and resilient SoS
As stated, our lives and livelihoods increasingly depend on Software-intensive Systems-of-Systems.

- Systems-of-Systems being by definition complex systems, they need to be architected before engineered.
  - Complexity poses the need for separation of concerns between architecture and engineering.

Remember that:

- Architecture: focus on reasoning about interactions of parts and their emergent properties.
- Engineering: focus on designing and building the specified architecture.
How would you (as an architect) describe the architecture of a software-intensive system-of-systems such as the one of the vehicle platoon presented in this video? (from a technology-independent viewpoint)
2. What is Collective Intelligence (CI)
2.1 Artificial Intelligence: Individual vs Collective Intelligence

- There are different forms of Artificial Intelligence

- Artificial ‘Individual’ Intelligence in Single Systems
  - It is the notion of embedding AI-enabled capabilities in an individual system, e.g. an autonomous drone

- Artificial ‘Collective’ Intelligence in Systems-of-Systems (a.k.a. Swarm Intelligence)
  - It is the notion that a group of “single” systems operating in concert can operate as a collective intelligence with superior capabilities to any of the individual systems, e.g. satellite constellations

Collective Intelligence (CI) refers to the intelligence that emerges at the macro-level of a heterogeneous or homogenous collection of individuals and transcends that of the individuals.

- Collective intelligence is a kind of emergent property in which the interactions of single systems acting together produce an overall capability that exceeds that of the individuals.
- In SoS, its main purpose is for coordination, cooperation, and collaboration, rather than cognition.

Collective Intelligence

- Cognition
- Coordination
- Cooperation
- Collaboration
2.2b The Notion of Collective Intelligence (CI)

- In Software-intensive Systems-of-Systems, Collective Intelligence is the result of emergent behavior, in general based on self-organization
  - What is Emergence and in particular emergent behavior?
  - What is Self-Organization and how it supports emergent behavior?
What is exactly an Emergent Behavior

- An emergent behavior is a global behavior that arises out of the interactions between parts of a whole and which cannot be predicted or extrapolated from the behavior of the individual parts.

- Stated simply: “The behavior of the whole is more (other) than the sum of the behaviors of its parts.”

- It arrives mostly in collective behaviors of groups of elementary constituent systems.

- Emergent behavior produces Collective Intelligence.
Emergence is a phenomenon occurring in Physics, in Chemistry, in Biology, in Sociology, in Engineering …

The nature of the relation between the whole and its parts (term coined by George Henry Lewes, 1875)

- **Aggregation**: properties of a whole that can be calculated knowing its parts, e.g. the weight of a table given its parts
- **Emergence**: properties of a whole that cannot be calculated knowing the parts of the whole, e.g. it is not possible to predict the properties of water, knowing the properties of the atoms of hydrogen and oxygen
  - studied since at least the time of Aristotle, 384 BC-322 BC

The meaning of **emergence** is of ‘unexpected’ properties that arise out of more fundamental properties and yet are ‘irreducible’ with respect to them

- In contrast, aggregative properties are always reducible to its fundamental entities, e.g. weight, spatial position and speed in physics
2.4a The Notion of Self-Organization

- Self-organization is a phenomenon occurring in Physics, in Chemistry, in Biology, in Sociology, in Engineering ...

- What is exactly Self-Organization?
  - Self-organization is a pattern formation behavior that arises when some form of overall order emerges from local interactions between parts of an initially disordered system

- Examples of Self-Organization:
  - Flock of drones
  - Swarm of robots
2.4b The Theory of Self-Organization

- The General Theory of Self-Organization
  - By Ilya Prigogine (Nobel Laureate in Chemistry, 1977)
  - Self-organization is spontaneous in dissipative structures

TheEDGEofChaos

what is collective intelligence: self-organization

Collective Intelligence is the result of emergence, in general based on self-organization.

How would you (as an architect) describe the architecture of a vehicle platoon in a way that enables raising the collective intelligence of platooning? (from a technology-independent viewpoint)

- Relying on emergence and self-organization.
II. Proposed solution for architecting Software-intensive Systems-of-systems (SoS) raising Collective Intelligence (CI)
3. How to achieve Collective Intelligence through Emergence and Self-Organization in Software-intensive Systems-of-Systems (SoS)
3.1 Architecting Emergence in SoS

- **Question:** How to describe an SoS architecture in terms of interacting constituents for enabling emergence?

- **Proposed approach:** Supervenience
  - Supervenience is the trans-ordinal relation that is used to describe emergence where the macro-scale properties are determined by its micro-scale properties, i.e. the emergent macro-properties supervene on the micro-properties.

- To describe emergent behavior in an SoS architecture, we need to define micro-scale behaviors that by supervenience (upward causation) will form the required macro-scale emergent behavior.

- The proposed approach supports systemic emergence.
3.2 Architecting Systemic Emergence in SoS through Mediators

- **Systemic emergence** in complex systems is a systemic emergent property that arises from the interaction of constituents, among themselves as well as with their operational environment.
- The proposed approach to support systemic emergence in SoS relies on the development of mediations among constituent systems.
- The SoS has **total control** over mediators:
  - Mediators are created, may evolve or may be discarded at run-time.
  - Mediators are only known by the SoS: they provide duties for constituent systems.
  - Mediators may enable communication, coordination, cooperation, and collaboration.
3.3 Supporting Self-Organization with Concurrent Constraints

- **Question:** How to describe self-organized SoS architectures for supporting systemic emergence?
- **Based on the General Theory of Self-Organization,** SoS are perceived as complex systems far from equilibrium when compared to single systems that are essentially systems near equilibrium.
  - To reach a valid SoS architecture, it is necessary to reach an “attractor”
    - An “attractor” is a set of states toward which a system tends to evolve.
  - To reach an “attractor” it is needed to tie or relax constraints in dissipative structures, until the “attractor” is achieved.
- **Proposed approach for self-organization is based on:**
  - The mechanism of concurrent constraints: To tell or untell constraints to make the order of the SoS constituents vary for reaching a target attractor.
  - The architectural concept of mediator: mediators permit to tie or relax behaviors of constituent systems through concurrent constraints, managing the degree of freedom of these constituent systems.
4. How to architecturally enforce Supervenience based on Concurrent Constraints in Software-intensive Systems-of-Systems (SoS)
4.1 Defining a novel ADL for SoS based on the conceived Architectural Emergentist Framework

- **SosADL**: novel ADL for SoS
- SoS architecture description from different viewpoints:
  - **Structure**
  - **Behavior**
    - Including systemic emergence via supervenience
  - Self-organization is obtained by constraining structure and behavior in supervenience

SoS architecture description in terms of:
- **Constituent systems**
  - locus of operational capabilities for enabling functionalities
- **Mediators**
  - locus of interaction capabilities for enabling emergent behavior
- **Coalition**
  - composition of constituent systems coordinated by mediators for fulling specified missions
    - driven by specified constraints
4.2a Defining a Novel Formal Foundation for SosADL based on Process Algebra

- The formal definition of SosADL is grounded in the $\pi$-Calculus for SoS
- $\pi$-Calculus for SoS generalizes the $\pi$-Calculus with the notion of concurrent behaviors with concurrent constraints

---


---

Abstract syntax of behaviors

constrainedBehavior ::= behavior,
  | valuing, constrainedBehavior,
  | behavior name, is { behavior, }
  | constraint name, is { constraint, }
  | compose { constrainedBehavior, ... and constrainedBehavior, }

behavior ::= baseBehavior,
  | valuing, behavior,
  | repeat { behavior, }
  | apply name, ( value, ..., value, )
  | compose { behavior, ... and behavior, }

baseBehavior ::= action, behavior,
  | choose { action, baseBehavior, 
    | or action, baseBehavior, ... or action, baseBehavior, }
  | if constraint, then { baseBehavior, } else { baseBehavior, }
  | done

action ::= baseAction,
  | tell constraint,
  | untell constraint,
  | check constraint,
  | ask constraint,

baseAction ::= via connection, send value,
  | via connection, receive name, : type,
  | do internalAction,
  | unobservable

collection ::= name, name, ..., name,

valuing ::= value name, is type, = value, 
  | typing

typing ::= datatype name, is type, { function, ... and function, }
4.2b Defining a Novel Formal Foundation for SosADL based on Process Algebra

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
- **send** value via connection
- **receive** value via connection
- **unobservable** internal actions

Transition rule:
\[
P_1 \xrightarrow{\alpha_1} P_1' \ldots P_n \xrightarrow{\alpha_n} P_n' \xrightarrow{C} C'
\]
**where** side conditions

**Flavio Oquendo**

**Formal semantics of \(\pi\)-Calculus for SoS: labeled transition rules for actions**

**Output:**
\[
\text{compose } \{\text{constraint}_n \text{ and } (\text{via connection} \rightarrow \text{send value})\} \xrightarrow{\text{via connection}} \text{send value} \xrightarrow{\text{compose}} \{\text{constraint}_n \text{ and behavior}\}
\]

**Input:**
\[
\text{compose } \{\text{constraint}_n \text{ and } (\text{via connection} \rightarrow \text{receive value})\} \xrightarrow{\text{via connection}} \text{receive value} \xrightarrow{\text{compose}} \{\text{constraint}_n \text{ and behavior}\}
\]

**Unobservable:**
\[
\text{compose } \{\text{constraint}_n \text{ and (unobservable)} \text{ behavior}\} \xrightarrow{\text{compose}} \{\text{constraint}_n \text{ and behavior}\}
\]

**Tell:**
\[
\text{compose } \{\text{constraint}_m \text{ and (tell constraint) }} \text{behavior}\} \xrightarrow{\text{compose}} \{\text{constraint}_m \text{ and constraint}_n \text{ behavior}\}
\]

**Untell:**
\[
\text{compose } \{\text{constraint}_n \text{ and (untell constraint) }} \text{behavior}\} \xrightarrow{\text{compose}} \{\text{constraint}_n \text{ and constraint}_m \text{ behavior}\}
\]

**Check:**
\[
\text{compose } \{\text{constraint}_n \text{ and (check constraint) }} \text{behavior}\} \xrightarrow{\text{compose}} \{\text{constraint}_n \text{ and behavior}\}
\]

**Ask:**
\[
\text{compose } \{\text{constraint}_m \text{ and (ask constraint) }} \text{behavior}\} \xrightarrow{\text{compose}} \{\text{constraint}_m \text{ and behavior}\}
\]
5. The SosADL: Key Concepts for architecting Software-intensive Systems-of-Systems (SoS)
5.1 The Notion of Software Architecture of Software-intensive Systems-of-Systems

SoS Architecture: fundamental conception of an SoS in its environment, embodied in its constituents, their relationships to each other and to the environment, and principles guiding the SoS design and evolution.

[Adapted from ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description]

Stakeholders

System-of-Systems (SoS)

Mission

Operational Environment

SoS Architecture

SoS Architecture Description

SoS Architecture Description Language (SosADL)
5.2 SosADL: A specific-purpose Architecture Description Language (ADL) for SoS

- SosADL comprises a new paradigm for architecture description of Software-intensive Systems-of-Systems
  - Enables to describe SoS software architectures abstractly at design-time without knowing which will be the actual concrete systems in the SoS at run-time
  - Enables to describe emergent behavior of evolutionary developed SoS through self-organization
  - Captures the formal semantics of the behavior of SoS architecture descriptions
    - based on a formal foundation: $\pi$-Calculus for SoS
      - $\pi$-Calculus enhanced with concurrent constraints
5.3 SosADL: Conceptual Model of SoS Architecture Description
5.4 SosADL: Key Concepts in Software-intensive SoS Architecture Description

- An SoS ADL needs to cope with the issue that concrete constituent systems are in general not known at design-time, being only identified at run-time, and may evolve unexpectedly.

- **SosADL**: an ADL for Software-intensive SoS

- SosADL describe the Software Architecture of SoS in terms of:

  - **Constituent systems**: architectural components of SoS
    - Defined by intention (declaratively, abstractly)
    - Identified at run-time (concretized)

  - **Mediators**: architectural connectors of SoS
    - Defined by intention (declaratively, abstractly)
    - Created at run-time (concretized) to achieve a goal

  - **Coalition**: architectural configurations of SoS
    - Defined by intention (declaratively, abstractly)
    - Created at run-time (concretized) to achieve a mission
5.5 Difference between Systems Architecture and SoS Architecture described by SosADL

- Single system architectures are described by extension
- SoS architectures are described by intention
- Single system architectures are described at design-time
  - for developing the system based on design-time components
- SoS architectures are described at design-time
  - for developing the SoS based on discovered constituents at run-time
- Single system architectures generally evolve offline
- SoS architectures generally evolve online
- Single system architectures generally support evolution
- SoS architectures generally support coevolution

Single System Architecture vs SoS Architecture
Constituent systems of an SoS are specified by System Abstractions

The SoS has no control on concrete systems
- Concrete systems exist independently of the SoS
- Concrete systems are identified at run-time

Assumptions: assertions about the environment in which the system is placed and that are assumed through the specified gate

Guarantees: assertions derived from the assumptions and the behavior

Behavior that satisfies all gate assumptions (including protocols) of all gates

Proof obligation
5.6b The SosADL Constructs: SosADL: Mediators among Constituent Systems

- Mediators among constituent systems in an SoS are specified by Mediator Abstractions
- The SoS has total control over mediators
  - Mediators control the interactions among constituent systems at run-time
  - Mediators are only known by the SoS
    - They provide duties for constituent systems

- Mediators enable
  - Communication, Coordination, Cooperation, Collaboration

- Assumptions: assertions about the gate of the system to be connected to fulfill the duty
- Guarantees: assertions derived from the assumptions and the behavior
- Behavior that satisfies all duty assumptions (including protocols) of all duties
- Proof obligation
5.6c The SosADL Constructs: Coalitions of Mediated Systems

- System-of-Systems are specified by **Coalition Abstractions**
- An SoS is abstractly defined in terms of coalitions of systems connected by mediators

- **SosADL**: Formal Architecture Description Language for SoS

- SoS are concretized and evolve dynamically at runtime
  - Coalitions are defined by policies for operation and evolution

**Assumptions**: assertions about the environment in which the SoS is placed and that are assumed through the specified gate

**Guarantees**: assertions derived from the assumptions and the coalition

**Missions** are achieved through the emergent behavior of coalitions

**Coalition of mediated systems** that satisfies all gate assumptions (including protocols) of all gates

**Proof of obligation**
6.1 SosADL Studio: Formal Architectural Development of SoS using SosADL

- **SosADL Studio** supports the application of SosADL for description and analysis of SoS architectures.

- A concrete SoS architecture is guaranteed as being **well-formed** and **well-behaved** with guaranteed emergent behavior under the assumptions of the abstract SoS architecture description and the given operational environment.
Question: What are the guaranteed properties of an SoS architecture when applying SosADL, using SosADL Studio?

Proposed solution for enforcing guarantees of correctness:

- **Formal type system and mechanisms for type checking**
  - Proof-carrying code techniques for ensuring correctness with respect to the mechanized type system
  - Machine-checkable proof that the resulting outcome conforms to the type system (verified using the Coq proof assistant and Gallina/Vernacular as language for proofs)

- **Formal behavior system and mechanisms for model checking**
  - Assume-guarantee properties are verified by model checking, with Uppaal
    - Properties are expressed in Timed CTL temporal logics
    - Assume-guarantees asserted in constituent systems and mediators as well as in coalitions
6.2b: Guarantees of correctness by SosADL Studio

With SosADL supported by SosADL Studio, the SoS architect can guarantee that the SoS architecture description is able to raise the targeted emergent behavior

- Validation of emergent behavior supported by simulation mechanisms

- Distributed simulation of SoS architecture based on Discrete Event System Specification (DEVS)
  - MS4Me simulation engine

- The simulation engine is interfaced with PlasmaLab, a Statistical Model Checker, which enables the verification of correctness properties of the architected emergent behaviors
  - Including analysis of extreme cases
6.3 SosADL Studio: The Toolset for Formal Architectural Development of SoS using SosADL

- SosADL Studio integrates tools for describing and analyzing SoS architectures with SosADL, supporting guarantees of correctness
  - Plugins eclipse

![Diagram showing the toolset for SosADL Studio]

- Architecture Synthesizer (Alloy/Kodkod)
- Architecture Verifier (Model Checker ioSTS/UPPAAL)
- Architecture Validator (Simulator/DEVS MS4Me)
- Architecture Statistical Model Checker (PlasmaLab)
- Architecture Description Editor (Xtext & Sirius)

---

SosADL Studio integrates tools for describing and analyzing SoS architectures with SosADL, supporting guarantees of correctness.

Plugins eclipse

- Architecture Synthesizer (Alloy/Kodkod)
- Architecture Verifier (Model Checker ioSTS/UPPAAL)
- Architecture Validator (Simulator/DEVS MS4Me)
- Architecture Statistical Model Checker (PlasmaLab)
- Architecture Description Editor (Xtext & Sirius)
III. Enhancing the proposed solution for raising Collective Intelligence (CI) under Uncertainty in Software-intensive Systems-of-systems (SoS)
7. Enhancing SosADL for handling Uncertainty in Software-intensive Systems-of-Systems (SoS)
Software-intensive Systems-of-Systems (SoSs) operate in inherently uncertain operational environments.

Epistemic Uncertainty

- Epistemic uncertainty is due to partial knowledge.
- In a SoS, each constituent system has only partial knowledge of its local environment, perceived via its sensors & actuated upon via its actuators.

SoSs need to be designed and operated in the presence of epistemic uncertainty (indeed, uncertainty cannot be avoided as it is intrinsic to SoS).

- Measures from sensors are subject to uncertainty.
- Effects from actuators are subject to uncertainty.
The proposed approach relies on Fuzzy Theory for handling uncertainty when architecting SoS under epistemic uncertainty.

- Fuzzy sets are used to represent epistemic uncertainty, by representing partial or incomplete knowledge.
- It can represent partial information on the operational environment.
- It provides several calculi for the fusion of partial information from different sources, e.g., different sensors of a self-driving vehicle, as well as for computing relevant steering commands to be executed by vehicle actuators.

Are these two cars close enough or too far?
7.3 The Fuzzy Theory for supporting Epistemic Uncertainty

- Fuzzy Theory is composed of a collection of different related theories:
  - The seminal one is Fuzzy Sets, the basis for Fuzzy Logics, which underlines different approaches for Fuzzy Control Systems, supported by Fuzzy Inference Systems
  - Fuzzy Theory encompasses different types of Fuzzy Sets and Systems

- Fuzzy Theory has been developed since the sixties:
  - In 1965, Lotfi A. Zadeh published the Theory of Fuzzy Sets
  - The Theory of Fuzzy Sets has given rise to over 50,000 patents in Japan and the United States
    - Most of these applications apply Fuzzy Theory for addressing Uncertainty
    - Highly applied for Fuzzy Control Systems (control systems based on Fuzzy Logic)
Fuzzy SosADL extends SosADL with Fuzzy Constructs:

- Fuzzy action can be a fuzzify action, a rulefy action, or a defuzzify action
- Fuzzy rule expressing whenever-do-else

```plaintext
fuzzyAction := fuzzifyAction | rulefyAction | defuzzifyAction
fuzzifyAction ::= fuzzify name1 { term0 .., termn }
rulefyAction ::= rulefy name1(name0 : type0 ..,
name0 : type0 .., name0 : type0)
aggregating by operationMethod0 ...
activating by activationMethod0
accumulating by accumulationMethod0 { fuzzyRule1 .. fuzzyRule n }
defuzzifyAction ::= defuzzify name1 [inf..sup1]
by defuzzifierMethod1 { term0 .., termn } default real0
operationMethod ::= andMethod | orMethod

// t-norm for logical operator and
andMethod ::= #and 'minimum' | 'product' | 'bounded difference'
| 'drastic product' | 'einstein product' | 'hamacher product'
| 'nilpotent minimum' | ...

// t-conorm for logical operator or
orMethod ::= #or 'maximum' | 'probabilistic sum' | 'bounded sum'
| 'drastic sum' | 'einstein sum' | 'hamacher sum'
| 'nilpotent maximum' | ...
activationMethod ::= andMethod
accumulationMethod ::= orMethod
defuzzifierMethod ::= 'mean of maxima' | 'left most maximum'
| 'right most maximum' | 'centre of gravity' | 'centre of area'
| 'centre of gravity on singletons' | ...

fuzzyRule ::= whenever fuzzyAntecedent1
do { fuzzyConsequent1 } else { fuzzyConsequent2 } weighted real0
fuzzyAntecedent ::= and-or-fuzzy-clause-expression
fuzzyConsequent ::= and-or-fuzzy-clause-expression
fuzzyClause ::= name, is hedge term,
hedge ::= above | any | below | extremely | intensify | more-or-less
| norm | not | plus | seldom | slightly | somewhat | very | ...
```
8.1a Architecting Platoons under Uncertainty: Self-Driving Vehicles as Platoonmates

- Self-driving vehicles are equipped with **sensors**, e.g. radars/lidars (light detection and ranging devices) as well as steering **actuators**
- Using these devices, a self-driving vehicle can sense information from its operational environment, including other vehicles in its perception range, as well as command its steering actuators
Vehicle Platooning: platooning is the process of vehicles (in this case self-driving vehicles) autonomously forming road convoys

- Each vehicle in the platoon follows the platoonmate in front of it, except the leader that drives to the destination
- It requires that each vehicle in the platoon control its velocity and the relative distance to the vehicle in front of it
8.2 How to Architect Platoons of Self-Driving Vehicles as an SoS under Uncertainty?

- Recall that architects describe SoS architectures at design-time specifying how SoS constituents will architecturally enable to create and maintain emergent behaviors at run-time.
- Challenge to describe Platooning as an SoS architecture:
  - The challenge in the architectural design of a Platooning SoS is to conceive an SoS architecture that is able to create, on the fly, and maintain emergent behaviors from self-driving vehicles, including creating platoons as well as having platoonmates joining or leaving the platoon, where the actual vehicles and the operational environment are not known at design-time.
8.3 Fuzzy Architecture Description for Platooning SoS: Micro-scale Behaviors for Creating the Emergent Behavior of Platooning

- The described solution for Platooning SoS with SosADL: Systemic Emergence with Supervenience based on Self-Organization
  - The emergent behavior of platooning results from three micro-scale behaviors of platoonmates which together enforce the constraints that are required for enabling self-organization
  - Cohesion micro-scale behavior: every platoonmate must steer to follow the platoonmate just in front of it (if any)
  - Separation micro-scale behavior: every platoonmate must steer to avoid the near platoonmate in front of it, thereby avoiding collision
  - Alignment micro-scale behavior: every platoonmate must steer to move towards the platoonmate in front of it, or towards the destination if no near platoonmate is in front of it while attempting to match velocity (heading and speed) with nearby platoonmates
The Platooning SoS architecture is described in terms of:

- **Self-driving vehicles**: they are the SoS constituents identified at run-time: they are declared at design-time by their system capabilities in SosADL.

- **Mediators for platooning**: they are the SoS mediators created at run-time: they are declared at design-time by their mediation capabilities.

  - Mediators are created at run-time (concretized by the SoS) to achieve a goal (in this case forming and maintaining platoons), part of an encompassing mission.

  - From the viewpoint of the self-driving vehicle, each mediator generates an independent request for a steering maneuver to be executed by the mediated self-driving vehicle as a micro-scale behavior.

- The architectural role of mediators is to mediate the interaction of constituent systems for creating the requested emergent behavior, in this case, platooning.
8.4b Fuzzy Architecture Description for Platooning
SoS: Inside of Fuzzy Mediators

- With the data received from the sensors, the Fuzzy Mediator applies the three fuzzy rule sets (cohesion, separation, alignment) to determine how the mediated self-driving vehicle must behave.
- Each resulting fuzzy value determines through a vector of speed and heading how the self-driving vehicle will be commanded to achieve cohesion, separation, and alignment.
- Achieving collectively platooning.
The Fuzzy SosADL Studio supports the application of Fuzzy SosADL

- Plugins eclipse
IV. Summing Up on architecting Software-intensive Systems-of-systems (SoS) for raising Collective Intelligence (CI)
9. Summing Up and Take Away Message
9.1 Summing Up

- Single Systems and Systems-of-Systems (SoS) are of different nature
  - SoS raises numerous challenges due to their defining characteristics, including independence of the constituent systems, evolutionary development, and intrinsic complexity implied by emergence

- SoS architecture calls for novel approaches
  - For mastering SoS complexity
    - Formalizing Emergent Behavior
  - For raising SoS emergent behaviors
    - Formalizing Self-Organization
  - For coping with uncertainty in operational environments
    - Formalizing Uncertainty
SosADL comprises a new paradigm for architecture description of Software-intensive Systems-of-Systems.

- Enables to describe SoS software architectures abstractly at design-time without knowing which will be the actual concrete systems in the SoS at run-time.

SosADL supports the description of emergence:

- Supervenience enables to describe systemic emergence.
- Self-organization enables to create spontaneous systemic emergence.

SosADL captures the formal semantics of the behavior of SoS architecture descriptions, based on a formal foundation: the $\pi$-Calculus for SoS.

Fuzzy SosADL handles epistemic uncertainty in the description of SoS architectures.
Global Earth Observation SoS (GEOSS)

- GEOSS is a set of coordinated, independent Earth observation, information and processing systems that interact and provide access to diverse information for a broad range of stakeholders
  - It targets missions for biodiversity and ecosystem sustainability
9.4 Take Away Message

- The Software Architecture of **Software-intensive Systems-of-Systems** calls for novel paradigms
- Single System and System-of-Systems are of different nature
  - Systems-of-Systems bring complexity to architecture
    - Emergent behavior
  - Systems-of-Systems bring independence of constituents to architecture
  - Systems-of-Systems call for new architectural approaches
    - For mastering Supervenience
    - For mastering Self-Organization
    - For handling Epistemic Uncertainty
- Architecting SoS has been identified as a major issue in the upcoming years, e.g. INCOSE Vision 2035
9.5 For More Information

- Contact me at flavio.oquendo@irisa.fr
- Consult the different publications: search for Flavio Oquendo at DBLP

Main publication venues on SosADL and the worked presented:
- Systems Engineering Journal
- Software and Systems Modeling (SoSyM) Journal
- The Computer Journal
- Science of Computer Programming (SCP) Journal
- IEEE International Conference on System-of-Systems Engineering (SoSE)
- European Conference on Software Architecture (ECSA)
- International Conference on Engineering of Complex Computer Systems (ICECCS)
- IFIP International Internet of Things Conference (IFIP IoT)
- International Conference on Communicating Process Architectures (CPA)
- International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)
Acknowledgment to the members of my research team who contributed to this research thread on Software-intensive System-of-Systems Architecture

- Jérémy Buisson
- Everton Cavalcante
- Lina Garcés
- Marcelo Gonçalves
- Valdemar Graciano Neto
- Milena Guessi
- Elena Leroux
- Gersan Moguérou
- Lucas de Oliveira
- Jean Quilbeuf
- Eduardo Silva
Thank You

Questions?

Flavio Oquendo
flavio.oquendo@irisa.fr
http://people.irisa.fr/Flavio.Oquendo/

ECSA 2023
Mon 18 - Fri 22 September 2023
Yeditepe University, Istanbul, Turkey.