

Systems-of-Systems

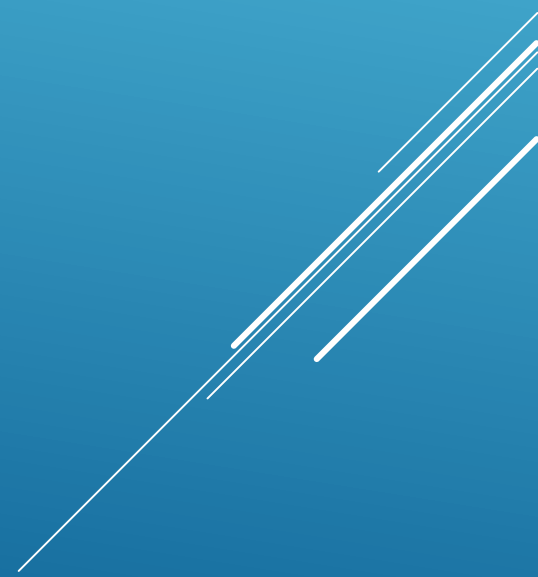
Rosana T. Vaccare Braga

Part of the slides produced by:

Milena Guessi Margarido

Prof. Dr. Elisa Yumi Nakagawa

Prof. Dr. José Carlos Maldonado



Definitions

- System of Systems: systems that are composed of independent constituent systems, which act jointly towards a common goal through the synergism between them (Nielsen, 2012)
- A system is considered a SoS when (Maier, 1998):
 - (1) Its components fulfilled valid purposes in their own right and continued to operate to fulfill those purposes if disassembled from the overall system, and
 - (2) the component systems are managed (at least in part) for their own purposes rather than the purposes of the whole

Definitions

- Systems-of-Systems are large-scale integrated systems that are heterogeneous and independently operable on their own, but are networked together for a common goal (Jamshidi, 2008)
- A set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities (DoD, 2008)

SoS Definition

- **System-of-Systems** is any system that:
 - results from the interoperation of organizational and managerial independent constituents, which have their individual mission and participate aware or not to comply with a global mission;
 - has evolutionary development resulting from evolution of constituents and/or changes in the environment;
 - presents emergent behaviors, expected or non-expected in design time, resulting from the interaction among constituents at runtime; and
 - depends on software as an enabling technology to its design and evolutionary development.

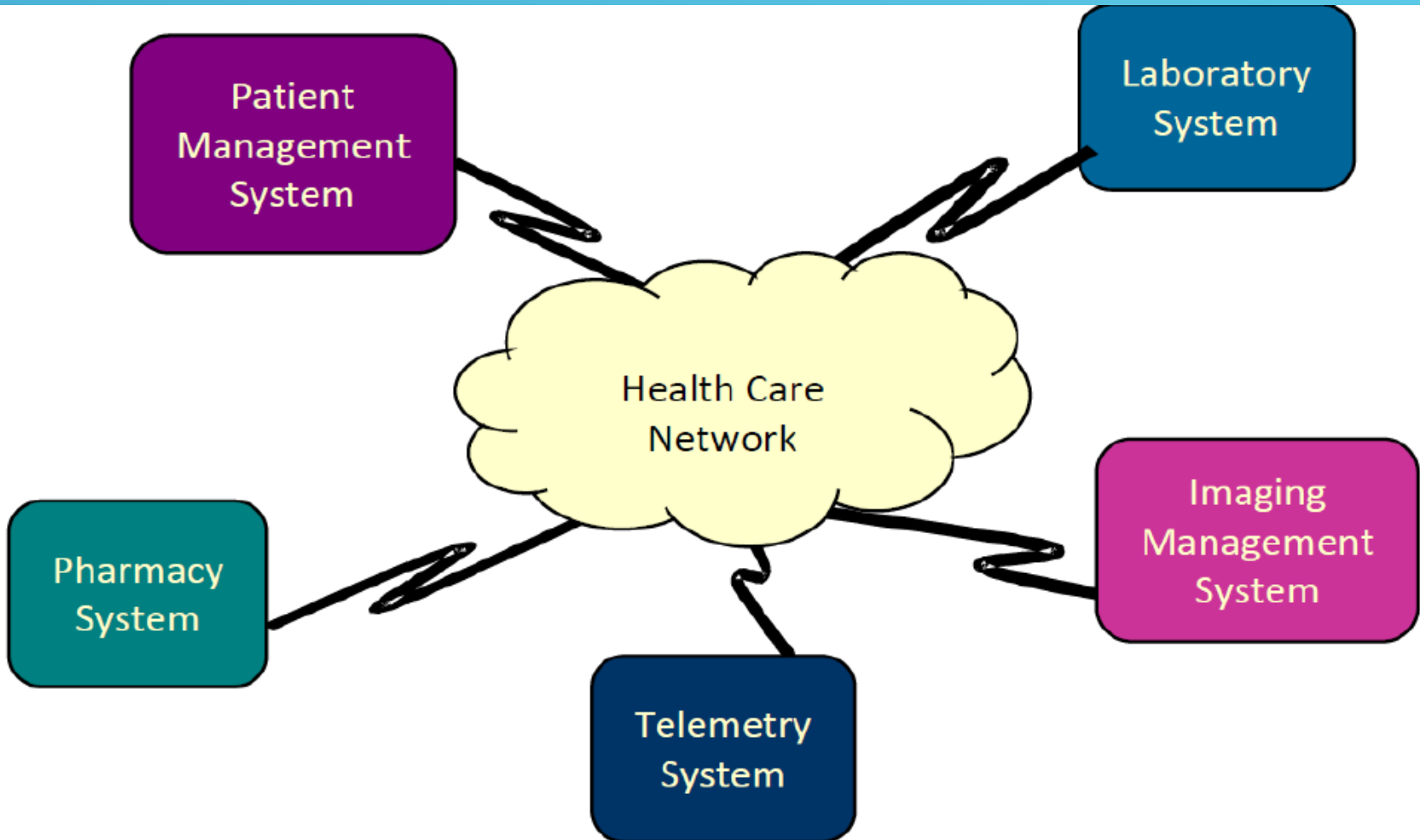
[Nakagawa, Maldonado, Oquendo 2016]

SoSs

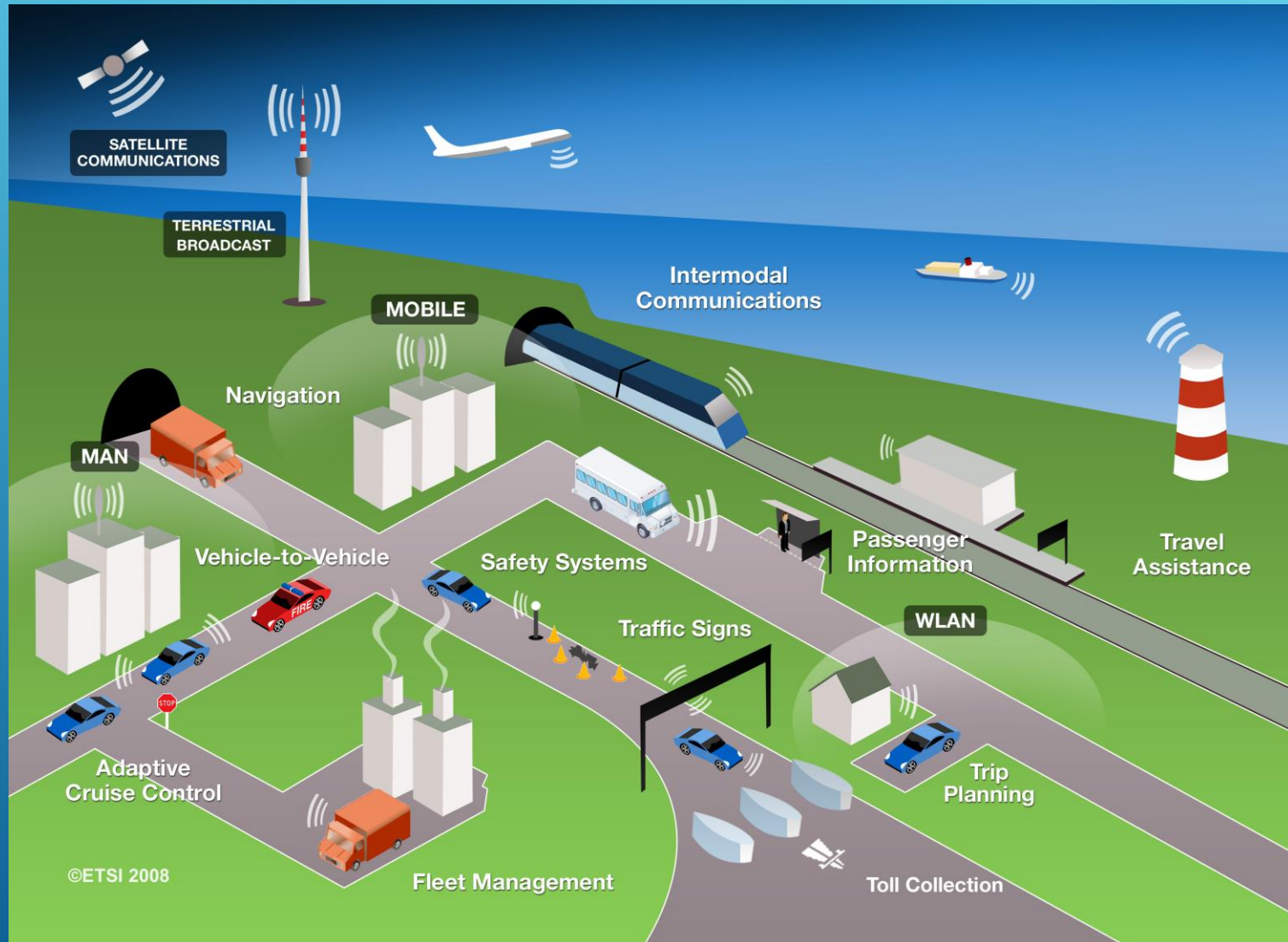
- Independent constituent systems
 - Action and decision making
- Geographic distribution
- Evolutionary development
- Emergent behavior



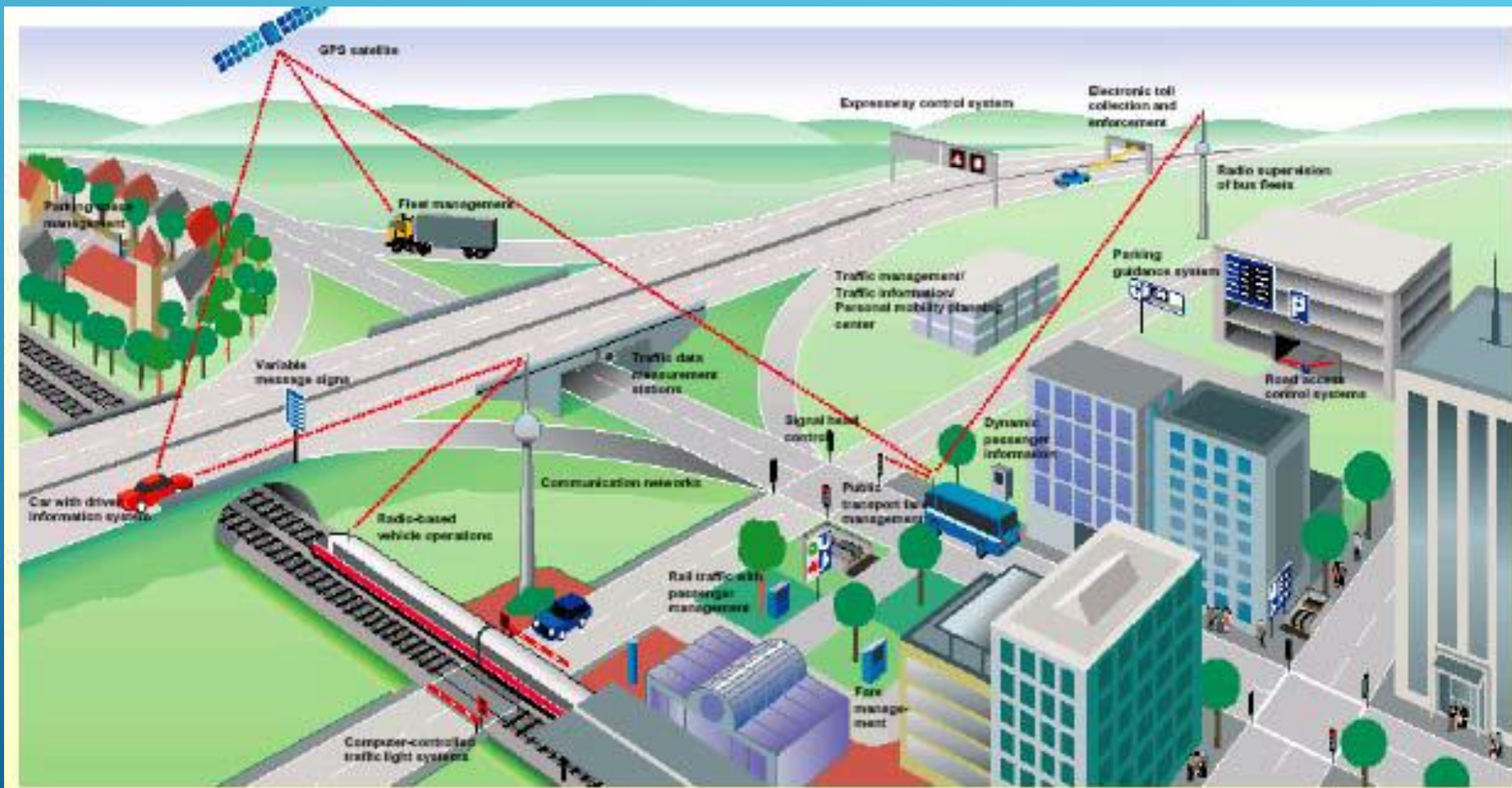
Examples of SoS



Examples of SoS



Examples of SoS



Examples of SoS



SoSs

- Open systems
 - Top
 - Continually open for addition of new applications and systems, without any top-level system defining the SoS
 - *Emergent behavior*
 - Bottom
 - The lowest level of the SoS (e.g., communication stack) may be changed at any time
 - *Interoperability*
 - Continually evolving
 - **An SoS is never complete** as it evolves **at run-time** according to changes in the surrounding environment

SoS main characteristics

System-of-Systems

Emergent behavior

Evolutionary development

Distribution

Software-intensity

Constituents

Operational Independence

Managerial Independence

SoS Characteristics

- Characteristics related to the nature of constituents:
 - *Operational independence*
 - Constituents operates independently, having its own mission and resources
 - *Managerial independence*
 - Constituents present independent management and evolve in ways not foreseen when they originally joined to particular SoS

SoS Characteristics

- Characteristics related to the nature of constituents:
 - *Emergent behavior*
 - New behaviors from constituents
 - Behaviors non-predictable in design time emerge only at runtime
 - *Evolutionary development*
 - Constituents continually evolve, implying evolution in SoSs
 - SoSs evolve due to changes in their environment

SoS Characteristics

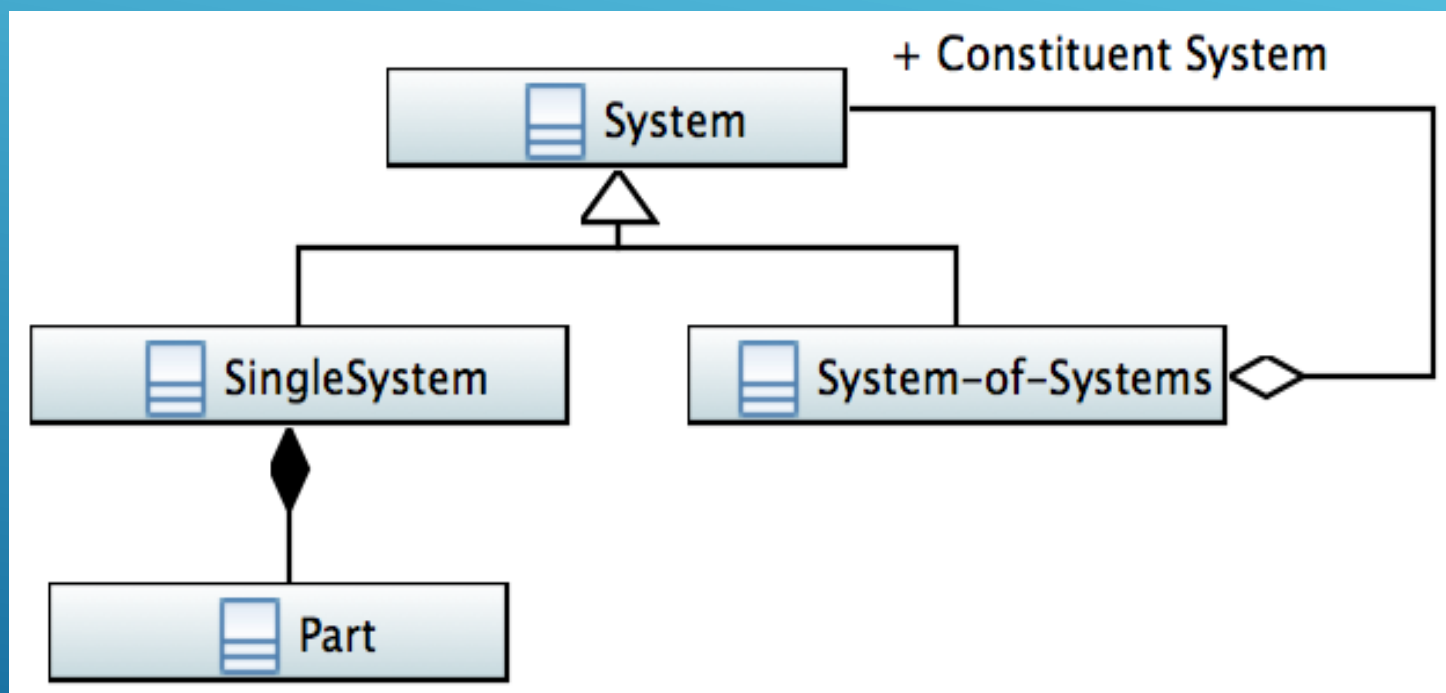
- Characteristics related to the nature of constituents:
 - *Distribution*
 - Distributed constituents, geographically or not
 - *Software-intensity*
 - Influence to the design, construction, deployment, and evolution of SoSs and constituents

Consequence of SoS characteristics: **dynamic architectures**

SoS distinguishing characteristics

Element	System	System of Systems
Autonomy	Autonomy is ceded by parts in order to grant autonomy to the system	Autonomy is exercised by constituent systems in order to fulfill the purpose of the SoS
Belonging	Parts are akin to family members; they did not choose themselves but came from parents. Belonging of parts is in their nature.	Constituent systems choose to belong on a cost/benefits basis; also in order to cause greater fulfillment of their own purposes, and because of belief in the SoS supra purpose.
Connectivity	Prescient design, along with parts, with high connectivity hidden in elements, and minimum connectivity among major subsystems.	Dynamically supplied by constituent systems with every possibility of myriad connections between constituent systems, possibly via a net-centric architecture, to enhance SoS capability.
Diversity	Managed i.e. reduced or minimized by modular hierarchy; parts' diversity encapsulated to create a known discrete module whose nature is to project simplicity into the next level of the hierarchy	Increased diversity in SoS capability achieved by released autonomy, committed belonging, and open connectivity
Emergence	Foreseen, both good and bad behavior, and designed in or tested out as appropriate	Enhanced by deliberately not being foreseen, though its crucial importance is, and by creating an emergence capability climate, that will support early detection and elimination of bad behaviors.

Relating Systems and SoS



A SoS is a system, too!!!


We can have a System of Systems of Systems!!!

Types (Tentative)

■ Directed SoS


- SoS that are centrally managed
- Constituents are developed or acquired to fit specific purpose
- Constituents operate under tight subordination

■ Acknowledged SoS

- SoS that are centrally managed
 - Constituents retain their operational independence
 - Constituents operate under loose subordination
- 
- A decorative graphic consisting of several parallel white lines of varying lengths and orientations, located in the bottom right corner of the slide.

Types (Tentative)

- Collaborative SoS
 - There is no central management
 - Constituent systems voluntarily agree to fulfill central purposes

 - Virtual SoS
 - There is no central authority or centrally agreed purpose
- 

Types (Tentative)

Central
Control

CENTRALIZED SoS

Central Mission
Tight Subordination

SEMI- CENTRALIZED SoS

Central Mission
Loose Subordination

No
Central
Control

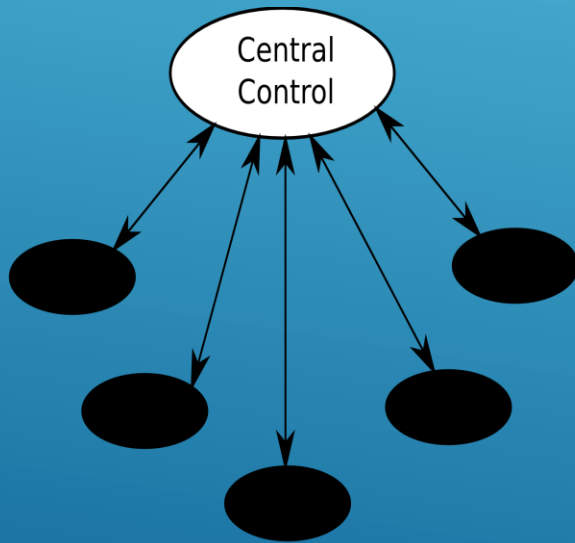
COLLABORATIVE SoS

Central Mission
No Subordination

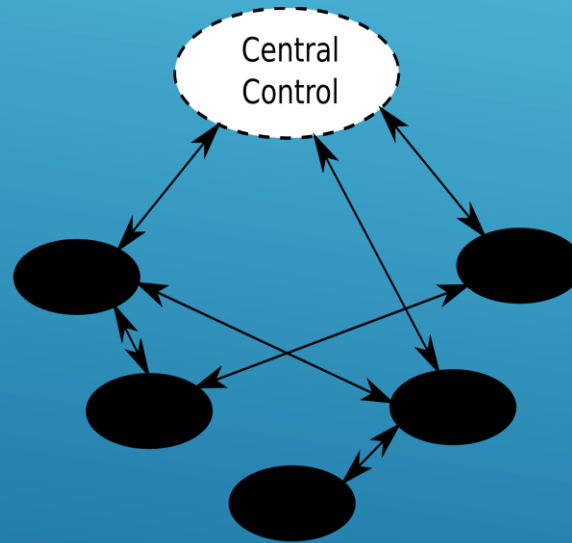
EMERGENT SoS

No Central Mission
No Subordination

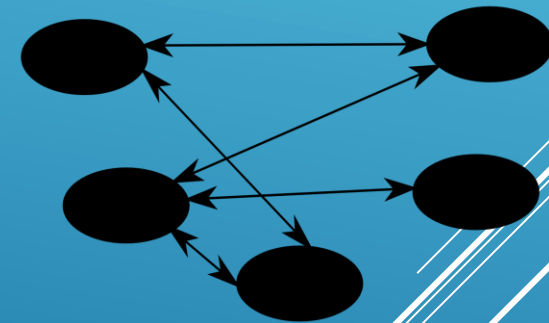
Types (Tentative)



(a) Centralized



(b) Semi-centralized



(c) Decentralized

Challenges/Questions on the SoS Development and Evolution

- Do traditional SEng processes/practices work on SoS?
 - What works? What does not work? What needs adaptation?
- How to manage SoS evolution?
 - How to manage the SoS emergent behaviors?
 - How to manage the SoS dynamic architectures?

One of the solutions: Software Architecture

SE x SoSE

- Traditional SE practices are often not sufficient to engineer a SoS .

Critical Point	System Engineering	SoS Engineering
Focus of Analysis	Single System	Integration of Systems
Focus of Improvement	Optimization	Realistic Cost and Scheduling
Target	End Product	Initial Deployment
System Requirements	Fixed	Evolving
System Boundaries	Well-defined	Indefinable

Traditional x SoS engineering (Keating et al., 2003)

SoS Software Architecture

- Software architectures
 - Backbone for software-intensive systems
 - Fundamental in determining the system quality
 - Considerable amount of research, mainly regarding their design, representation, and evaluation

**Software architectures for SoS is
a new, important research area!!**

A decorative graphic consisting of several parallel white lines of varying lengths, slanted diagonally from the bottom right towards the top right, set against a blue background.

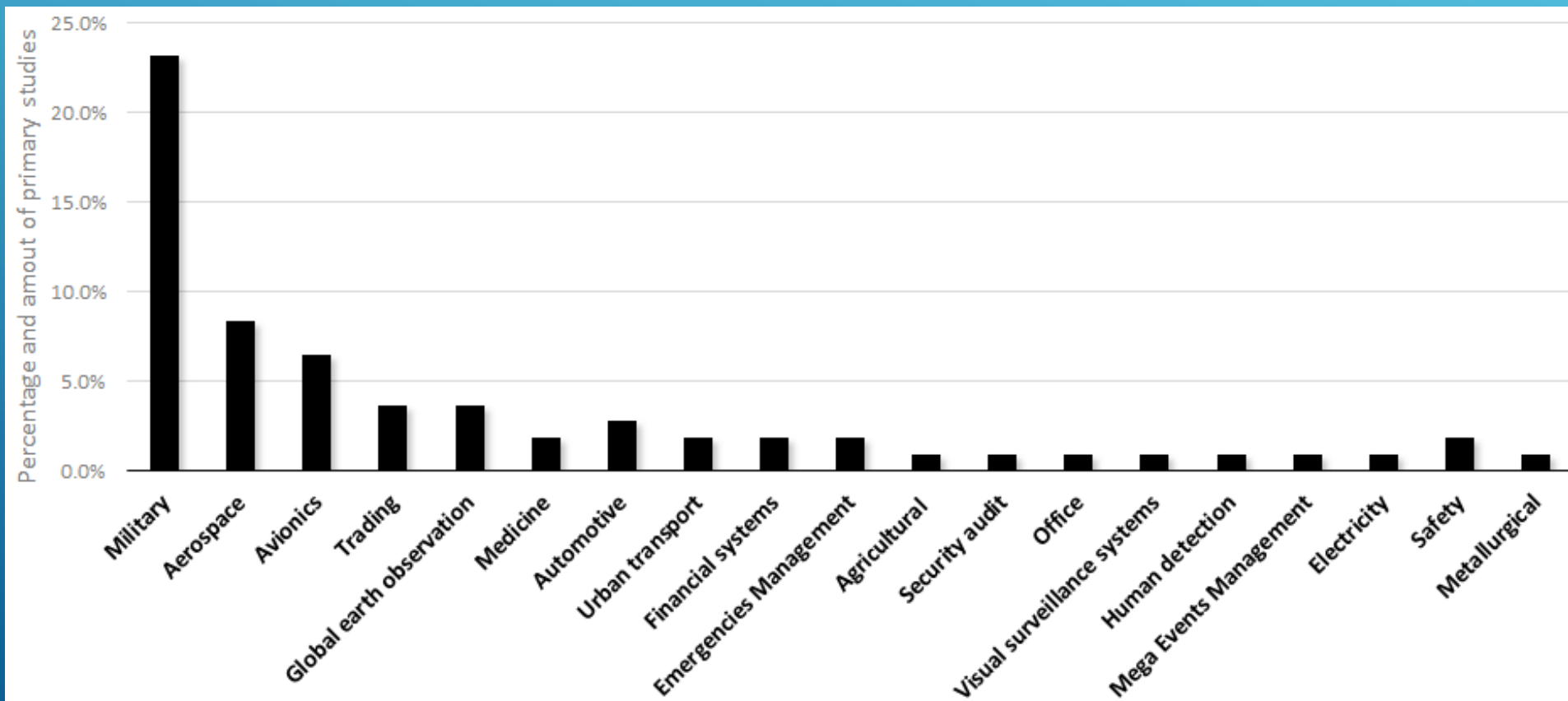
SoS Software Architecture

- “The software architecture of a SoS is a dynamic structure or structures of a system, which comprise the independent constituent systems, the externally visible properties of those constituents, the relationships among them, and the principles and constraints that guide both its initial design and its evolution imposed by the emergence of expected and non-expected missions at runtime.”

(NAKAGAWA et al., 2016)

SoS Software Architecture

■ Application domains:



Global Earth Observing System of Systems (GEOSS)

SoSs Example GEOSS

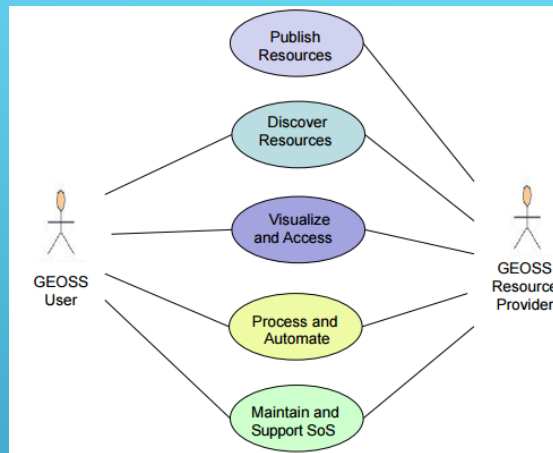
- GEOSS is to be a global, coordinated, comprehensive and sustained system of Earth observing systems
 - Promote coordinated access to data and products produced amongst all contributing systems
- Introduces consistency of content through guidelines to data providers for the appropriate characterization of the observing systems and their derived products
 - Adoption of **standardized best practices**

SoSs Example GEOSS

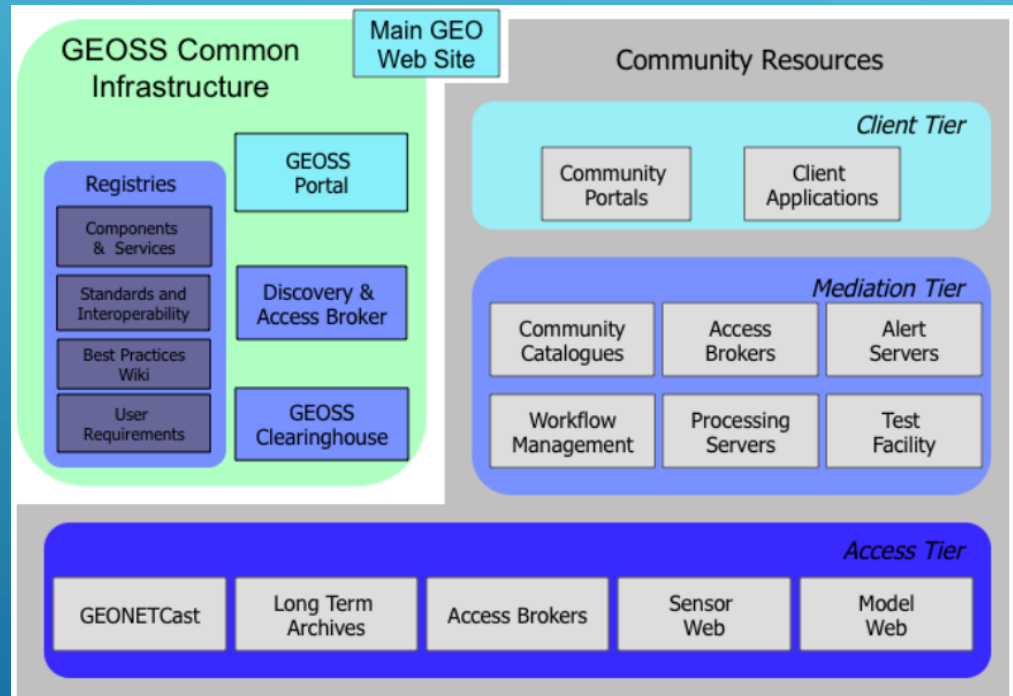
- Variety of users
- Various communities with their own cultures
- Distributed system
 - No new single architecture imposed to everyone
 - Preserve the existing infrastructures as much as possible
 - Enforce simple and robust interfaces and formats
- Dynamic, open system
 - Grow and attract third-party data and service providers and accepts intermitent participation with disconnected/connected modes without disruption
- Comprehensive information flow
 - End-to-end: product order, planning, acquisition, processing, archiving, and distribution

SoSs Example

GEOSS Architecture Implementation Pilot (AIP)



Use Cases



Engineering components with services

SoSs

Example

GEOSS

- Interoperability through open interfaces and reference methods
 - Interoperability specifications agreed to among contributing systems
 - Access to data and information through service interfaces
- Open standards and intellectual property rights
 - Preference for formal international standards
 - Multiple software implementations compliant with the open standards should exist

SoSs Example GEOSS

- Build upon existing systems and historical data
 - National, regional or international agencies that subscribe to GEOSS but retain their ownership and operational responsibility
- Implementation plan must address cost effectiveness, technical feasibility, and institutional feasibility
- To be sustained over a long period of time, GEOSS needs to be adjustable, flexible, adaptable, and responsive to changing needs
 - Capture future capabilities through open architecture



SOA is configurable and scalable to customer needs and leverages robust systems and processes for global interoperability

SoSs Description

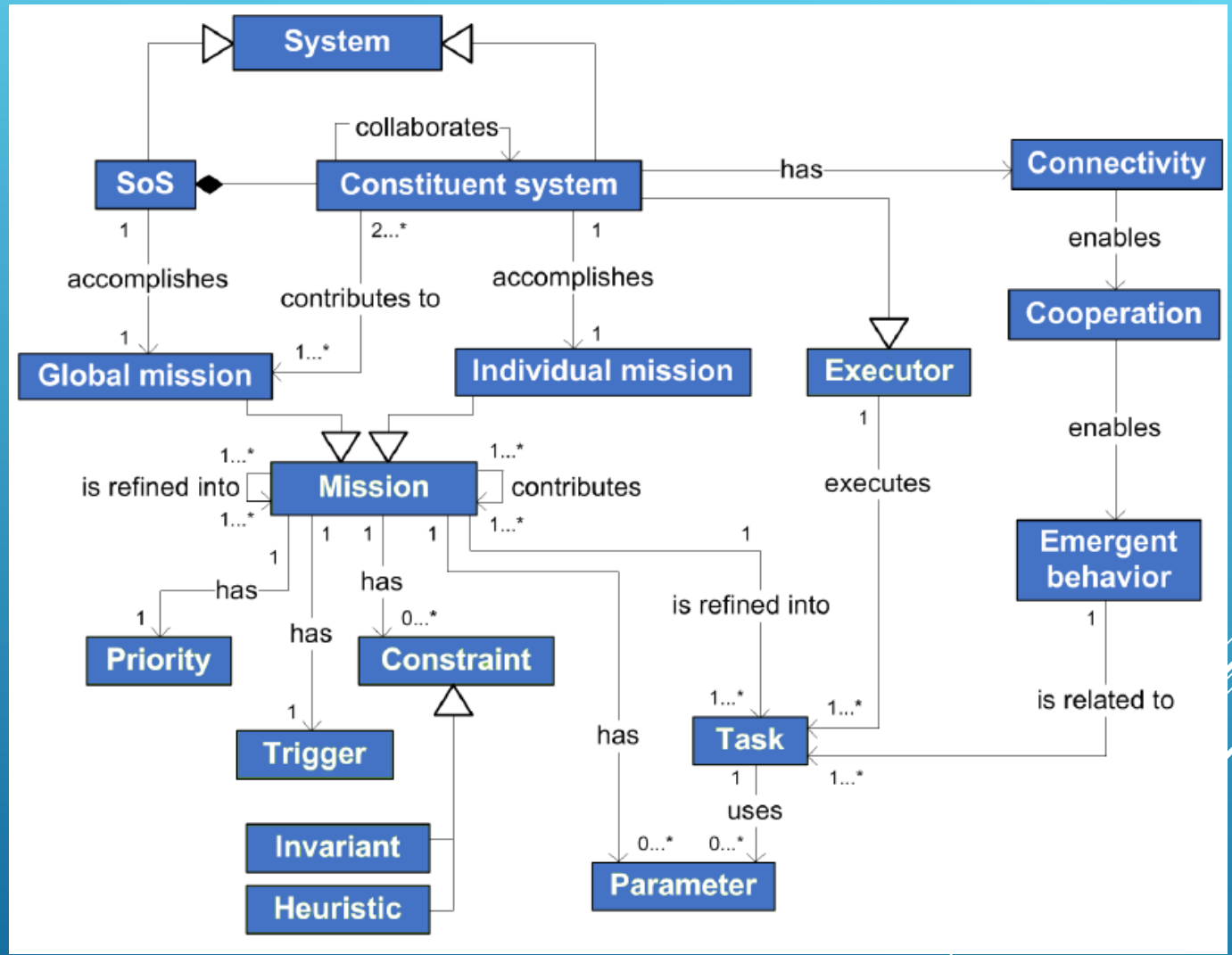
SoSs Description

- Two levels
 - Mission
 - Identifies required capabilities for constituents, operations, connections, emergent behavior, etc.
 - Architecture
 - Describes structure, behavior, and properties about the SoS

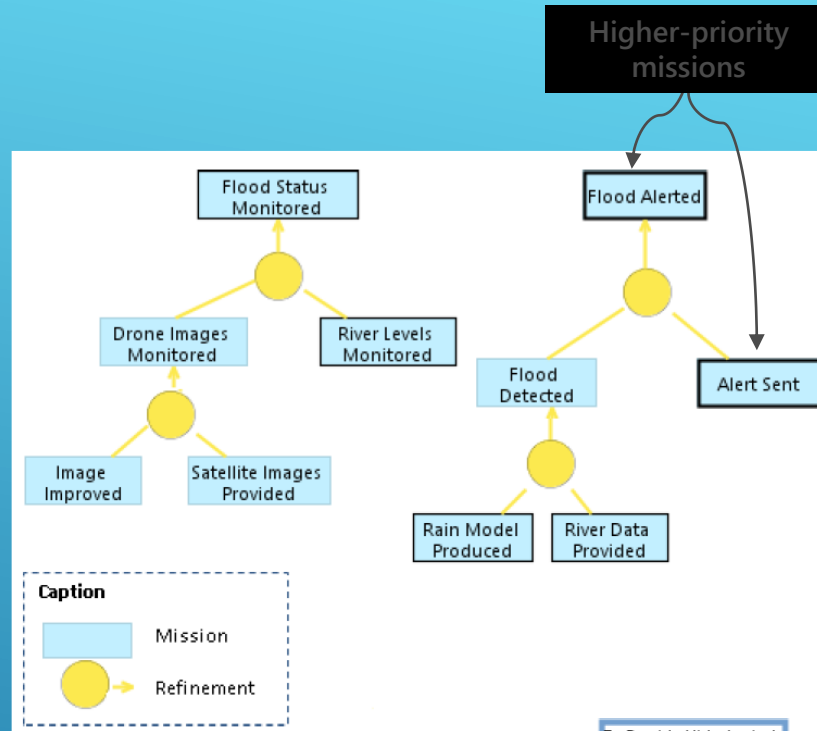
Mission

- Definition
 - Higher functionality that cannot be performed by any constituent alone
 - Accomplished by *emergent behaviors*
 - Guides the whole SoS development process
- mKAOS
 - Language for describing mission models
 - Tool: mKAOS Studio

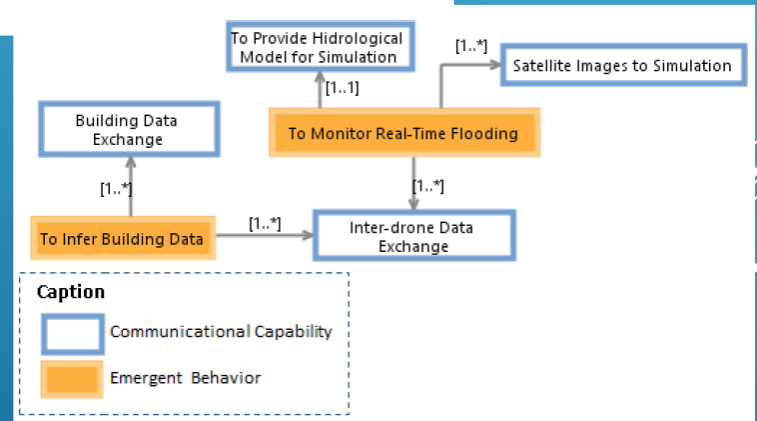
Mission Conceptual model



Mission



Mission model in mKAOS



Emergent behavior model in mKAOS

SoSs Architectural Description

"To gain confidence that an SoS architecture will respect key properties, it is paramount to have a precise model of the constituents and the connectors between them, the properties of the constituents, and the SoSs environment."

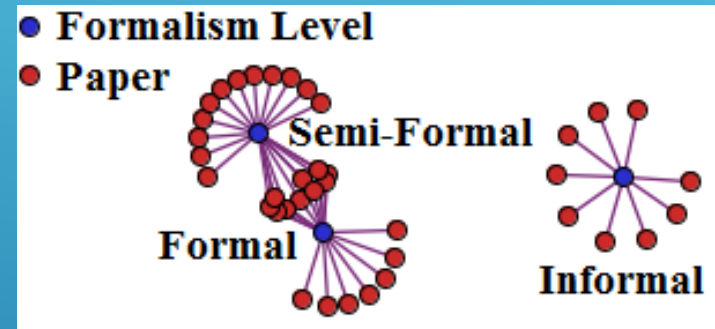
Nielsen et al. (2015)

SoSs Architectural Description

- How has the literature addressed the architecture description of SoS?
- Which are the techniques used in the description of software architectures of SoS?
- Does the primary study focuses on a specific type of SoS?

Techniques Used for Describing SoSs Architecture

- Formal languages:
 - CML, CFML, FSM, OWL, VDM-SL, among others
- Semi-formal languages:
 - UML, SysML, and UPDM
- Combination of formal and semi-formal languages:
 - UML/SysML + Petri nets
 - SysML + VDM-SL



- ▶ [ADLs] provide mechanisms for expressing composition, abstraction, reusability, configuration, and analysis of software architectures (Shaw and Garlan, 1994)
- ▶ An ADL must explicitly model components, connectors, and their configurations; furthermore, to be truly usable and useful, it must provide tool support for architecture-based development and evolution (Medvidovic and Taylor, 2001)

ADLS TRADITIONAL DEFINITIONS

- ▶ Architecture building blocks
 - ▶ Components
 - ▶ Connectors
 - ▶ Configurations
- ▶ Tool Support
 - ▶ Enable automated analyses on the architecture description

ADLS
CHARACTERISTICS

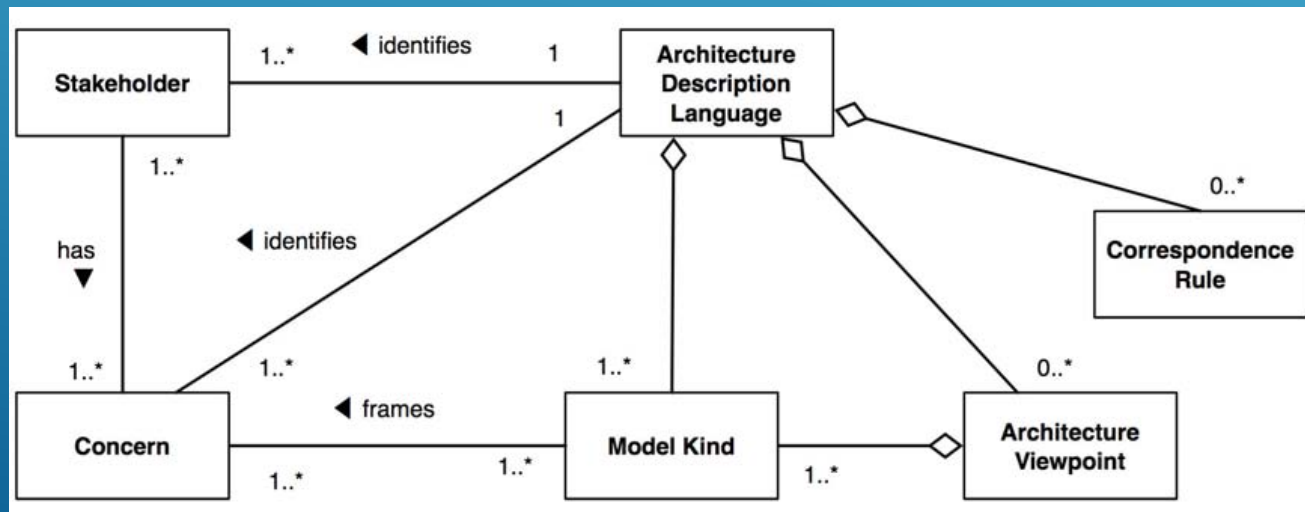
ADLS CHARACTERISTICS

- ▶ Components and Connectors
 - ▶ Interface
 - ▶ Type
 - ▶ Semantics
 - ▶ Constraints
 - ▶ Evolution
 - ▶ Non-functional properties
- ▶ Tool Support
 - ▶ Active specification
 - ▶ Multiple views
 - ▶ Analysis
 - ▶ Refinement
 - ▶ Implementation generation
 - ▶ Dynamism
- ▶ (Architectural) Configuration
 - ▶ Understandability
 - ▶ Compositionality
 - ▶ Refinement and traceability
 - ▶ Heterogeneity
 - ▶ Scalability
 - ▶ Evolution
 - ▶ Dynamism
 - ▶ Constraints
 - ▶ Non-functional properties

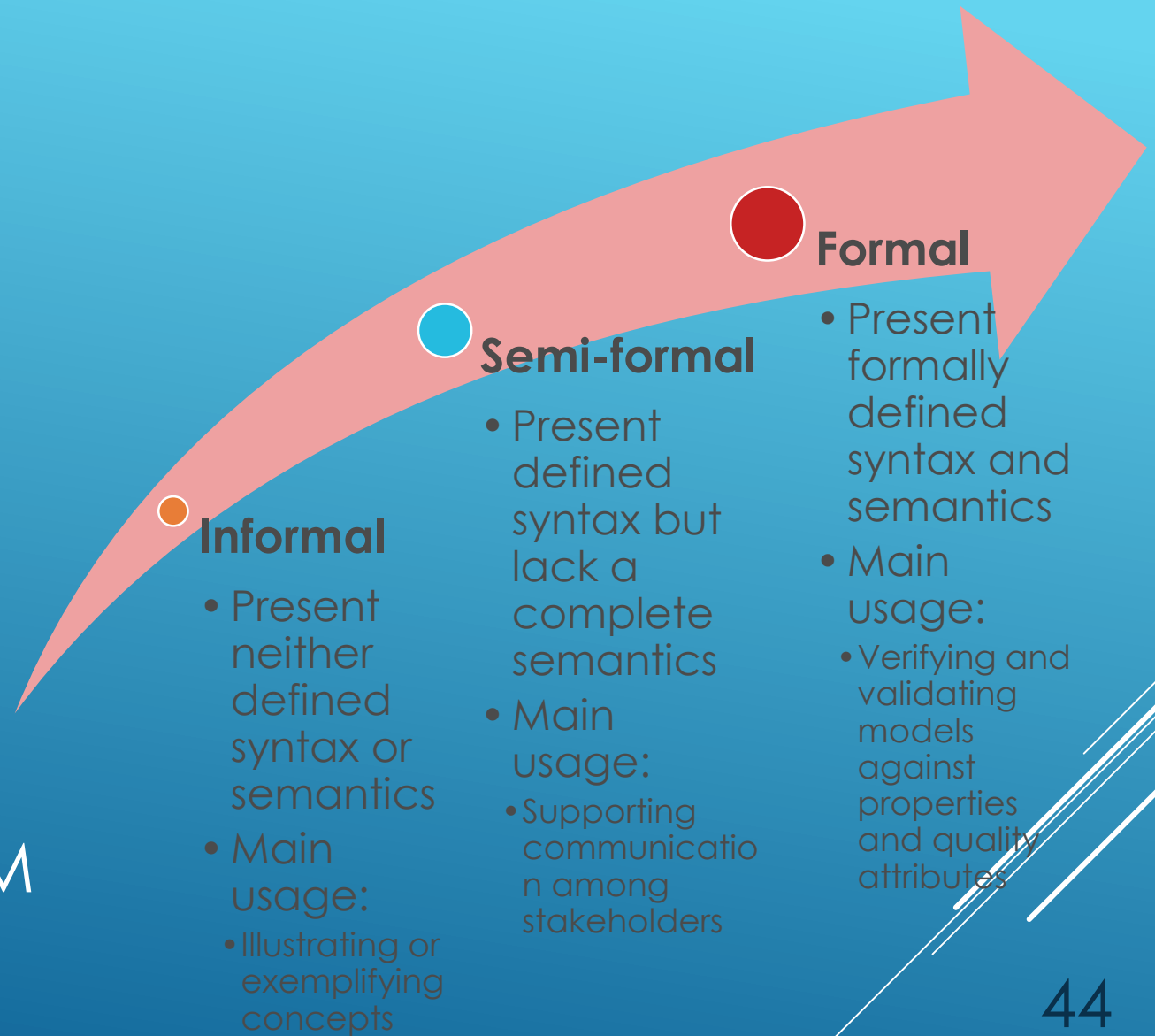
ADL CONCEPTUAL MODEL

ISO/IEC/IEEE 42010

- ▶ An ADL is any form of expression for use in architecture descriptions

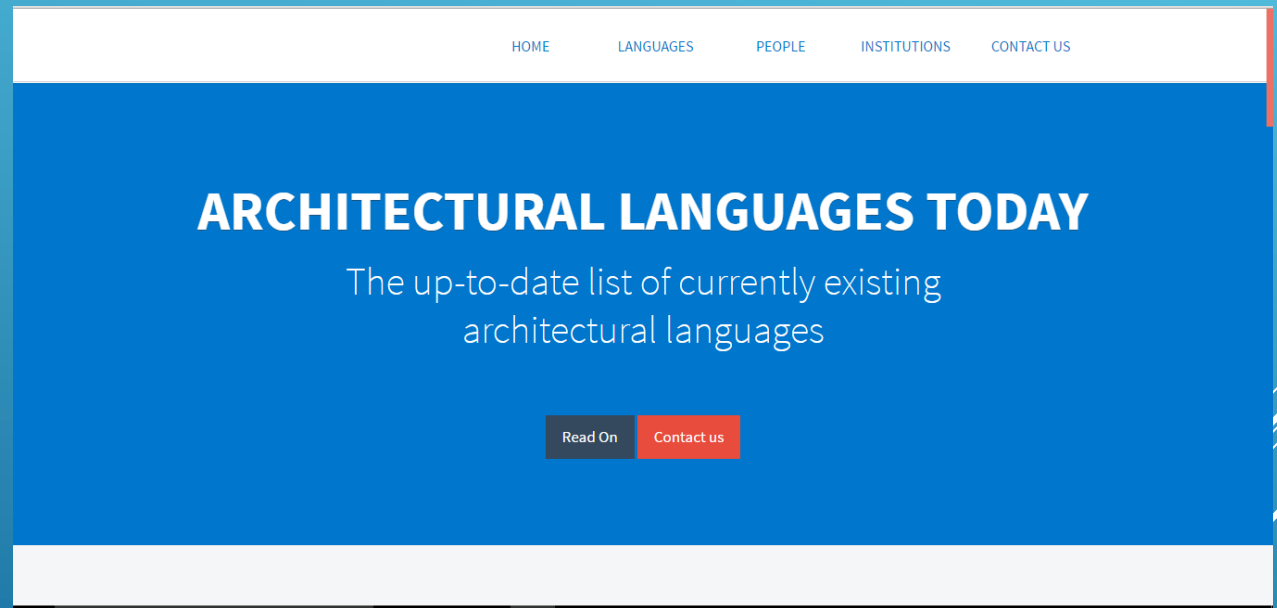


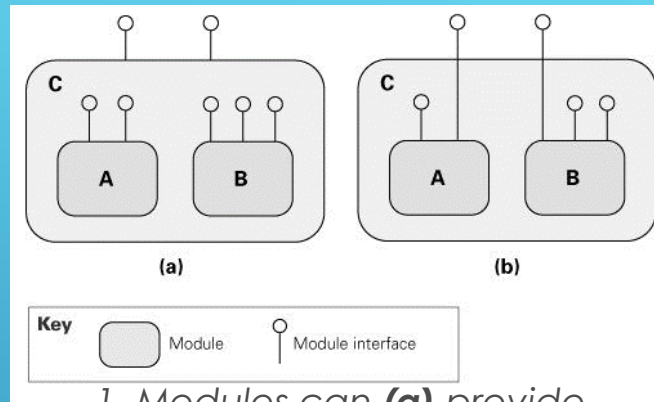
ADL FORMALISM LEVEL



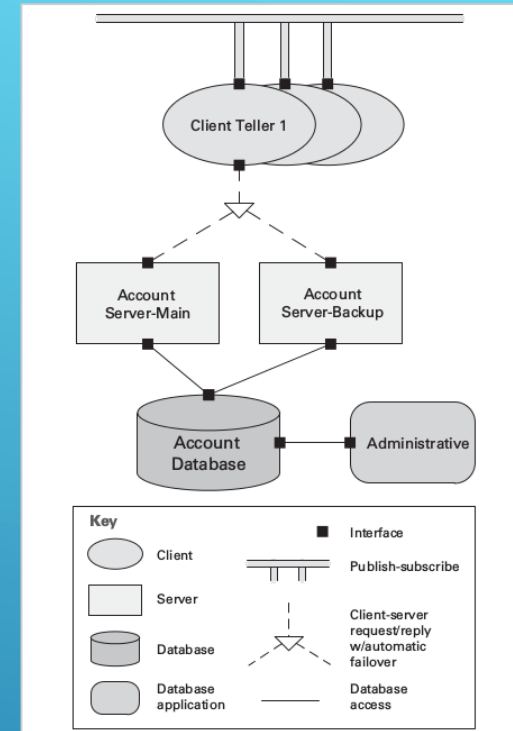
- ▶ Many, many, many ADLs...
 - ▶ 123!!

ADL EXAMPLE

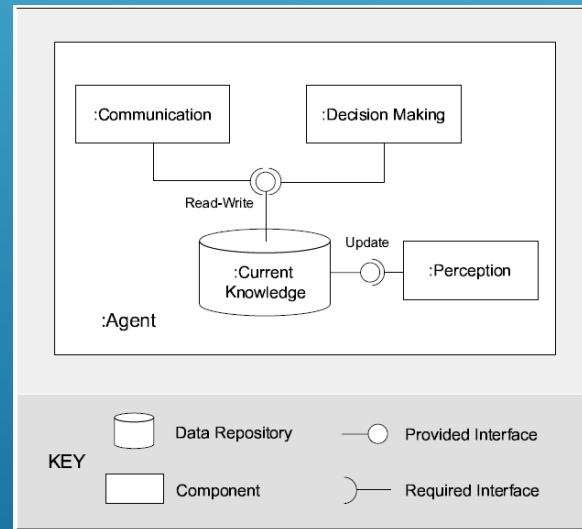




1. Modules can **(a)** provide interfaces, hiding other modules, or **(b)** exposing some interfaces of internal modules



2. A bird's-eye-view of a system as it appears at run-time.



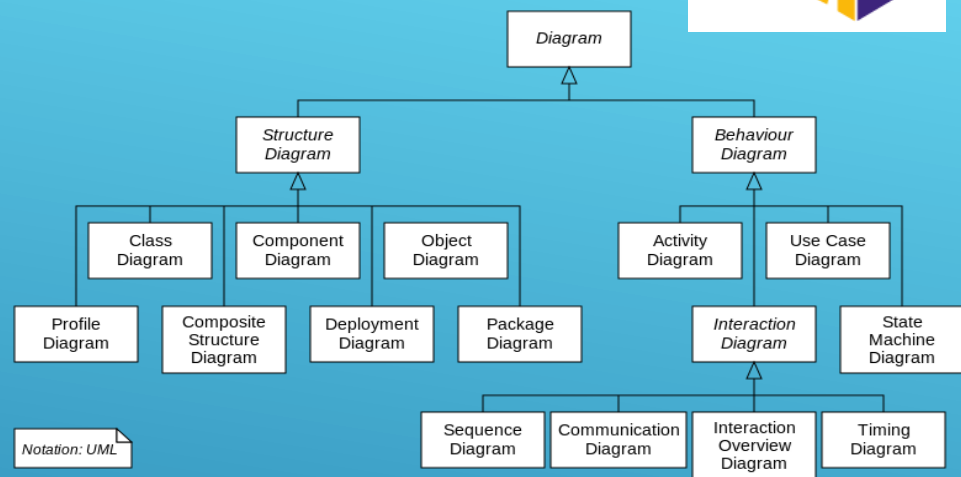
3. Shared data view of an agent

INFORMAL ADL EXAMPLE

Source:

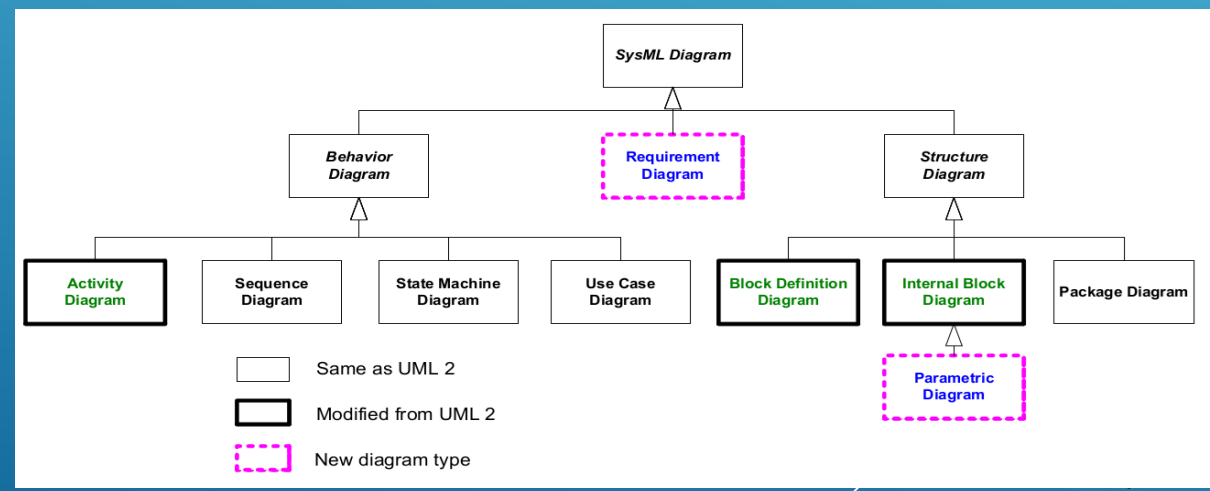
1,2 Clements, P. et al., 2011

3 Weyns, D. An Architecture-Centric Approach for Software Engineering with Situated Multiagent Systems. PhD Thesis. 2006. Available at: http://www.cs.kuleuven.be/publicaties/doctoraten/cw/CW2006_09.abs.html



1. UML 2.x diagram types

SEMI-FORMAL ADL EXAMPLE



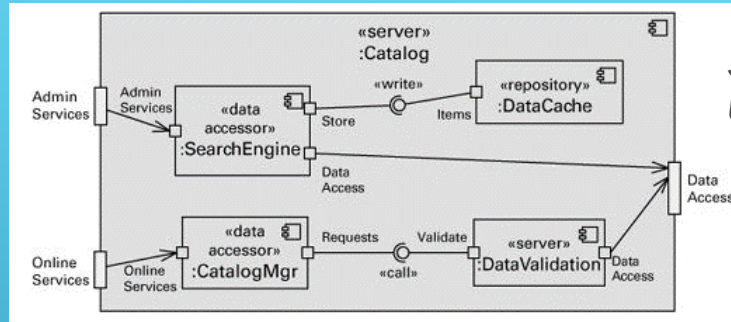
2. SysML 1.x diagram types

Source:

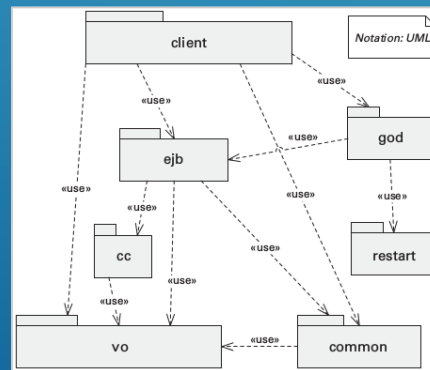
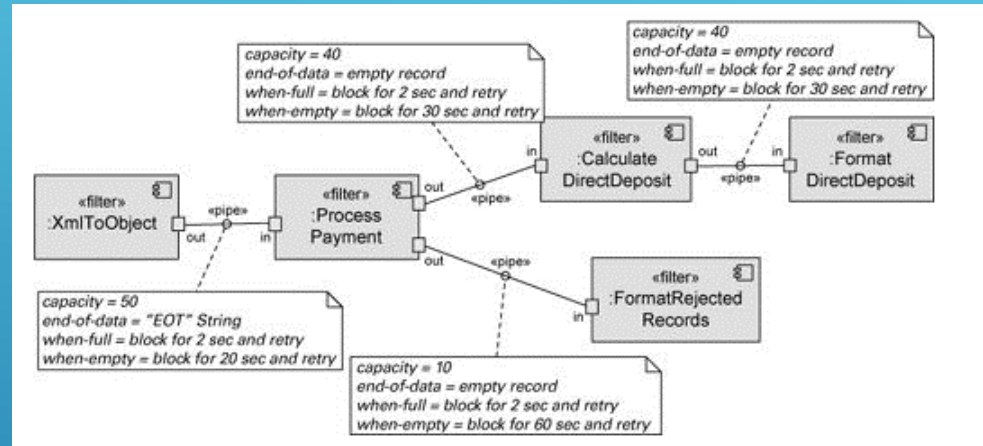
- 1 <http://www.omg.org/spec/UML/2.5/>
- 2 <http://www.omg.org/spec/SysML/1.4/>

SEMI-FORMAL ADL EXAMPLE

UML diagram of a pipe-and-filter view



Substructure of a UML component



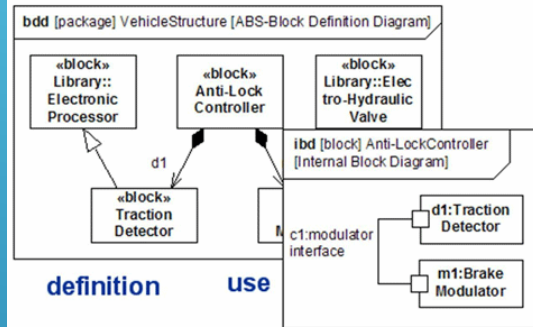
using module \ used module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	1	0	0	1	0	0	0
cc	0	1	0	0	0	0	0
god	1	0	0	0	0	0	0
restart	0	0	0	1	0	0	0
common	1	1	0	0	0	0	0
vo	1	1	1	0	0	1	0

Key: "1" means module in column uses module in row

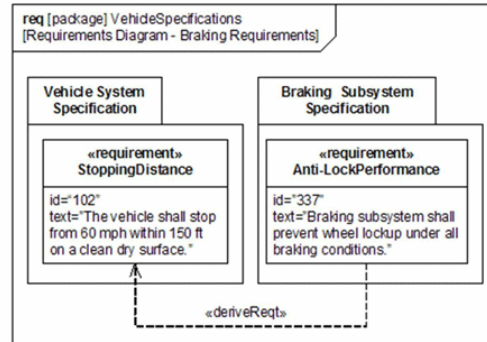
UML package diagram (left) and Dependency Structure Matrix (DSM) (right)

SEMI-FORMAL ADL EXAMPLE: SYSML

1. Structure

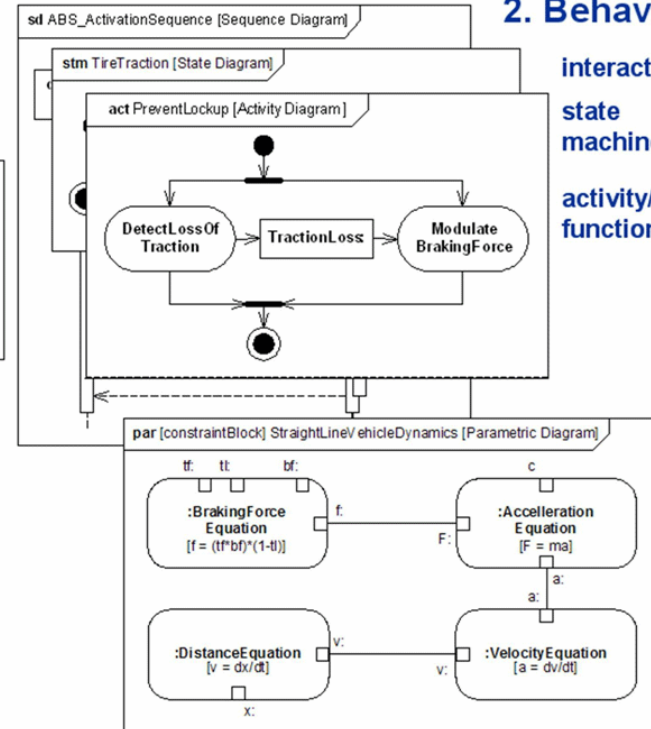


definition use



3. Requirements

2. Behavior

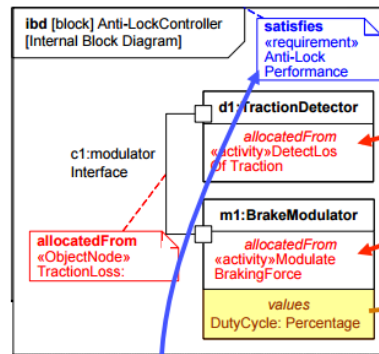


interaction
state machine
activity/function

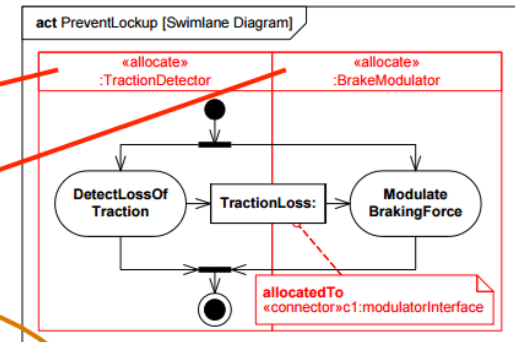
4. Parametrics

SEMI-FORMAL ADL EXAMPLE: SYSML

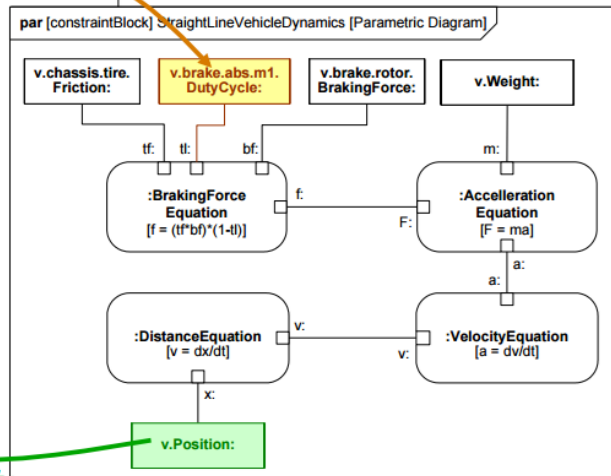
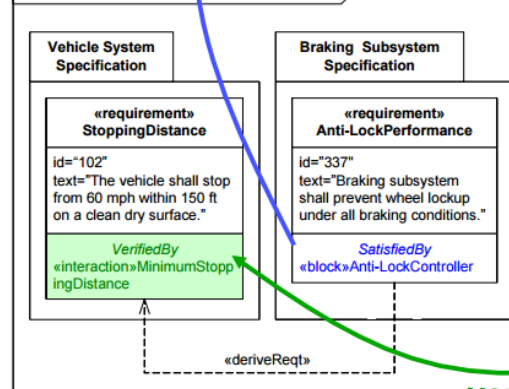
1. Structure



2. Behavior



3. Requirements



4. Parametrics

satisfy

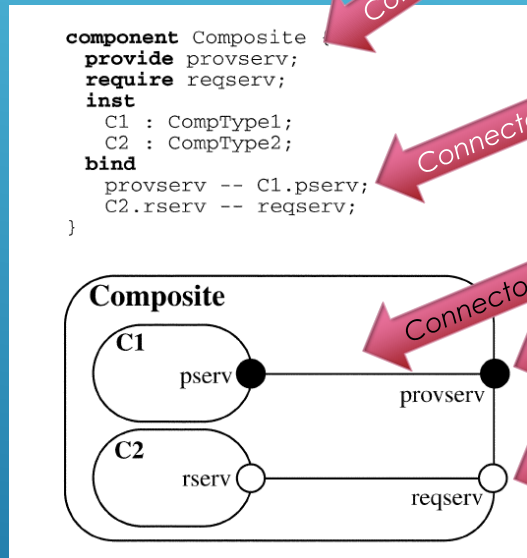
allocate

value binding

verify

Formal ADL Example

EXAMPLES (FORMAL)



A composite component specified in Darwin (top) and (bottom) the graphical view of the component

```

Sample_Arch.addComponent(Comp5);
Sample_Arch.weld(Conn1, Comp5);
Sample_Arch.weld(Comp5, Conn2);
Comp5.start();
    
```

Dynamic insertion of a component into a C2SADEL architecture.

```

Style Pipe-Filter
...
Constraints
   $\forall c : \text{Connectors} \bullet \text{Type}(c) = \text{Pipe}$ 
   $\wedge \forall c : \text{Components}; p : \text{Port} \mid p \in \text{Ports}(c) \bullet$ 
     $\text{Type}(p) = \text{DataInput} \vee \text{Type}(p) = \text{DataOutput}$ 
    
```

The pipes-and-filters style declared in Wright.

```

Family fam = {
  Component Type comp1 = { Port p1; }
  Component Type comp2 = { Port p2; }
  Connector Type conn1 = { Roles ... }; }

Family sub_fam extends fam with {
  Component Type sub_comp1 extends comp1 with {
    Port p1 = { Property attach : int <<default = 1>>; }
  }
  Component Type comp3 = { ... }
}
    
```

Declaration in ACME of a family of architectures, **fam**, and its subfamily, **sub_fam**, which has new components and properties

ADLS FOR SoS

SoS characteristics	Do Single System ADLs cope with SoS characteristics?
<i>Operational independence of constituent systems</i>	No, they do not. Single system ADLs are based on the notion that components' operation is totally controlled by the system, which is not the case for constituents. Moreover, the concrete components of single systems are known at design-time, which is not necessarily the case of SoSs either.
<i>Managerial independence of constituent systems</i>	No, they do not. Single system ADLs are based on the notion of components whose management is totally controlled by the system, which is not the case of SoSs.
<i>Geographical distribution of constituent systems</i>	No, they do not. Single system ADLs are based on the notion of logically distributed components. None supports the notion of physical mobility, in particular regarding unexpected local interactions among components that physically move near to each other, as it is the case of SoSs.
<i>Evolutionary development of SoS</i>	No, they do not. Single system ADLs are based on the principle that concrete components are known at design-time and that they may possibly enter or leave the system at run-time under the control of the system itself, which is not necessarily the case of SoSs.
<i>Emergent behavior drawn from SoS</i>	No, they do not. Single system ADLs have been defined based on the principle that all behaviors are explicitly defined (including global ones). None supports the notion of emergent behavior required in SoSs.

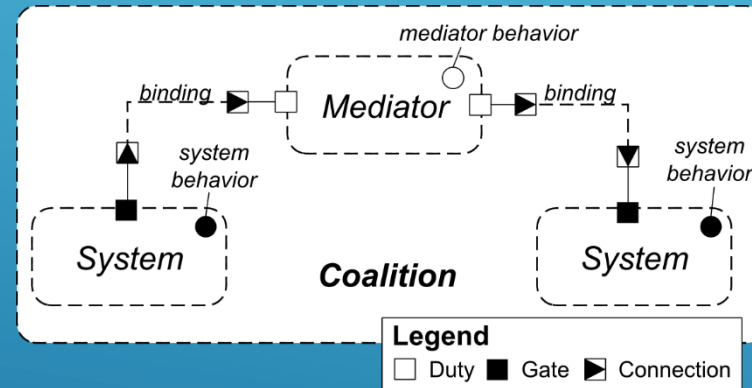
SOSADL

AN ARCHITECTURE
DESCRIPTION

LANGUAGE FOR SOS

▶ Description of an abstract architecture for SoS

▶ It can be evolutionarily concretized at run-time by identifying and incorporating concrete constituent systems

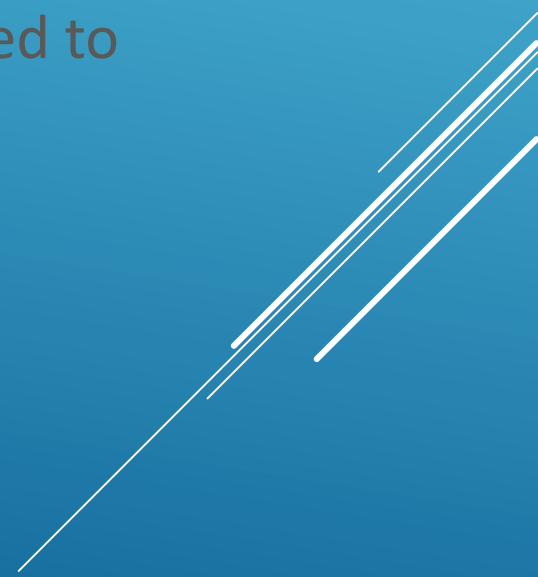


Coalition represents on-the-fly composition of systems (i.e., constituents)

SoSs Research Directions



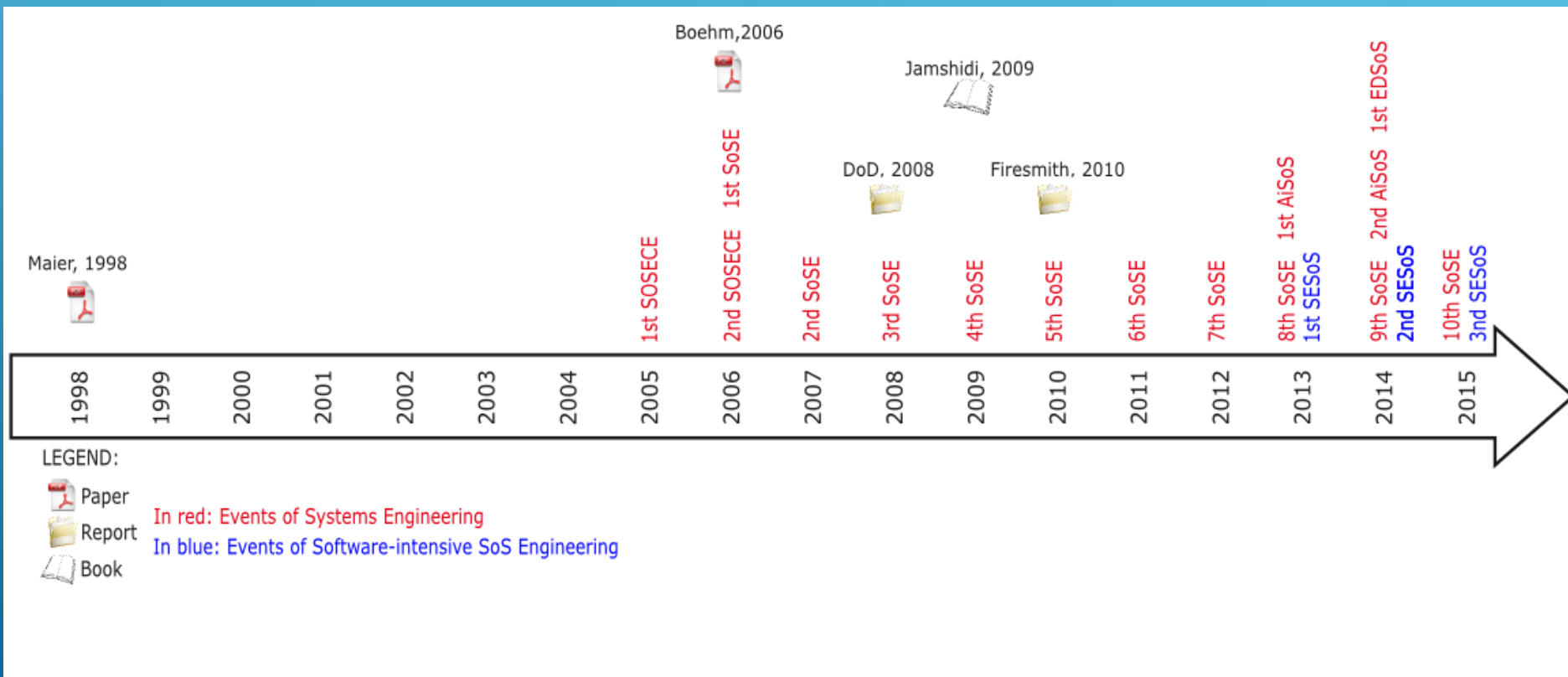
A Roadmap for SoS

- Some challenges:
 - Deal adequately with SoS software architectures
 - Investigate how to develop SoS for diverse domains
 - Propose solutions to different types of SoS
 - What are the fundamental challenges we need to address in the SEng Community?
- 

Research Directions

- Formal ADLs for SoSs
 - Promote correctness, consistency, and completeness of architecture descriptions
 - Support evolutionary development of SoSs
- Desired properties of ADLs for SoSs
 - Understandability,
 - Scalability,
 - Refinement,
 - Traceability, among others others
- Support different phases of SoS life cycle
 - Enforce correctness, consistency, and understandability of architecture descriptions
 - Ensure semantic consistency among heterogeneous models of constituents
 - Interchangeable, complementary techniques should be explored for supporting different abstraction/formalism levels

Main Publications and Venues



Literature on SoS

- B. Boehm and L.A. Lane, “21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems”. The Journal of Defense Software Engineering, vol. 19, no. 5, 2006, pp. 4-9.
- Department of Defense, “Systems Engineering Guide for Systems of Systems”. Aug. 2008; www.acq.osd.mil/se/docs/SE-Guide-for-SoS.pdf.
- D. Firesmith, “Profiling Systems Using the Defining Characteristics of Systems of Systems (SoS)”,. Technical Report CMU/SEI-2010-TN-001, Software Engineering Institute, Carnegie Mellon University, 2010.
- M. Jamshidi, ed., “System of Systems Engineering: Innovations for the Twenty-first Century”. Wiley & Sons, 2009, p. 616.
- M.W. Maier, “Architecting Principles for Systems-of-Systems”. Systems Engineering, vol. 1, no. 4, 1998, pp. 267-284.
- C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, “Systems of systems engineering: Basic concepts, model-based techniques, and research directions”. ACM Computing Survey, 48, 2, 2015.

Bibliography Part I

- Bass, L., Clements, P., and Kazman, R. 2003. Software Architecture in Practice (2ed.). Addison-Wesley Longman Publishing Co.
- Gorton, I. 2006. Essential Software Architecture. Springer-Verlag New York, Inc.
- Kruchten, P. What do software architects really do? In: Journal of Systems and Software, v.81, p.2413-2416. 2008
- Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A. and America, P. A general model of software architecture design derived from five industrial approaches. In: Journal of Systems and Software, v.80, n.1, p. 106-126. 2007.
- Garland, J. and Anthony, R. 2003. Large-Scale Software Architecture: A Practical Guide Using UML. John Wiley & Sons, Inc., New York, NY, USA.Hofmeister
- ISO/IEC/IEEE 42010:2010 International Standard for Systems and Software Engineering -- Architectural description
- Malavolta, I.; Lago, P.; Muccini, H.; Pelliccione, P. and Tang, A. What Industry Needs from Architectural Languages: A Survey IEEE Transactions on Software Engineering, 2013, v. 39, n. 6, 869-891.
- Lago, P.; Malavolta, I.; Muccini, H.; Pelliccione, P. and Tang, A. The road ahead for architectural languages. IEEE Software, 2014, 32, 98-105.
- Medvidovic, N. and Taylor, R. N. A classification and comparison framework for software architecture description languages. In: IEEE Transactions on Software Engineering, 2000, v. 26, n.1, 70-93.
- Oquendo, F. pi-ADL: An Architecture Description Language based on the Higher Order Typed pi-Calculus for Specifying Dynamic and Mobile Software Architectures. In: ACM Software Engineering Notes, 2004, v. 29, n.3, 15-28.
- Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Merson, P.; Nord, R.; and Stafford, J. Documenting Software Architectures: Views and Beyond. Addison-Wesley, 2011.
- Shaw, M. and Garlan, D. Characteristics of Higher-Level Languages for Software Architecture. Carnegie Mellon University, 1994. <http://www.sei.cmu.edu/reports/94tr023.pdf>

Bibliography

Part II

- Boehm, B.; Brown, W.; Basili, V. & Turner, R. Spiral Acquisition of Software-Intensive Systems-of-Systems. In: Crosstalk, 2004, p. 4-9
- Guessi, M.; Neto, V. V. G.; Bianchi, T.; Felizardo, K. R.; Oquendo, F. & Nakagawa, E. Y. A systematic literature review on the description of software architectures for systems of systems. In: ACM/SIGAPP SAC' 2015, 2015a, p. 1442-1449
- Guessi, M., Cavalcante, E., and Bueno, L.B.R. Characterizing ADLs for Software-Intensive SoS. In: SeSoS at ICSE' 2015. 2015b. p. 12-18.
- Medvidovic, N. and Taylor, R. N. A classification and comparison framework for software architecture description languages. In: IEEE Transactions on Software Engineering, 2000, v. 26, n.1, 70-93.
- Nielsen, C. B.; Larsen, P. G.; Fitzgerald, J.; Woodcock, J. & Peleska, J. Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions. In: ACM Comput. Surv., 2015, v. 48, p. 1-41
- Oquendo, F. Formally Describing the Software Architecture of Systems-of-Systems with SosADL. In: SoSE' 2016, 2016a, p.1-6
- Oquendo, F. π -Calculus for SoS: A Foundation for Formally Describing Software-intensive Systems-of-Systems. In: SoSE' 2016, 2016b, p. 1-6
- Silva, E.; Batista, T. & Oquendo, F. A Mission-Oriented Approach for Designing System-of-Systems. In: SoSE' 2015, p. 346-351.
- Ulieru, M. & Doursat, R. Emergent engineering: a radical paradigm shift. In: Int. J. Autonomous and Adaptive Communications Systems, 2011, v. 4, n.1, p. 39-60.