SoSECIE Webinar

Welcome to the
2021 System of Systems Engineering Collaborators Information Exchange (SoSECIE)

We will start at 11AM Eastern Time

You can download today's presentation from the SoSECIE Website:

https://mitre.tahoe.appsembler.com/blog

To add/remove yourself from the email list or suggest a future topic or speaker, send an email to sosecie@mitre.org

NDIA System of Systems SE Committee

- **Mission**
  - To provide a forum where government, industry, and academia can share lessons learned, promote best practices, address issues, and advocate systems engineering for Systems of Systems (SoS)
  - To identify successful strategies for applying systems engineering principles to systems engineering of SoS
- **Operating Practices**
  - Face to face and virtual SoS Committee meetings are held in conjunction with NDIA SE Division meetings that occur in February, April, June, and August

NDIA SE Division SoS Committee Industry Chairs:
Mr. Rick Poel, Boeing
Ms. Jennie Horne, Raytheon
OSD Liaison:
Dr. Judith Dahmann, MITRE

Simple Rules of Engagement

- I have muted all participant lines for this introduction and the briefing.
- If you need to contact me during the briefing, send me an e-mail at sosecie@mitre.org.
- Download the presentation so you can follow along on your own
- We will hold all questions until the end:
  - I will start with questions submitted online via the CHAT window in Teams.
  - I will then take questions via telephone; State your name, organization, and question clearly.
- If a question requires more discussion, the speaker(s) contact info is in the brief.

Disclaimer

- MITRE and the NDIA makes no claims, promises or guarantees about the accuracy, completeness or adequacy of the contents of this presentation and expressly disclaims liability for errors and omissions in its contents.
- No warranty of any kind, implied, expressed or statutory, including but not limited to the warranties of non-infringement of third party rights, title, merchantability, fitness for a particular purpose and freedom from computer virus, is given with respect to the contents of this presentation or its hyperlinks to other Internet resources.
- Reference in any presentation to any specific commercial products, processes, or services, or the use of any trade, firm or corporation name is for the information and convenience of the participants and subscribers, and does not constitute endorsement, recommendation, or favoring of any individual company, agency, or organizational entity.

2021-2022 System of Systems Engineering Collaborators Information Exchange Webinars

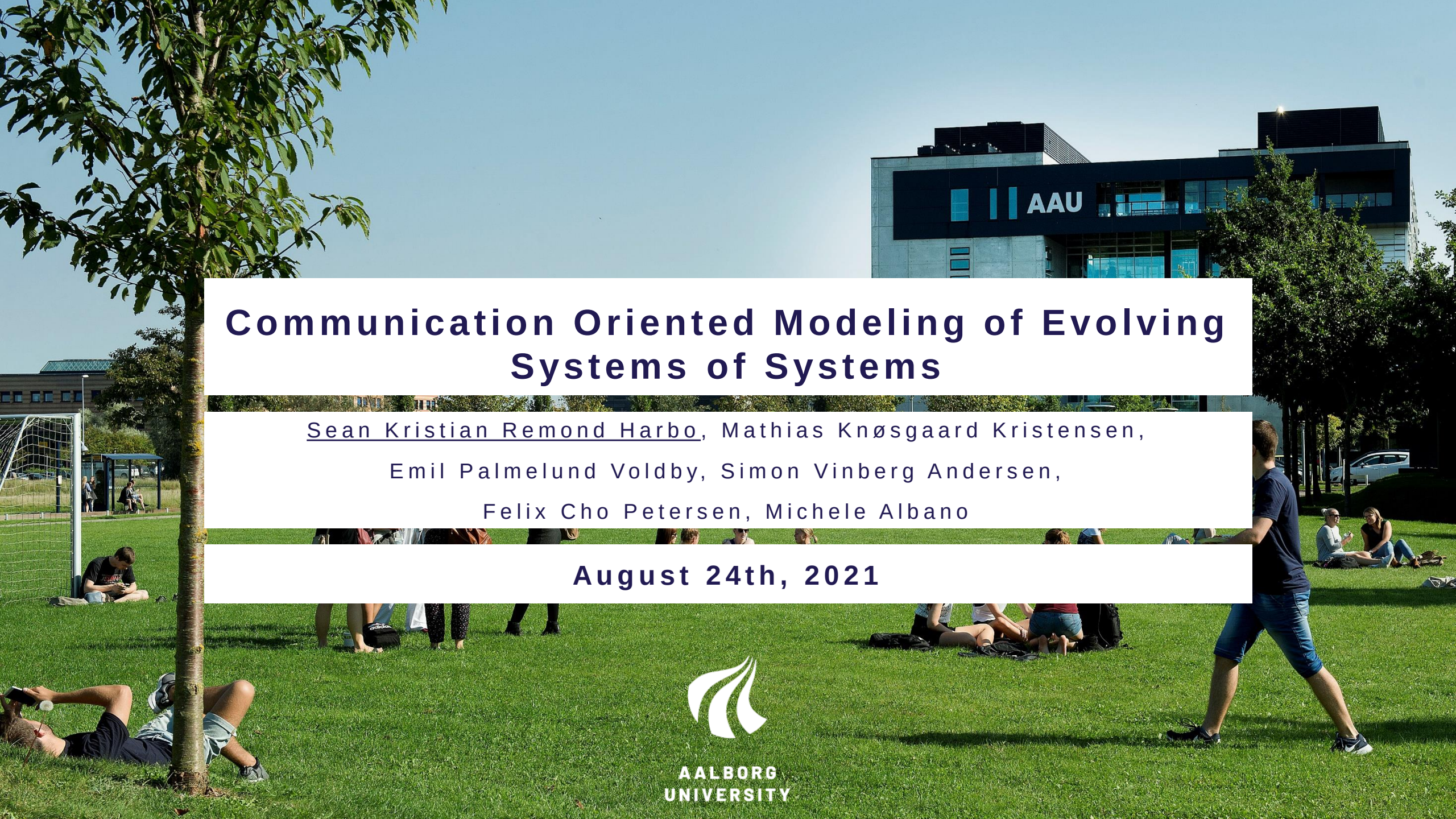
Sponsored by MITRE and NDIA SE Division

September 7, 2021
System of Systems Meta-Architecture Approach to Improve Legacy Metrorails for Enhanced Customer Experience
Dr. Cihan Dagli and Maxwell Polley

October 19, 2021
Resilience in Systems of Systems: Electrified Transport Systems
Pontus Svenson, Kerstin Eriksson, and Sara Janhäll

November 16, 2021
A Design Method for Collaborative Systems of Systems Applied to Metropolitan Multi-Mode Transport System
Pontus Svenson, Frida Reichenberg, and Jakob Axelsson

November 30, 2021
Should I Stay or Should I Go? How Constituent Systems Decide to Join or Leave Constellations in Collaborative SoS
Pontus Svenson and Jakob Axelsson



Communication Oriented Modeling of Evolving Systems of Systems

Sean Kristian Remond Harbo, Mathias Knøsgaard Kristensen,

Emil Palmelund Voldby, Simon Vinberg Andersen,

Felix Cho Petersen, Michele Albano

August 24th, 2021

AALBORG UNIVERSITY

CONTENT

- Introduction to ACS
- Processes and approaches
- The ACS modeling framework
- Proof of concept
- Conclusion

Definitions

- ACS: The “Abstract Communicating Systems” Modelling Framework.

- Atomic System (AS): A system that cannot be broken down into a composite without loosing operational or managerial independence.
- Composite: A system that is composed of other composites and ASs.

Our motivation

- SoSE and support tools thereof are becoming increasingly important and complex.
- Such tools may reduce cost and increase maintainability of SoSs though model-based verification.
- Must consider that systems in a SoS can:
  - Pertain to different stakeholders
  - Evolve over time.

- Such a tool could use existing tools for verifying operational properties/correctness (e.g. UPPAAL) and for generating code for network interfaces of ASs (e.g. OpenAPI Generator).

Related works

- Tool suites DANSE and COMPASS use SysML to model SoSs and support the verification and simulation thereof.
- Extensive modeling capabilities with an emphasis on internal computations results in complex models.
- DANSE uses statistical model checking for verification. COMPASS does not support time constraints in behavior.
- Current approaches require the designer to get familiar with custom tools and languages.

Other observations

- Current tools don’t explicitly support the evolution of SoSs.
- Behavior of a SoS is attributed to that of its CSs more than the interactions between CSs.
- Too many details in a model approaches an implementation.
- Too few details limits the designer’s ability to express themself.

The problem

- How can we make a tool that finds a good compromise between complexity & simplicity and directly supports continuous evolution & verification, while still having high expressivity?

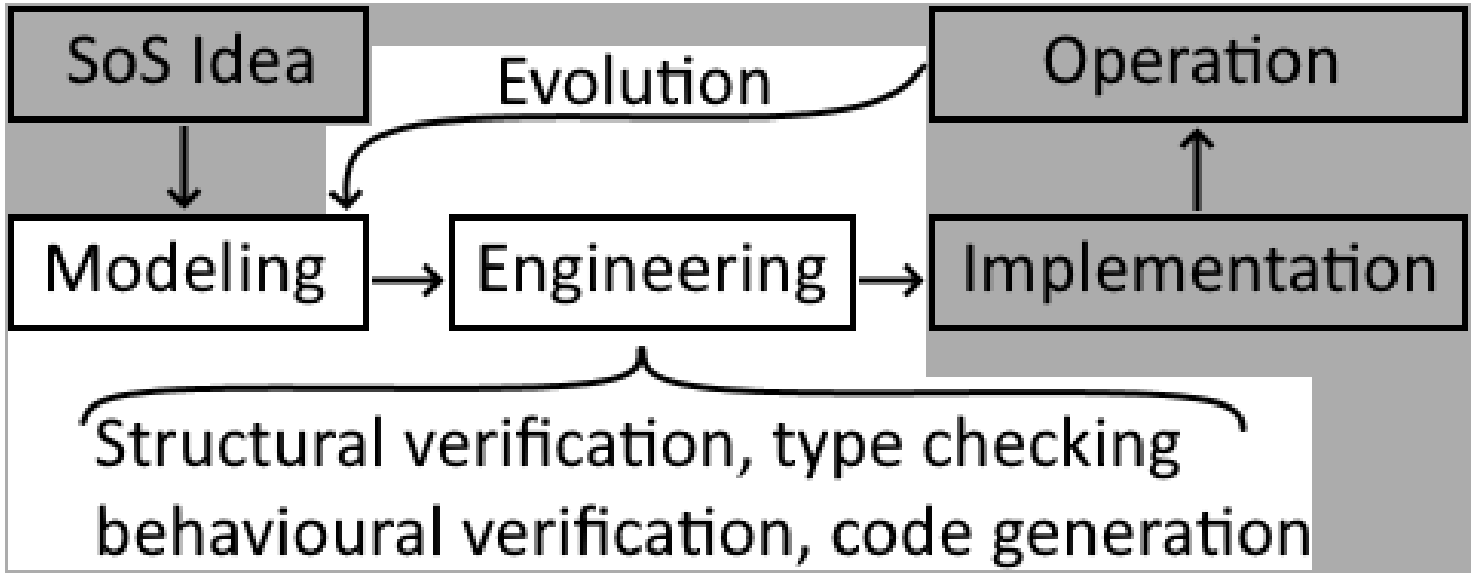
Main contribution

Abstract Communicating Systems (ACS) Modeling Framework:

- A novel framework for modeling evolving SoSs based on a model of a SoS's Network Structure, Communication Patterns, and Structure & Visibility of communicated data.
- ACS models the externally visible properties of a SoS, with a focus on communication activities.

The ACS development process

- The lifecycle of a SoS is an iterative process.
- ACS focuses on assisting the Modeling, Engineering, and Evolution phases.



The modeling approach

- Only Atomic Systems (ASs) can communicate. Composite Systems cannot (but the ASs inside them can).
- A SoS model cares not how ASs are implemented. Only how they communicate.

Structure of a model

An ACS model captures the following properties of a SoS:

- Network Structure.
- Communication Patterns.
- Structure & Visibility of Communicated Data.

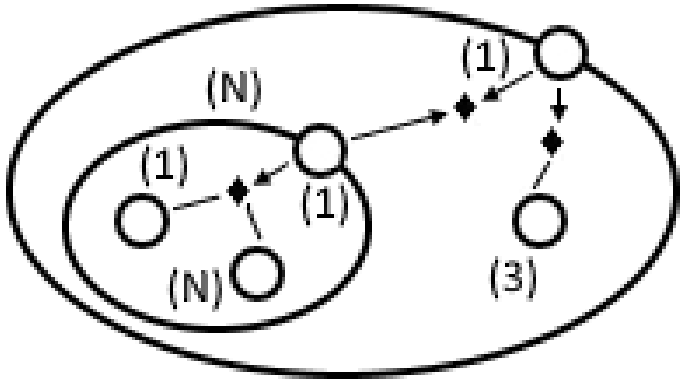
Running example: Bluetooth tracker

- Goal: To mine patterns in the stores/shops people visit.
- Shops track Bluetooth IDs of visitors' smart devices and send IDs to a central computer for analysis.
- Used throughout the description of the framework.

Network Structure

Simplified example

- Recursive hierarchy of Composites and interlinked ASs.
- References to models of existing systems allows for reuse.
- Only ASs with directed Link Connections may initiate communication.
- Constituent Systems (CSs) have a count denoting the number of instances that exist in operation.

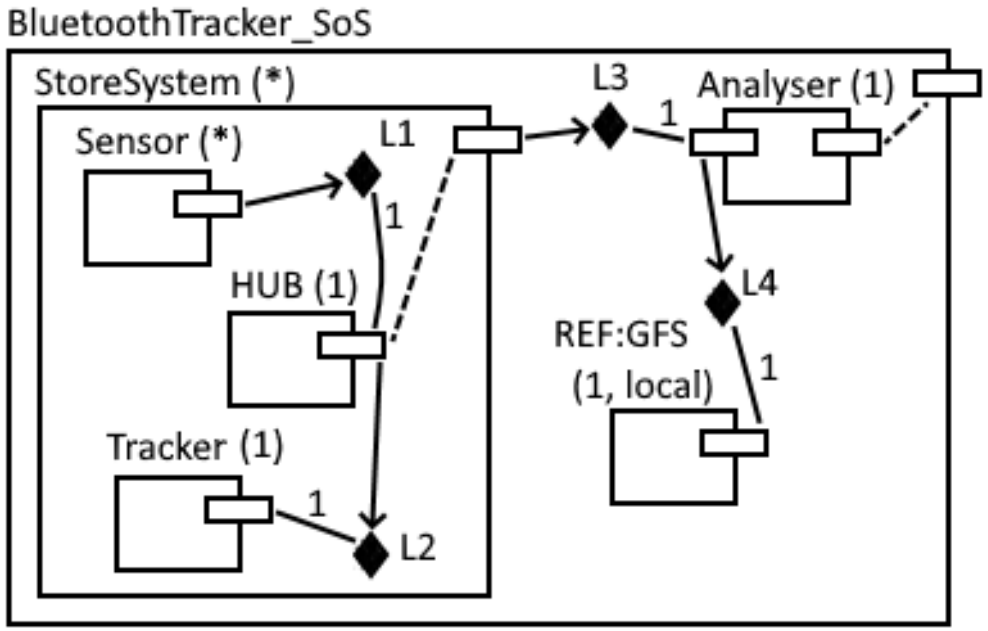


- Network Structure is agnostic to the SoS's physical host architecture.

Network Structure

Running example with full syntax

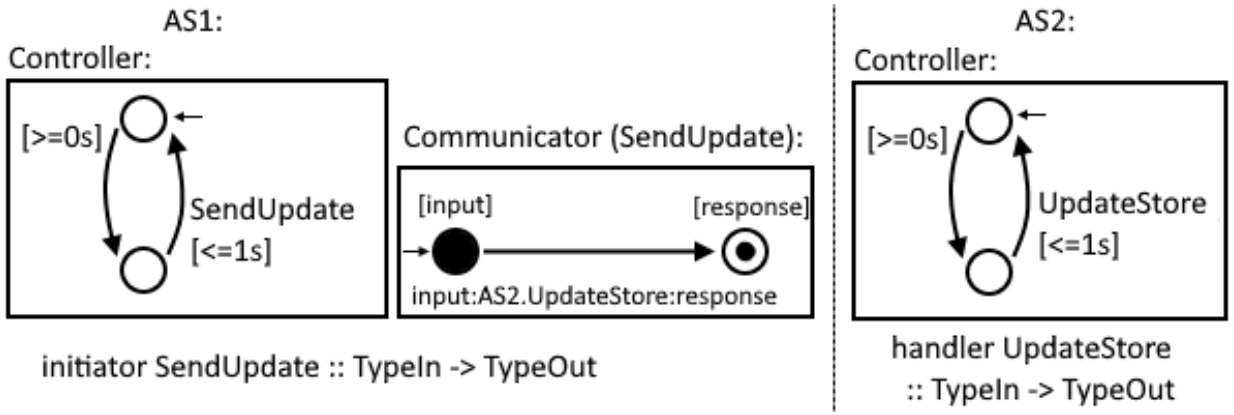
- Sensors send Bluetooth IDs to HUB, which send visitation data to Tracker and Analyser.
- Analyser stores results in Google File System.



Communication Pattern

Simplified example

- Controllers use typed events and time constraints to describe the behavior of an AS.
- Events invoke communicators which describe data flow and communication between ASs.
- Events can be an "*initiator*" (initiates communication) or a "*handler*" (continues or terminates communication).

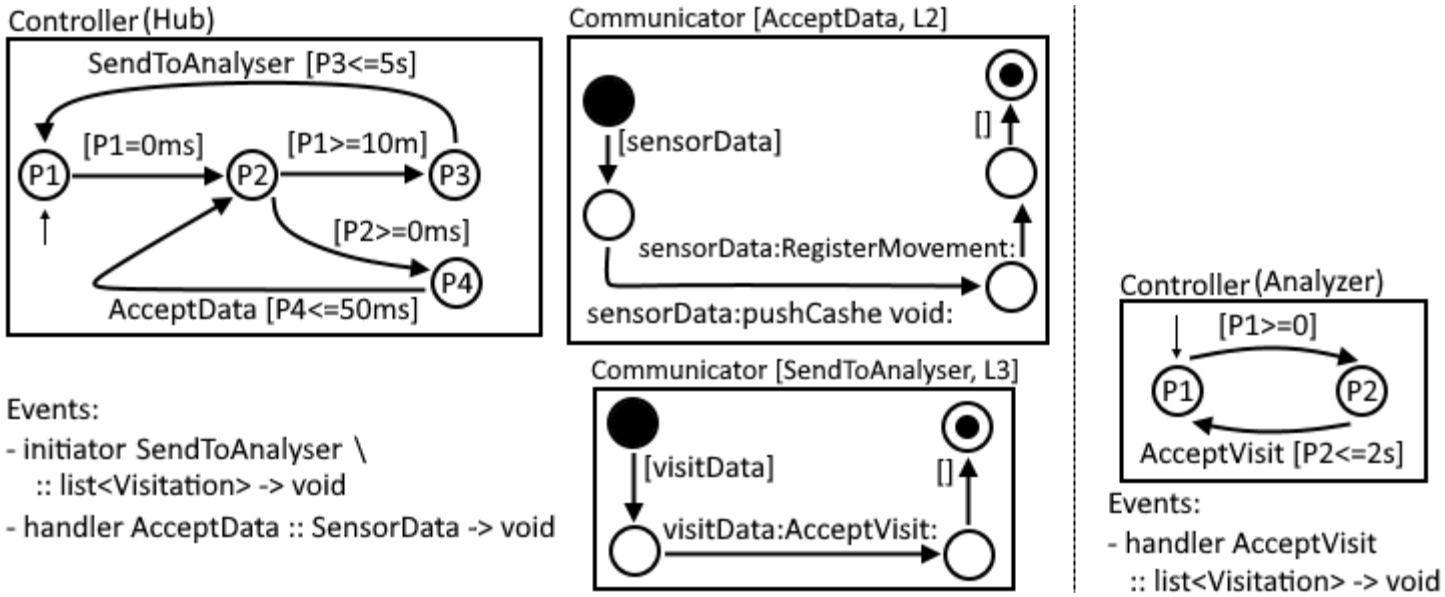


- Only model the structure of communicated data to avoid dealing with concrete values.
- Describe chains of communication events where internal computation is abstracted as delays.

Communication Patterns

Behavior of HUB and Analyzer with full syntax

- HUB receives SensorData from Sensors, stores it in a cache, and sends it to Tracker.
- HUB sends a list of Visitations to Analyzer every 10 minutes.



- Faults such as unsatisfiable constraints or untraversable edges are detectable with this approach.

Structure & Visibility of Data

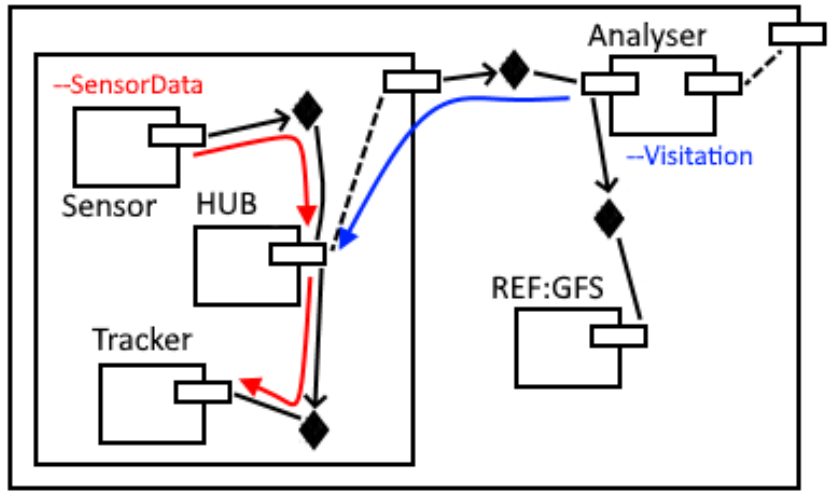
Structure & definition

- The structure of data is described using Data Transfer Objects (DTOs) with typed, named fields.
- Use Object Constraint Language (OCL) constraints on DTOs to limit field values at run-time.
- DTOs support single-inheritance and polymorphism.
- Each DTO is defined/controlled by a single AS, around which the DTO’s scope is centered.
- Scopes are tree-shaped. DTOs propagate though Links to the ASs that use them.

Structure & Visibility of Data

Simplified scope rules & example

- If an AS A can initiate communication with another AS B, then DTOs can propagate between A and B.
- If a propagated DTO is specialized, the specialization cannot propagate back.

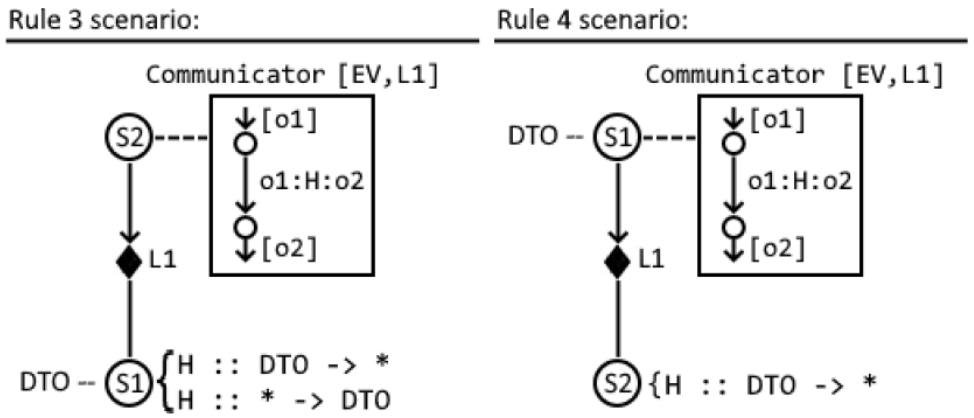


- If the (only) definition of Visitation is modified, then all systems using it will register this immediately.
- Supports modelling interconnected systems from multiple stakeholders.

Structure & Visibility of Data

Full scope rules (aka. propagation rules)

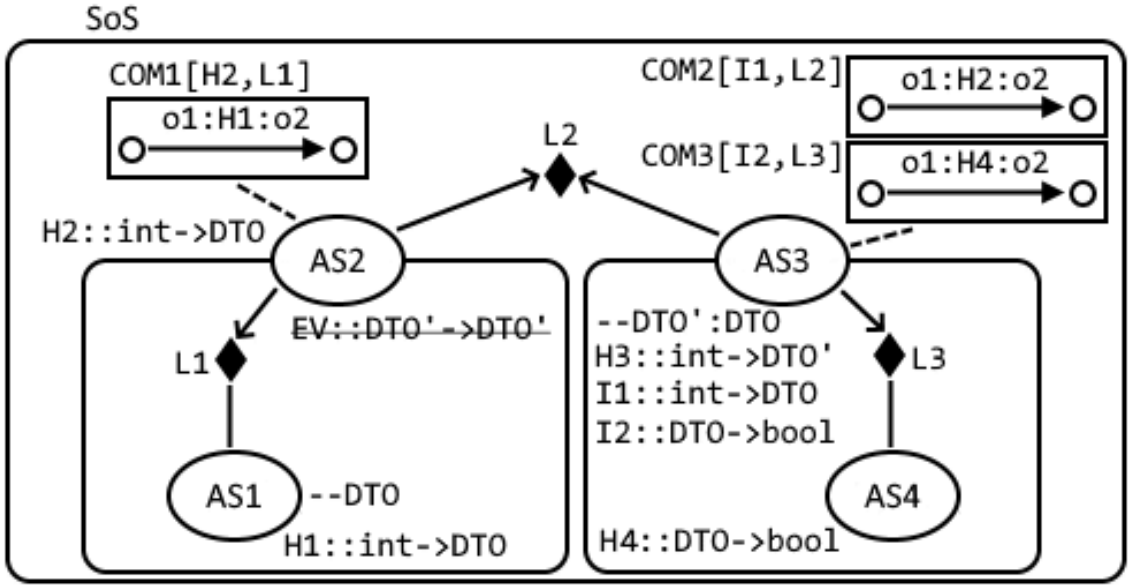
1. Two systems S1 and S2 must be connected by a link, where at least one system must be able to initiate communication (directed link connection).
2. To be propagated to S2, a DTO must have propagated to (or be defined at) S1, but not S2. If it exists, the base of the DTO cannot have propagated from S2 to S1.
3. In case S2 can initiate communication and S1 cannot => A communicator on S2 is able to invoke a Handler on S1 which has DTO as input and/or output.
4. In case S1 can initiate communication and S2 cannot => A communicator on S1 is able to invoke a Handler on S2 which takes DTO as input.
5. In case both S1 and S2 can initiate communication => At least one of the implications from rules 3 and 4 hold true.



Structure & Visibility of Data

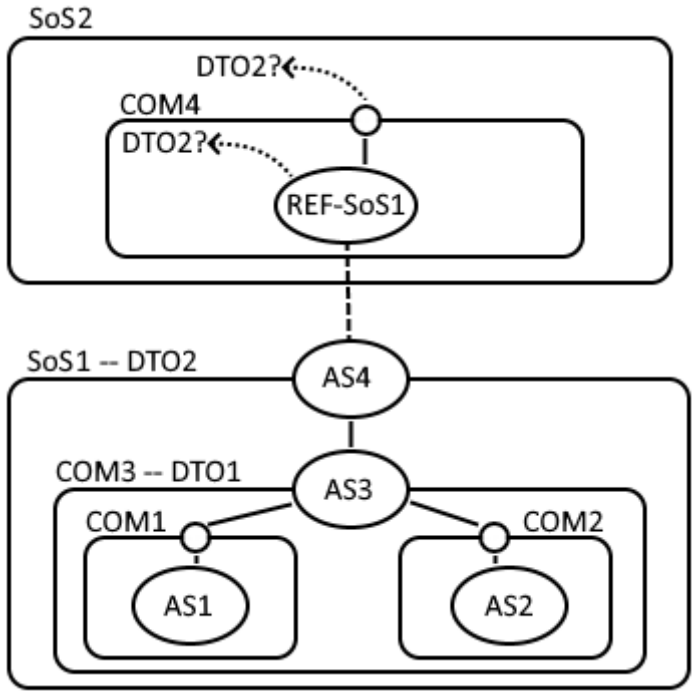
Example with full scope rules

- *DTO* is defined on AS1 and can propagate all the way to AS4, across two interoperating Composites.
- The specialization DTO' of DTO can, as per the exception in rule 2, not propagate to AS2.



Rejected Alternative Scope Rules

- Composite-based scopes may seem natural, where DTOs are visible inside the defining Composite, but:
- Maintaining the best Composite of definition scales badly with DTO count and SoS size.
- Inconsistent visibility: What if AS3 outputs DTO1? What if AS4 outputs DTO2?



Engineering phase

Support through automatic verification process:

- Structure: verify that the SoS is structured legally.
- Patterns and Structure: verify that all communication paths are reachable and legal according to Links.
- Data, Patterns, and Structure: verify that all data flow is type correct and all DTOs are used within scope.

Evolution phase

- The systems in a SoS can change (evolve) over time.
- The previous phases are designed to ease the adaption to new requirements.
- Updating a DTO’s single source of definition propagates changes to the ASs which use the DTO as delimited by the DTO’s scope.
- The verification processes captures faults introduced during evolution, possibly through cascading effects.

Proof of Concept

Overview

We made a preliminary prototype tool:

- Models are made using Eclipse Papyrus.
- Papyrus file is mapped to an Abstract Syntax Tree (AST).
- Verify structure and type/scope check model based on AST.
- Controllers and Communicators are mapped to UPPAAL templates and queries and then verified.
- WIP: Code generation for REST clients and servers using OpenAPI.

- Note: Due to similarities between ACS and UML, ACS could possibly be defined as a UML meta model.

Video demonstration

Benefits provided by ACS

Why choose ACS (a possible UML meta model) over pure UML/SysML?

- DTOs' single source of definition and their scopes impose additional structure on models.
- ACS verifies that a SoS's structure is legal and that all communication paths are reachable & type-correct.
- ACS ensures errors are caught during evolution, even when multiple stakeholders collaborate on a SoS.

Takeaways

- ACS allows to model SoSs based on Network Structure, Communication Patterns, and Data Structure & Visibility.
- ACS supports evolution of SoSs with multiple stakeholders through a continuous verification process.

Future works

- A complete, formal semantics. Current, unfinished semantics cause some undefined behavior.
  - E.g., there are no semantics for references.
- Finalize generation of easily replaceable code-frameworks for client and server interfaces.
- Integrate ACS into Papyrus as a plugin instead of being an external tool.

THANK YOU

AALBORG
UNIVERSITY