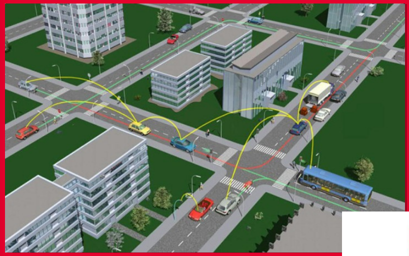
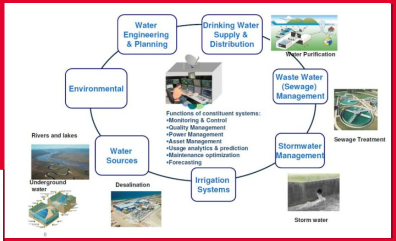
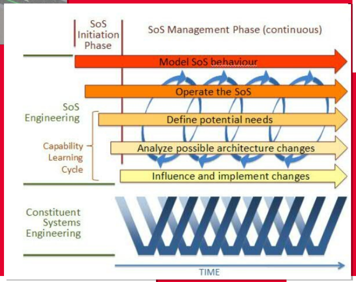





SoS Initiation Phase
SoS Management Phase (continuous)
Model SoS behaviour
Operate the SoS
SoS Engineering
Define potential needs
Capability Learning Cycle
Analyze possible architecture changes
Influence and implement changes
Constituent Systems Engineering
TIME

Surveillance System
FMS
Datalink Communications
Airlines Operations
Airport Surface Control
GPS
4D Trajectory Exchange
ADS Broadcast
Galileo
System Wide Information Management
Air Traffic Control
Cooperative Flight Data Processing System and traffic flow management
Control Tower

Aircraft
Meteo
Airport
ATFCM
ATC Centre
1 A-SMGCS
2 Navaids
3 Runway Lighting System, Runway Occupancy System, Stop bars
4 Airport Centre
5 Aircraft via data link
6 Meteo Centre
7 Air Traffic Flow Management Centre
8 APP Control Centre

Applying SMC to Systems of Systems

Axel Legay,
Benoît Boyer, Louis-Marie Traonouez, Jean Quilbeuf

November 2015

Systems of Systems

A <u>System of Systems</u> (SoS) composes <u>Constituent Systems</u> (CS) that:

- Operate <u>independently</u>
- Are owned and <u>managed by different parties</u>
- Are constantly <u>evolving</u>
- Are <u>geographically distributed</u>

to provide an <u>emergent behavior</u> that no CS alone can provide.

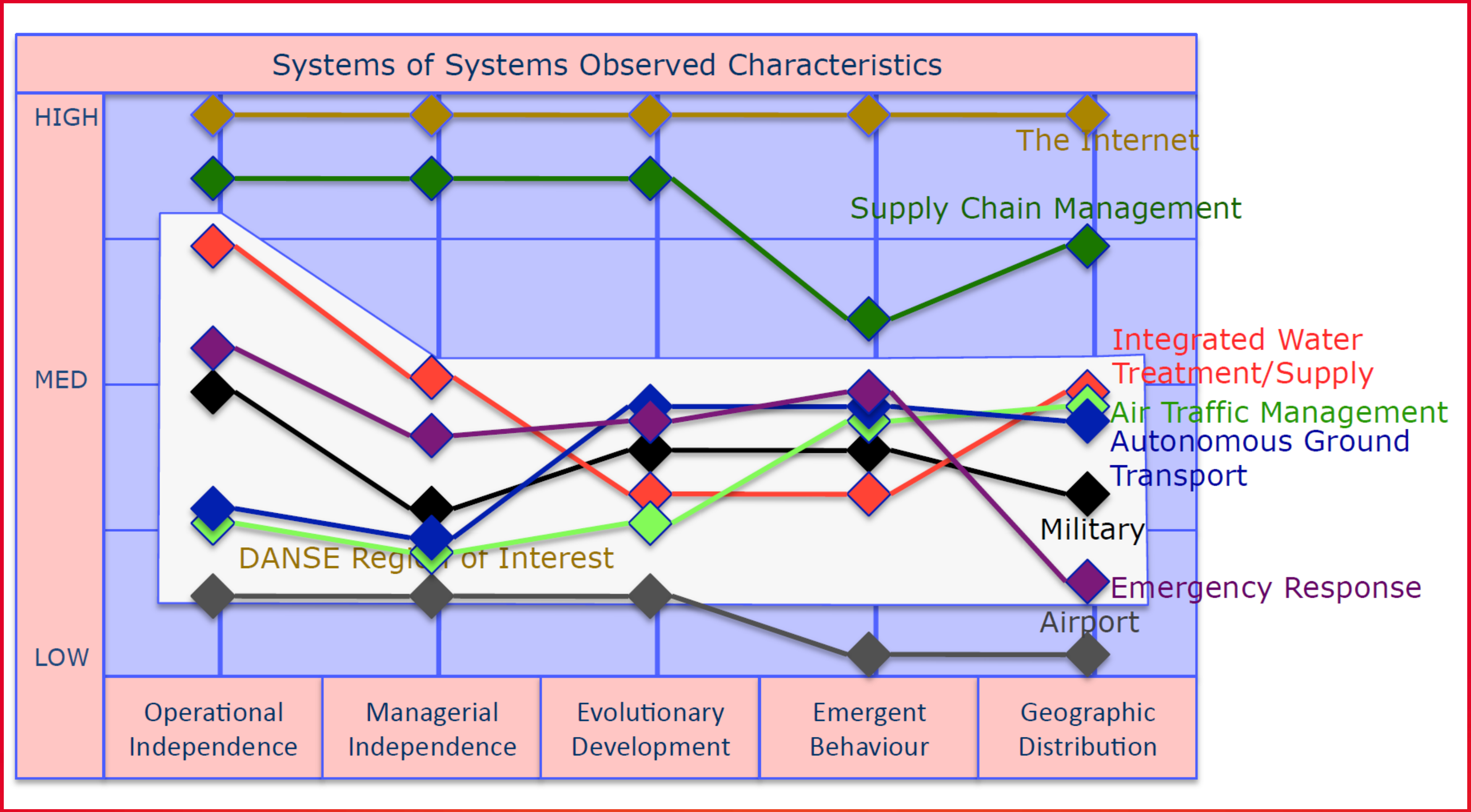
Example of SoS: The Internet

Internet is composed of several interconnected networks that

- Operate independently
- Are owned and managed by different parties
- Are constantly evolving
- Are geographically distributed

to provide an emergent behavior: worldwide routing of packets.

Type of SoS considered



Architecture of such SoSs is usually managed by a single party.

Evolution of the SoS requires fast decision making.

Challenges for SoS

Modeling:

- Constituent systems are modeled in <u>various languages</u>
- Architecture need to compose <u>heterogeneous Constituent Systems</u>

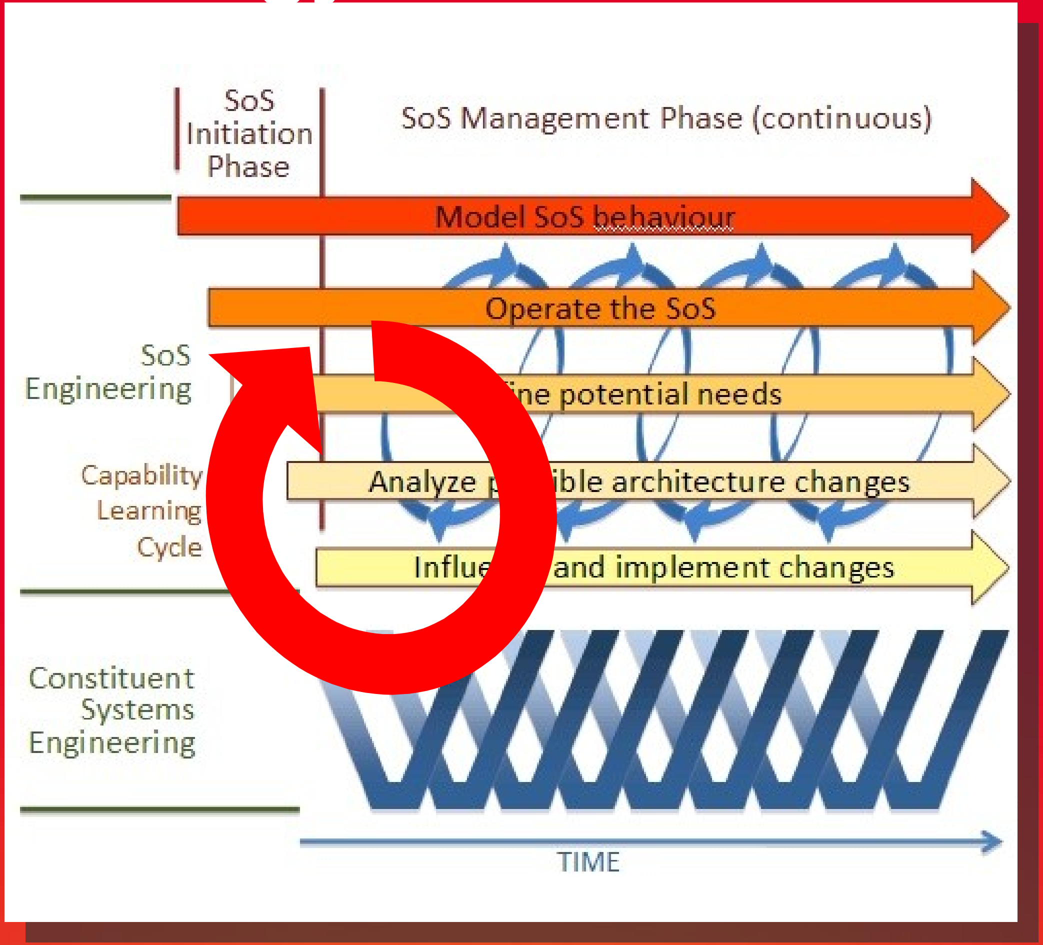
Validation

- SoS are very large systems: <u>exhaustive exploration is doomed to fail</u>

Emergent behavior

- Expected behavior might not show up because composition triggers a <u>unexpected emergent behavior</u>
- Global behavior hard to infer from behavior of each component
- Handling dynamicity is complex !

Methodology


SoS Initiation Phase
SoS Management Phase (continuous)
Model SoS behaviour
Operate the SoS
SoS Engineering
ine potential needs
Capability Learning Cycle
Analyze p ible architecture changes
Influe and implement changes
Constituent Systems Engineering
TIME

The continuous evolution of an SoS requires iterative analyzes.

Outline

1

Modeling SoSs

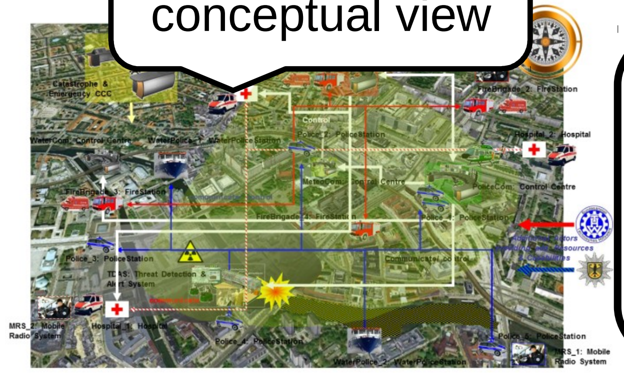
Architecture and Constituent Systems

SoS Modelling Approach

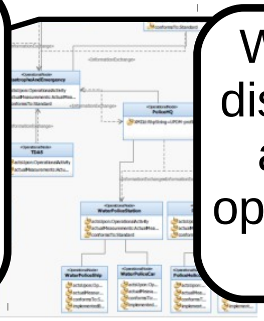
- Reuse existing models for constituent systems
  - 羊Models created during the engineering of CS
  - 羊Different formats for different analyses
- SoS as a composition of black boxes
  - 羊Architecture:connections between CS
    - Described in UPDM (multiple views, as in UML)
  - 羊Constituent systems abstracted by their interface
    - Follows the FMI standard
    - FMU for joint simulation (compiled from model/code of CS)

Modelling Architecture in UPDM

Different Views to describe the SoS



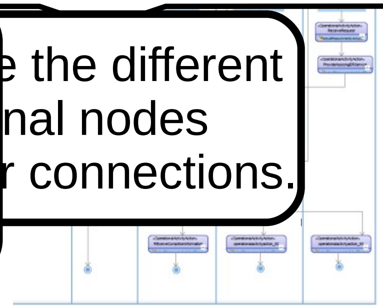
High level conceptual view



Operational resource flow description

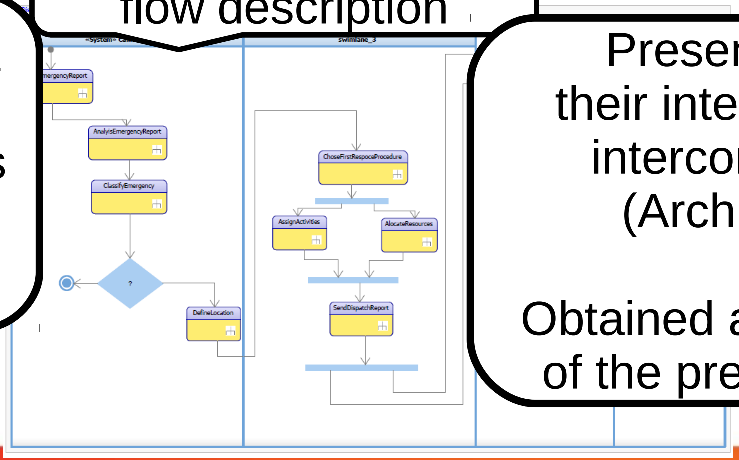
Illustrative representation of the CSs and their interactions.

Workflow distributed among operational nodes



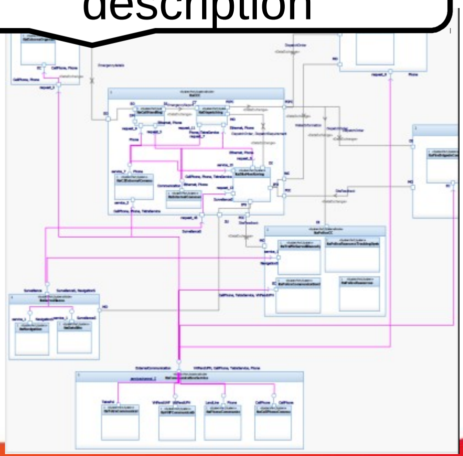
Operational activity model

e the different nal nodes r connections.



System functionality flow description

Shows how a functionality is obtained as a flow of activities



System Interface description

Presents CSs, their interfaces and interconnections (Architecture)

Obtained after analysis of the previous views

Probablistic Behavior

- Random variables are needed
  - Models inputs of the system (value of a sensor)
  - Models unknown timing (time to a failure)

- UPDM extended with a probabilistc sterotype
  - Applicable to attributes of each component system in the SV-1 (Architecture) view
  - Such attributes become random variables
    - « observe » function that samples a new value at each call (automatically generated)
    - Several probability distributions are available : uniform, normal or custom

Modeling Constituent Systems

Any modeling framework that can export FMU can be used.

```
<?xml version="1.0" encoding="UTF8"?>
<fmiModelDescription fmiVersion="1.0" modelName="ModelicaExample" modelIdentifier="ModelicaExample_Friction" ...
   <UnitDefinitions>
      <BaseUnit unit="rad">
         <DisplayUnitDefinition displayUnit="deg" gain="23.26"/>
      </BaseUnit>
   </UnitDefinitions>
   <TypeDefinitions>
     <Type name="Modelica.SIunits.AngularVelocity">
        <RealType quantity="AngularVelocity" unit="rad/s"/>
     </Type>
   </TypeDefinitions>
   <ModelVariables>
      <ScalarVariable name="inertia1.J" valueReference="16777217" description="Moment of inertia" variability="parameter">
         <Real declaredType="Modelica.SIunits.Torque" start="1"/>
      </ScalarVariable>
   ...
   </ModelVariables>
</fmiModelDescription>
```

Define a variable in the interface.

Link to the model done by the exporting tool.

2

Describing Goals

GCSL Patterns

Goals

Goals express requirements of the SoS.

- Expressed as contracts i.e. (assume, guarantee)
  - Attached to a component
    - Contract of that particular component
  - Or global to the SoS
    - Capture a behavior resulting of the composition : **emergent behavior**

- Designed with usability in mind
  - OCL for quantifiers and atomic properties
  - Patterns for expressing temporal properties

Modeling Goals

Goals are described in GCSL, that mixes:

- Temporal operators (LTL) through pre-defined patterns (next slide)
- OCL constraints

Values exchanged between Constituent Systems are visible:

- `district.firearea` is the value `firearea` sent by the CS `district`

Collections of CS are obtained through OCL-like constructs:

- `SoS.itsDistricts` is a collection of all CS of type district in the model
- `SoS.itsDistricts->forAll( d | <expr>(d) )` is true if the expression `<expr>` holds for each district d
- `SoS.itsDistricts.firearea->sum()` is the sum of the the `firearea` attributes of all districts

Patterns

Patterns express requirements in an intuitive way

- Over 1300s, there is no significant fire in district 5 for at least 99% of the time

```
at the end of [0,1300], [district5.fireArea < 0.01]
        has been true at least [ 99 ] % of time
```

- Include only atomic OCL propositions between [ ], no nested patterns

Generic patterns independent of the architecture

- On every district, the fire area is below a given threshold:

```
SoS.itsDistricts->forAll(district | always
  [ district.fireArea * 1000000 < 1.0 ] )
```

- Patterns might be quantified, or contain quantifiers

GCSL semantics is defined by transformation to BLTL

- Each pattern is translated to a BLTL pattern
- Quantified expressions are unfolded according to the (static) architecture. For instance `c->forAll(d| f(d))` is replaced by $f(d_1) \wedge f(d_2) \wedge \ldots \wedge f(d_n)$ where $d_1, d_2, \ldots, d_n$ are the elements of the collection c

Selected Patterns and their Translation

	GCSL Pattern	BLTL Translation
2	always [Ψ]	$G_{\leq k}(\Psi)$
3	whenever [Ψ_1] occurs [Ψ_2] holds	$G_{\leq k}(\Psi_1 \rightarrow \Psi_2)$
	...	...
8	[Ψ_1] occurs at most n times during [a,b]	$occ(\Psi_1, a, b) \leq n$
	...	...
12	whenever [Ψ_1] occurs [Ψ_2] occurs within [a,b]	$G_{\leq k-b}(\Psi_1 \rightarrow X_{\leq a} F_{\leq b-a} \Psi_2)$
13	always during [a,b], [Ψ] has been true at least [e] % of time	$G_{\leq b}(\#Time < a \vee dur(\Psi) \geq (\frac{e}{100} * \#Time))$
14	at [b], [Ψ] has been true at least [e] % of time	$F_{\leq b}(dur(\Psi) \geq \frac{e}{100} * b)$

$G_{\leq t}, F_{\leq t}$: time bounded temporal operators (always and eventually)
k : maximum simulation duration
a, b : timings such that $a \leq b \leq k$
$occ(\Psi_1, a, b)$ is the number of occurrences of Ψ_1 between a and b
$dur(\Psi)$ is the time during which Ψ was true since the beginning
$\#Time$ is the time elapsed since the beginning of the simulation

3

Simulating SoSs

FMI/FMU, Master Algorithm

Simulation

The architecture (i.e UPDM model) knows only about the interface of the Constituent Systems

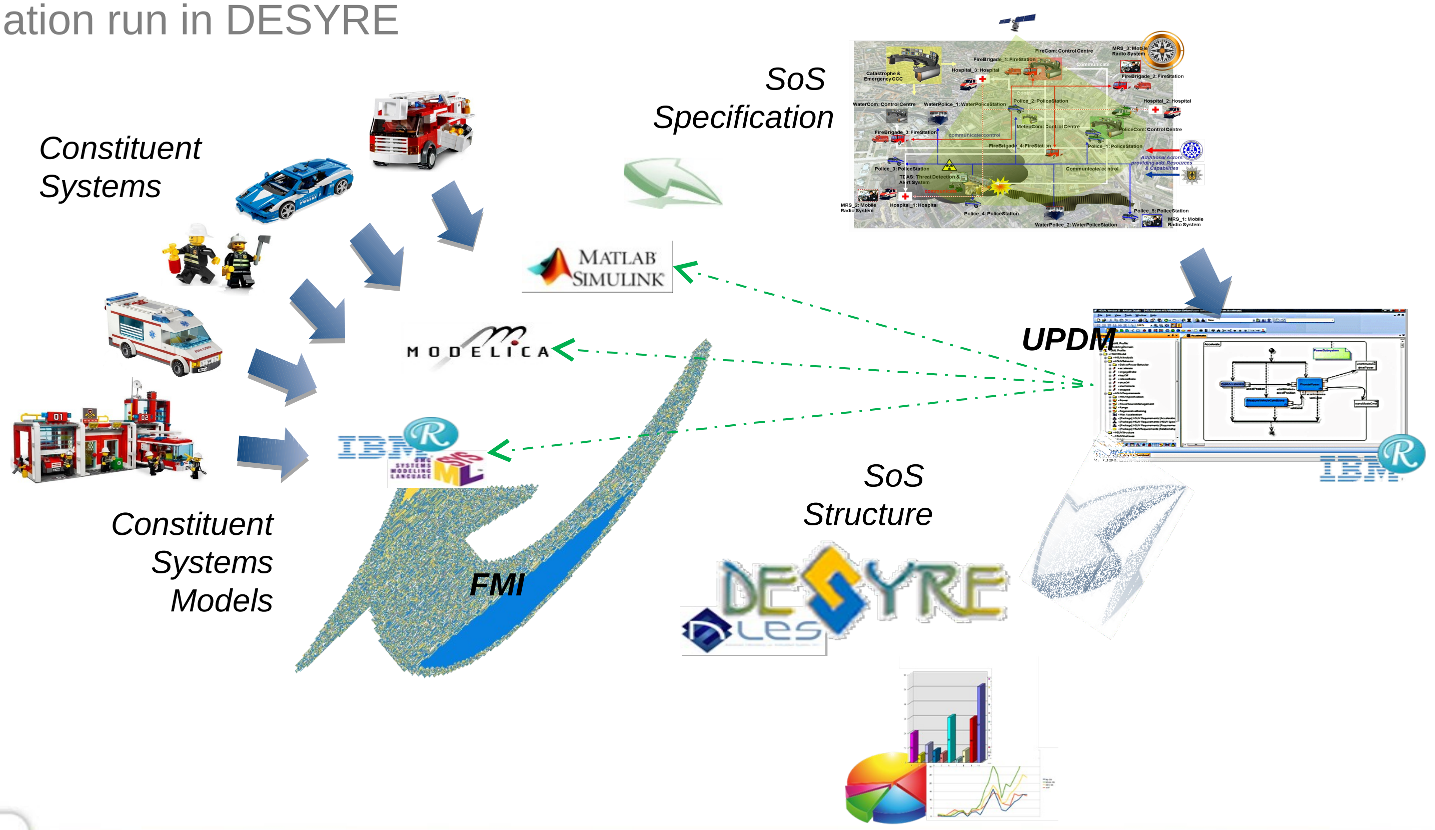
- FMI (Functional Mockup Interface) standard

For the simulation, each CS is compiled to a FMU (Functional Mockup Unit) that

- Implements its behavior
- Contains equations describing the behavior of its continuous time variables

Joint simulation

- FMI standard for component integration
- Constituent system models exported as FMUs from modeling tools
- SoS architecture exported to DESYRE
- FMUs imported in DESYRE
- Simulation run in DESYRE



Master Algorithm for Simulation

Challenges

- Correction w.r.t. computing models of arch and Cs
- Convergence of the step (for continuous variable)
- Determinism

- 2 approaches for simulating composition of FMU
  - Co-simulation: continuous variable evolution computed by the FMUs
  - Model Exchange: FMUs provide their model to the MA, which computes everything
- We selected model exchange

Master Algorithm for Simulation

```
while ($simTime \leq simEndTime$ and not($simStopEvt$)) do
    while (not($isSoSFixPtReached()$)) do
        for all $cs \in csList$ do
            $cs$.updateDiscrState($simTime$);
        end for
    end while
    for all $cs \in csList$ do
        $cs$.updateContState($simTime$);
    end for
    $evtQueue$.updateEvts();
    $simTime = evtQueue$.getClosestEvtTime();
    waitNextActivationEvt();
end while
```

Perform discrete updates until no events remain to be processed.

Update continuous variables

Update events and time

Waits until a new state is asked

4

Tool-Chain

Tool chain

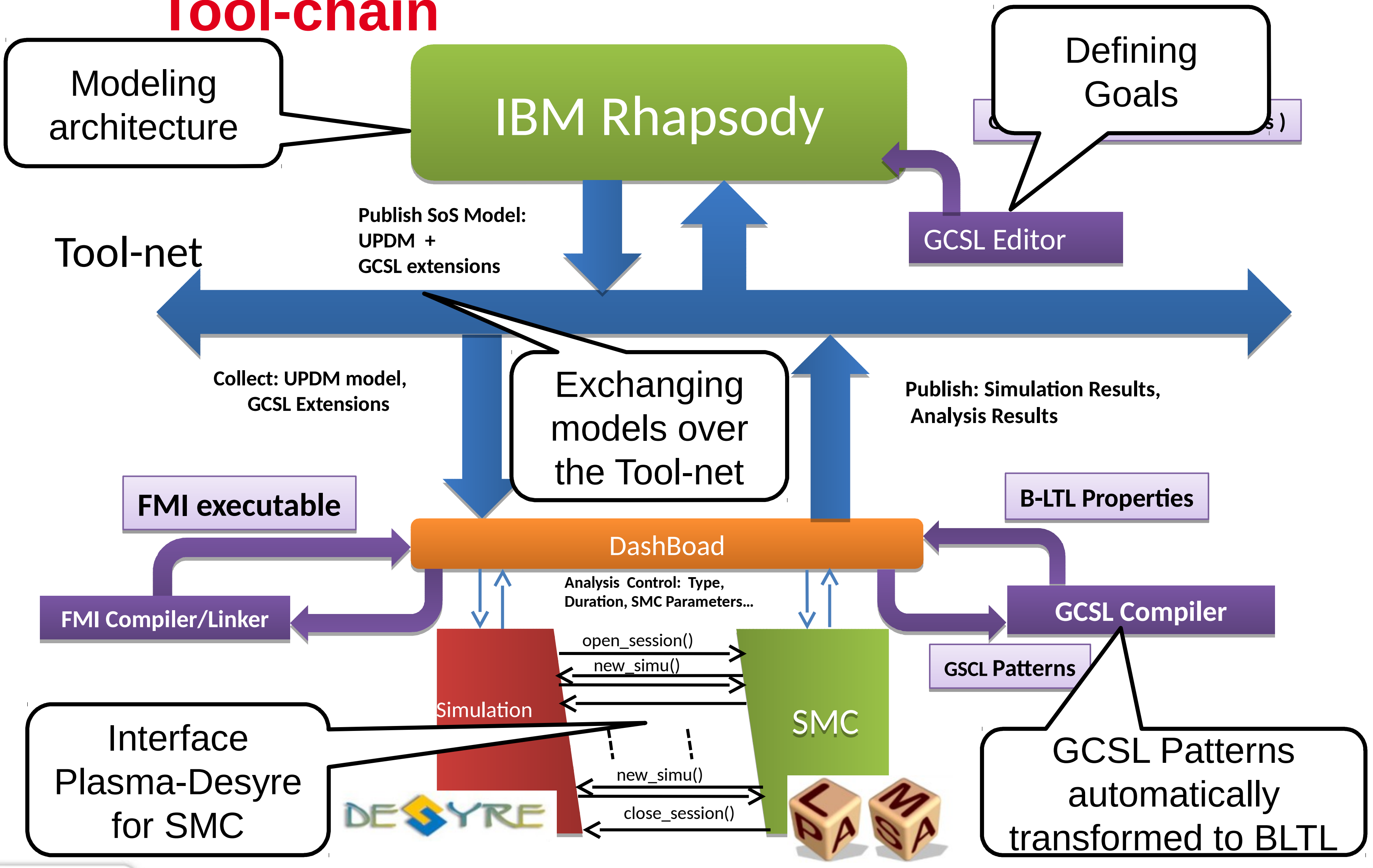
- Relies on joint simulation from DESYRE
  - Allows analysis of any model supported by DESYRE
  - Launches simulations and request new states as needed
- Checks a transformed version of the GCSL patterns
  - Contracts attached to the UPDM model
  - Automatic transformation to BLTL before an SMC session
- Using the PLASMA Statistical Model Checker

Tool-chain

From the modeling to the verification:

- Modeling of Constituent Systems: various tools (anything that exports FMU)
- Modeling of the architecture: **IBM Rhapsody**, enhanced with a SoS profile
- Defining goals: **dedicated GCSL editor**. Goals are attached to the architecture
- **Tool-net:** network allowing exchange of models, patterns and results
- **Dashboard**:
  - Load models from the tool-net
  - Parameterize and launch simulations
  - Parameterize and launch SMC analyzes
- Simulation handled by **DESYRE**
  - Loading of architecture from the tool-net
  - Loading of FMUs from the tool-net
- SMC handled by **PLASMA-LAB**
  - GCSL automatically converted to BLTL
  - Interface with DESYRE to control the simulation step-by-step

Tool-chain


Modeling architecture
IBM Rhapsody
Defining Goals
GCSL Editor
Tool-net
Publish SoS Model: UPDM + GCSL extensions
Collect: UPDM model, GCSL Extensions
Exchanging models over the Tool-net
Publish: Simulation Results, Analysis Results
FMI executable
B-LTL Properties
DashBoad
FMI Compiler/Linker
GCSL Compiler
Analysis Control: Type, Duration, SMC Parameters...
open_session()
new_simu()
Simulation
SMC
GSCL Patterns
Interface Plasma-Desyre for SMC
new_simu()
close_session()
DESYRE
PLASMA
GCSL Patterns automatically transformed to BLTL

5

Case Study

Emergency Response System

Emergency Response System

Models the reaction of several emergency systems:

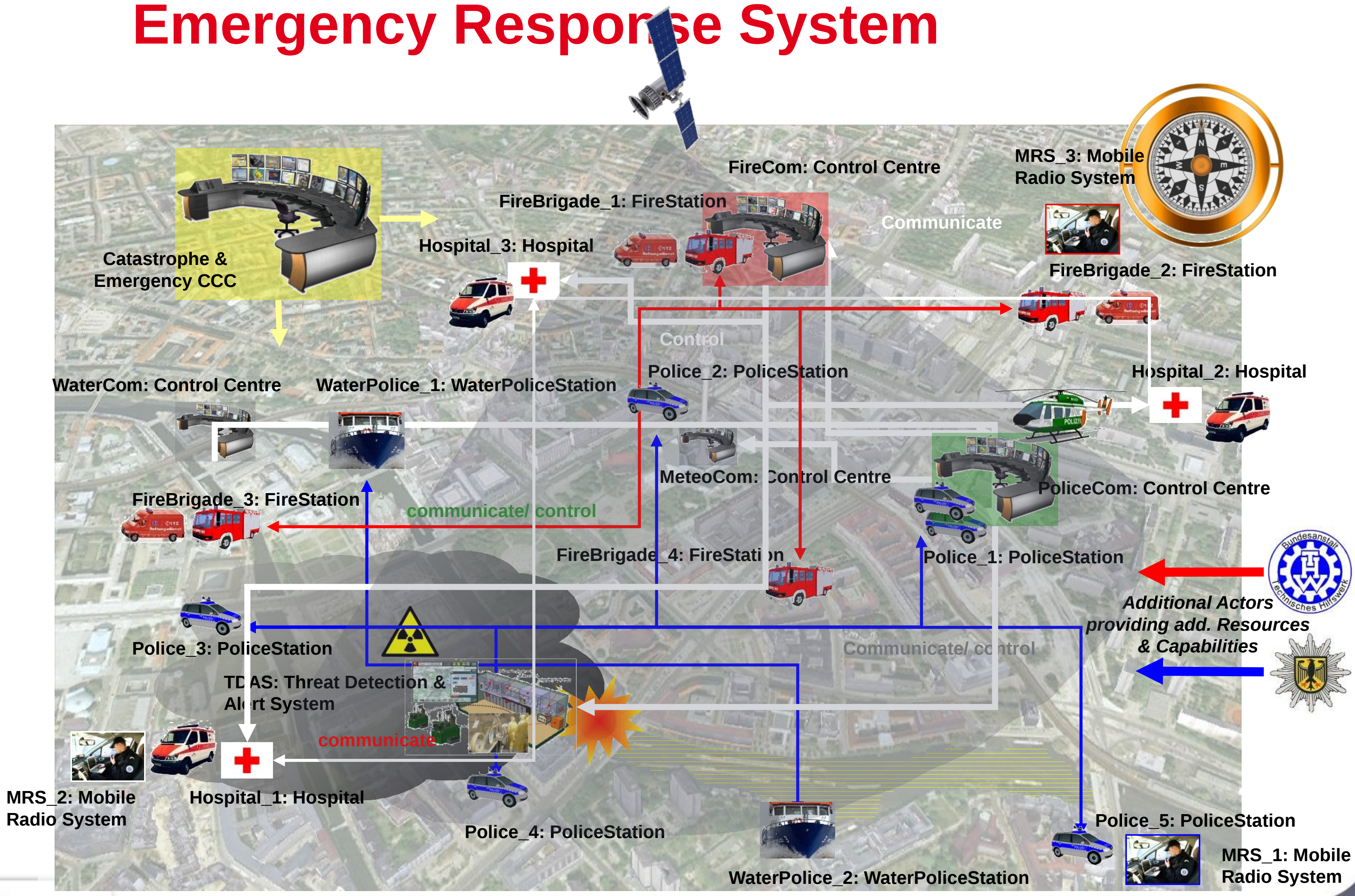
- Police
- Firefighters
- Hospitals

to a catastrophic event.

In particular communication protocols and communication channels are modelled.

The emergent behavior of the SoS should be an appropriate response.

Emergency Response System

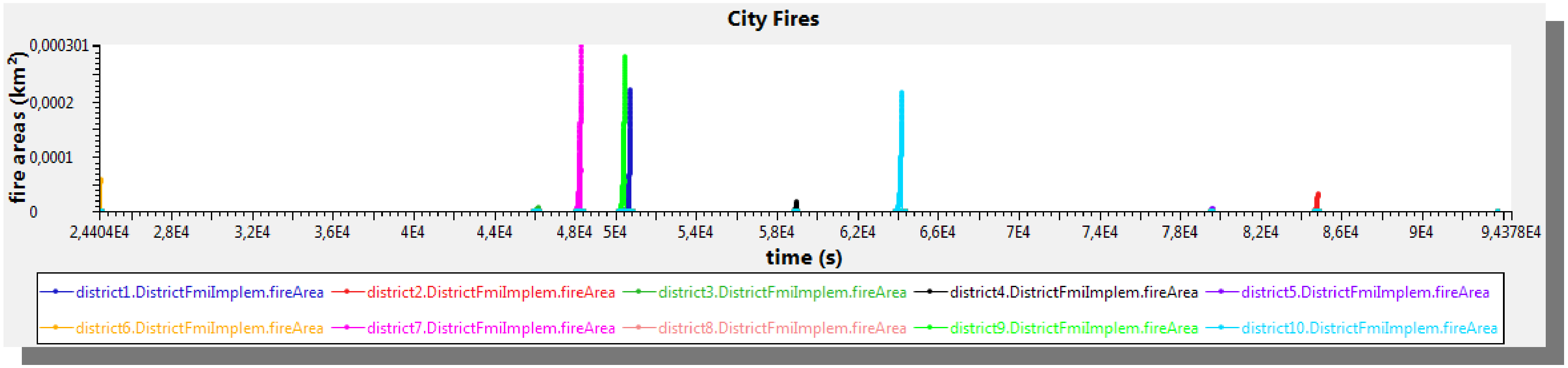

FireCom: Control Centre
MRS_3: Mobile Radio System
FireBrigade_1: FireStation
Communicate
Hospital_3: Hospital
Catastrophe & Emergency CCC
FireBrigade_2: FireStation
Control
Police_2: PoliceStation
Hospital_2: Hospital
WaterCom: Control Centre
WaterPolice_1: WaterPoliceStation
MeteoCom: Control Centre
PoliceCom: Control Centre
FireBrigade_3: FireStation
communicate/ control
FireBrigade_4: FireStation
Police_1: PoliceStation
Additional Actors providing add. Resources & Capabilities
Police_3: PoliceStation
Communicate/ control
TDAS: Threat Detection & Alert System
communicate
MRS_2: Mobile Radio System
Hospital_1: Hospital
Police_4: PoliceStation
Police_5: PoliceStation
MRS_1: Mobile Radio System
WaterPolice_2: WaterPoliceStation

Focus on a fire scenario

- Constituent systems (modelled in UPDM):

  Head Quarter, Stations, Cars, Firemen, Districts
- Whenever a fire occurs (determined probabilistically),
  - the districts sends a message to the Head Quarter,
  - the Head Quarter sends a message to the concerned Station,
  - the Station deploys Cars and Firemen

Simulation Output: Evolution of fire areas, for each district, during time



Evolution of the SoS requires a new analysis ...

Evaluating the probability of a fire

1: The fire is always smaller than X% of the total area:

$$\text{always } [\text{SoS.itsDistricts.fireArea} \rightarrow \text{sum}() > (X/100){*}\text{SoS.itsDistricts.area} \rightarrow \text{sum}()]$$

2: The fire is smaller than X% of the total area for 90% of the time

$$\text{at } [\ 10000\ ],\ [\text{SoS.itsDistricts.fireArea} \rightarrow \text{sum}() > (X/100){*}\text{SoS.itsDistricts.area} \rightarrow \text{sum}()] \text{ has been true at least } [\ 10\ ]\ \%\ \text{of time}$$

X	P(1)	Time
1	0.98	34 m
10^{-1}	0.95	39 m
10^{-2}	0.96	31 m
10^{-3}	0.93	36 m
10^{-4}	0.60	28 m
10^{-5}	0.35	25 m

SMC parameters:

Simulation time: 10000s

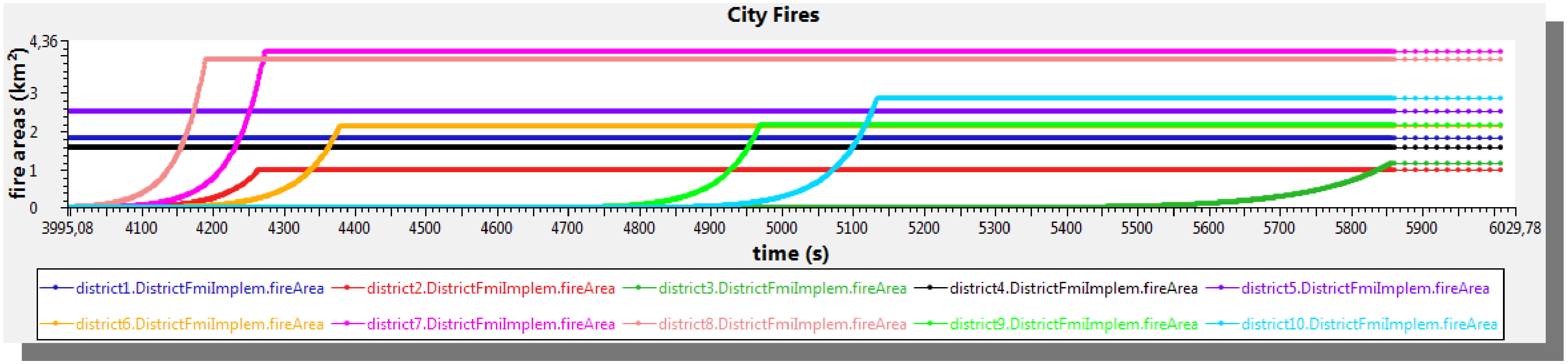
$\varepsilon = 0.1$
$\delta = 0.01$

X	P(2)	Time
1	0.95	40 m
10^{-1}	0.98	34 m
10^{-2}	0.96	43 m
10^{-3}	0.97	42 m
10^{-4}	0.97	42 m
10^{-5}	0.99	37 m

Next iteration

The probability that a fire lasts more than 10% of the time is too high.

This is due to a unwanted emergent behavior



Arises when two fires occur simultaneously.
Need to fix the architecture and reiterate the analysis.

Summary

- Modelling SoS
  - Reuse models of constituent systems
  - UPDM profile for SoS
  - Contracts

- Simulation
  - FMI/FMU based execution
  - Heterogeneous modelling langages

- Verification
  - Statistical Model Checking
  - Properties automatically obtained from contracts

Thank You

www.inria.fr