Towards Multi-Models for Self-* Cyber-Physical Systems

Hansen Salim and John Fitzgerald

15th Overture Workshop, Newcastle
15 September 2017

Agenda

Introduction

MetaSelf & INTO-CPS

Case study (UAV Swarm)

Conclusion

Newcastle University

Introduction

Self-* Systems

- self-adapting, organising, healing, optimising, …
- System improvement through a form of autonomy

Cyber-Physical Systems

- Interaction between cyber parts and physical entities.

E.g. Smart cars, UAVs

Self-Driving Vehicles[1]

[1] *http://www.documentarytube.com/articles/self-driving-cars-when-we-will-have-them*

Newcastle University

Self-* Cyber-Physical Systems

Design challenges

- Cyber & Physical, Heterogenous, Level of Autonomy, Real time
- Transferability between different CPSs

Use existing self-* framework for CPSs?

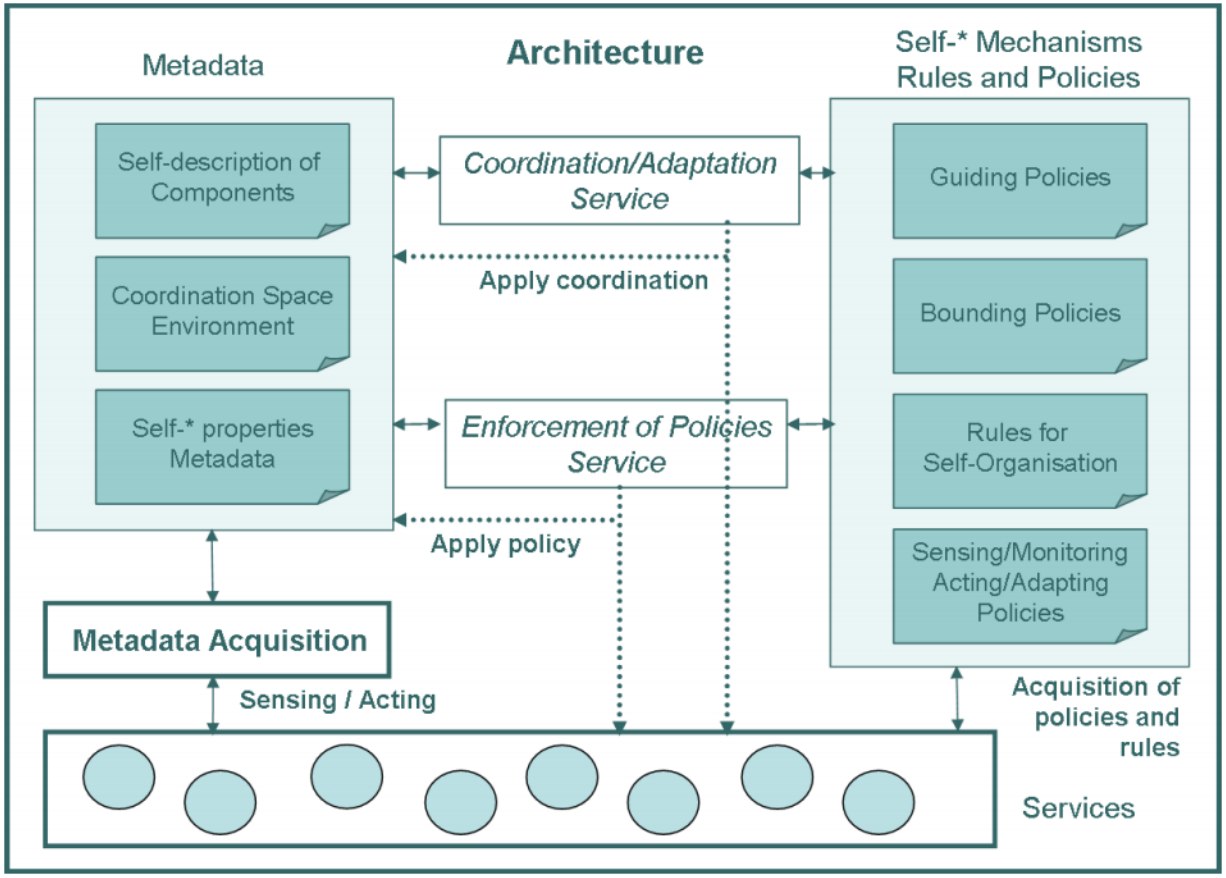
- MetaSelf: Service Oriented Architecture (SOA) - Component/Agent based, Middleware framework

CPS Model-Based Design support Self-*?

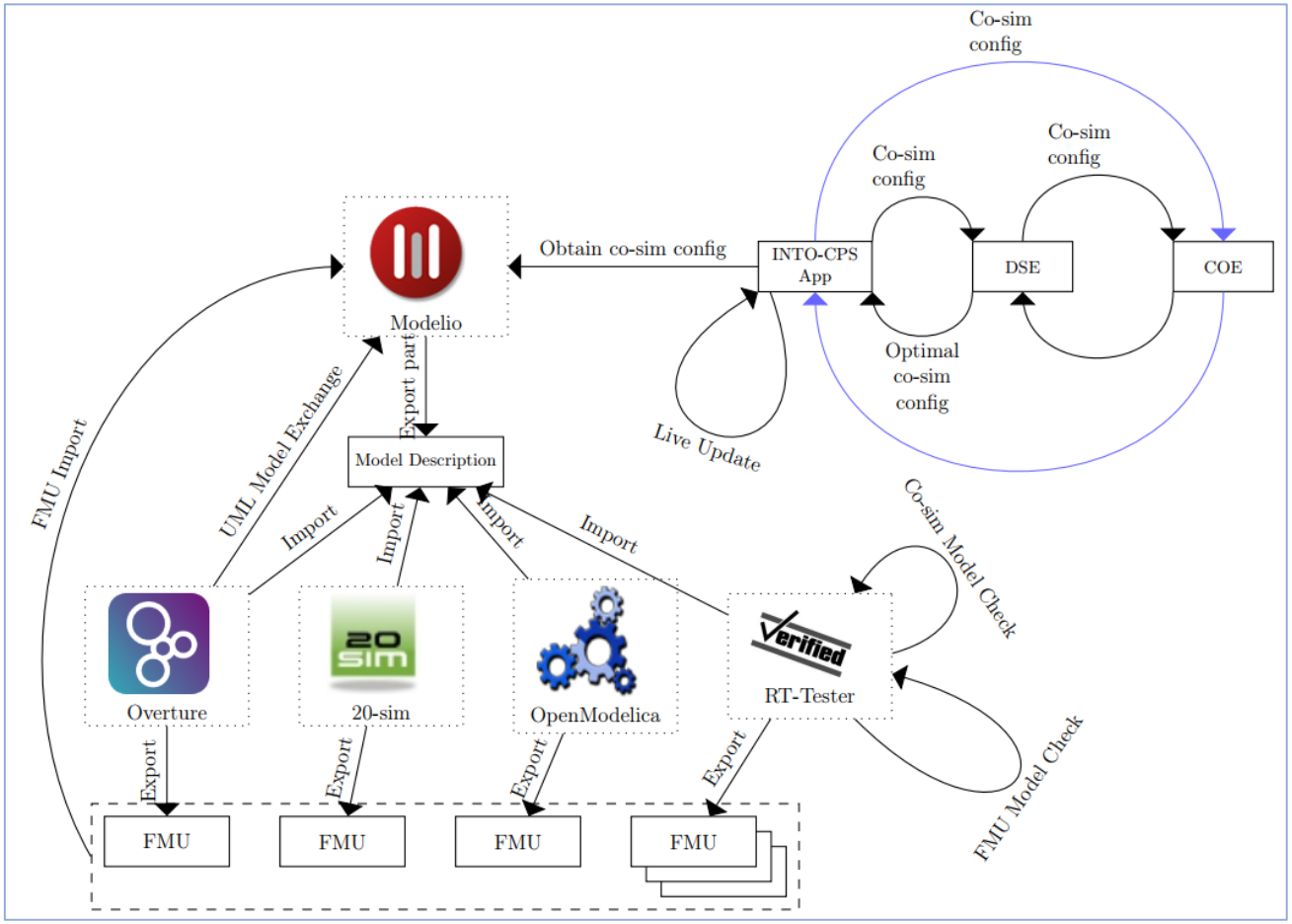
- INTO-CPS: multi model approach

MetaSelf

- Architectural framework, basis for assurance in some self-* capabilities
- Follow SOA approach, not designed for CPSs



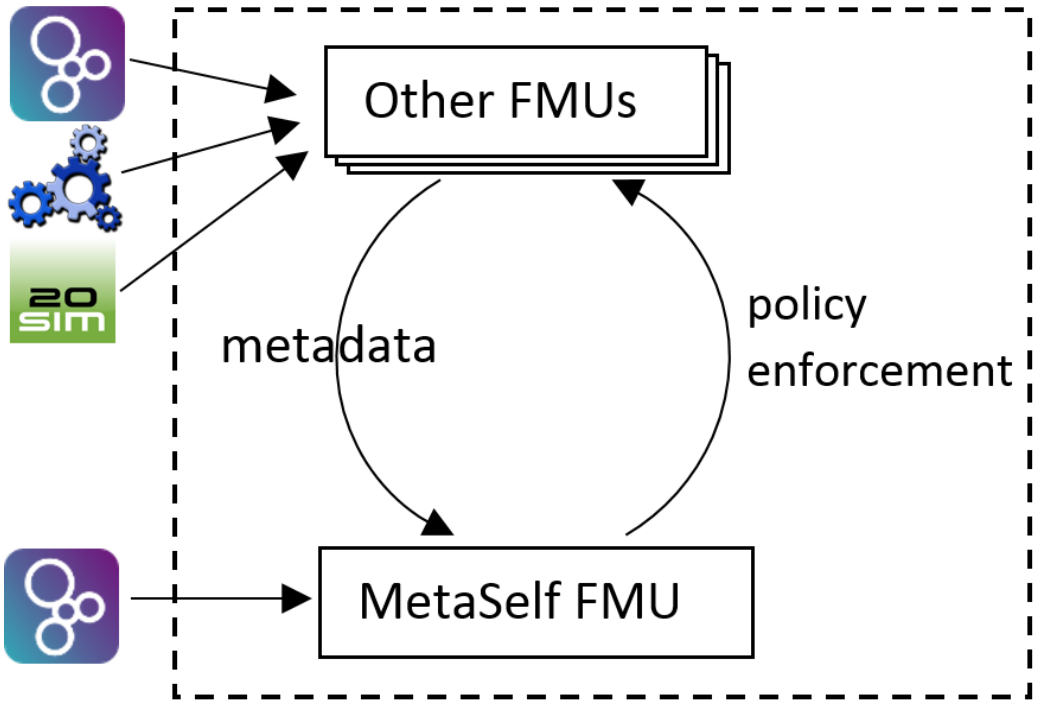
Newcastle University



Overview of the structure of the INTO-CPS tool chain

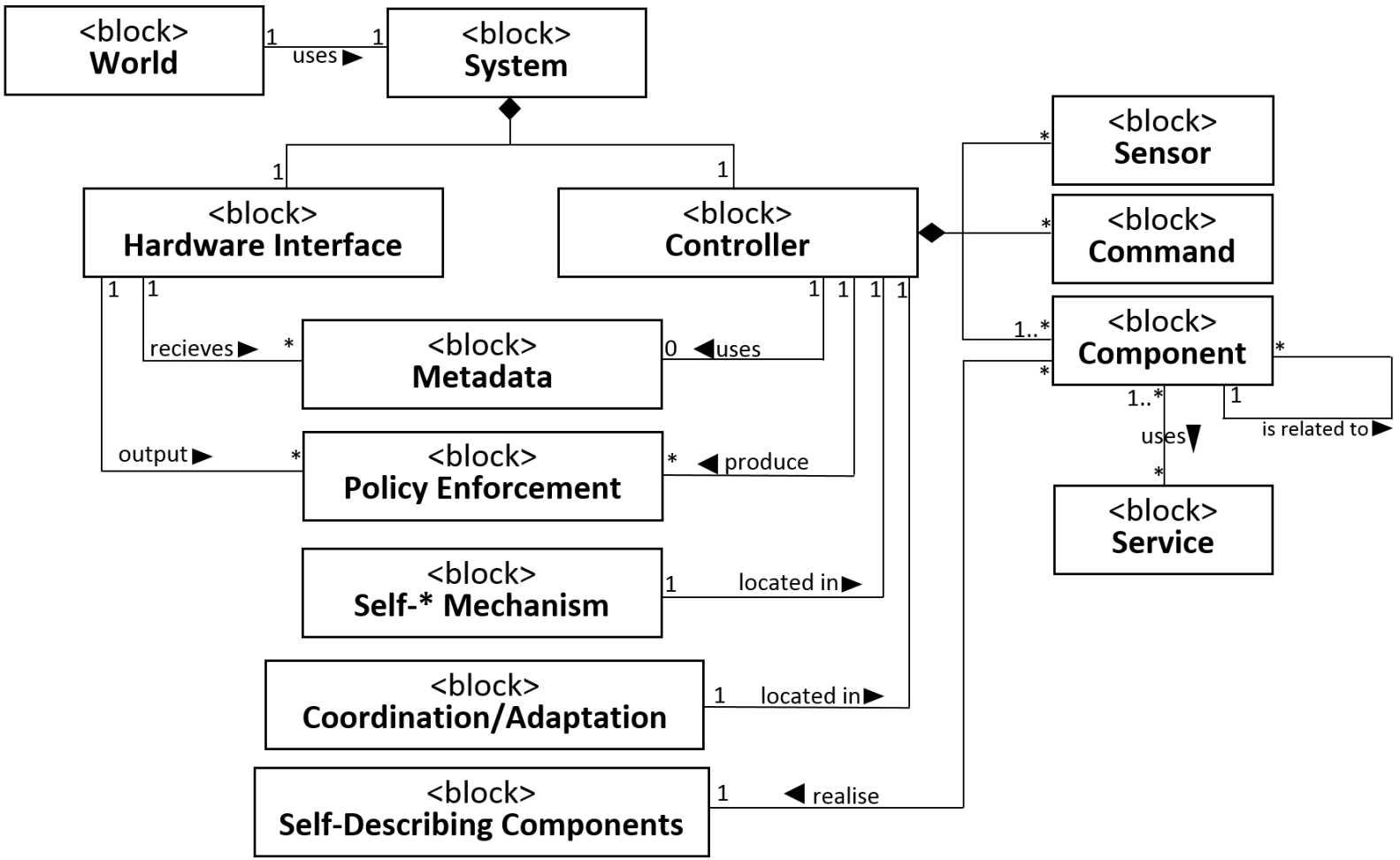
Our approach

- Standalone MetaSelf FMU
- DE side – VDMRT files



Interaction between MetaSelf FMU with other FMUs in INTO-CPS multi-model

Newcastle University



MetaSelf FMU (SysML block definition diagram)

Newcastle University

```
-- POLICY
public Step : () ==> ()
Step() ==
(
  dependabilityPolicies();
  SOrules();
);

public dependabilityPolicies: () ==> ()
dependabilityPolicies() ==
(
  --Define the dependability policies here
  --E.g if components.metadata = faulty() then (replaceComponent();moveToRegistry();)
  return
);

public moveToRegistry : token ==> ()
moveToRegistry(t) ==
(
  registry := registry ++ {t |-> listOfComponents(t)};
  listOfComponents := {t} <-: listOfComponents;
);

public moveToComponentList : token ==> ()
moveToComponentList(t) ==
(
  listOfComponents := listOfComponents ++ {t |-> registry(t)};
  registry := {t} <-: registry;
);

public SOrules: () ==> ()
SOrules() == (
  --SOrules typically applies to all components and dictates its global behaviour
  --E.g if components.metadata = faulty() then replaceComponent()
  return
);
```

Controller.vdmrt
Internal Policy for self-* mechanism

VDMRT classes defined following the MetaSelf framework

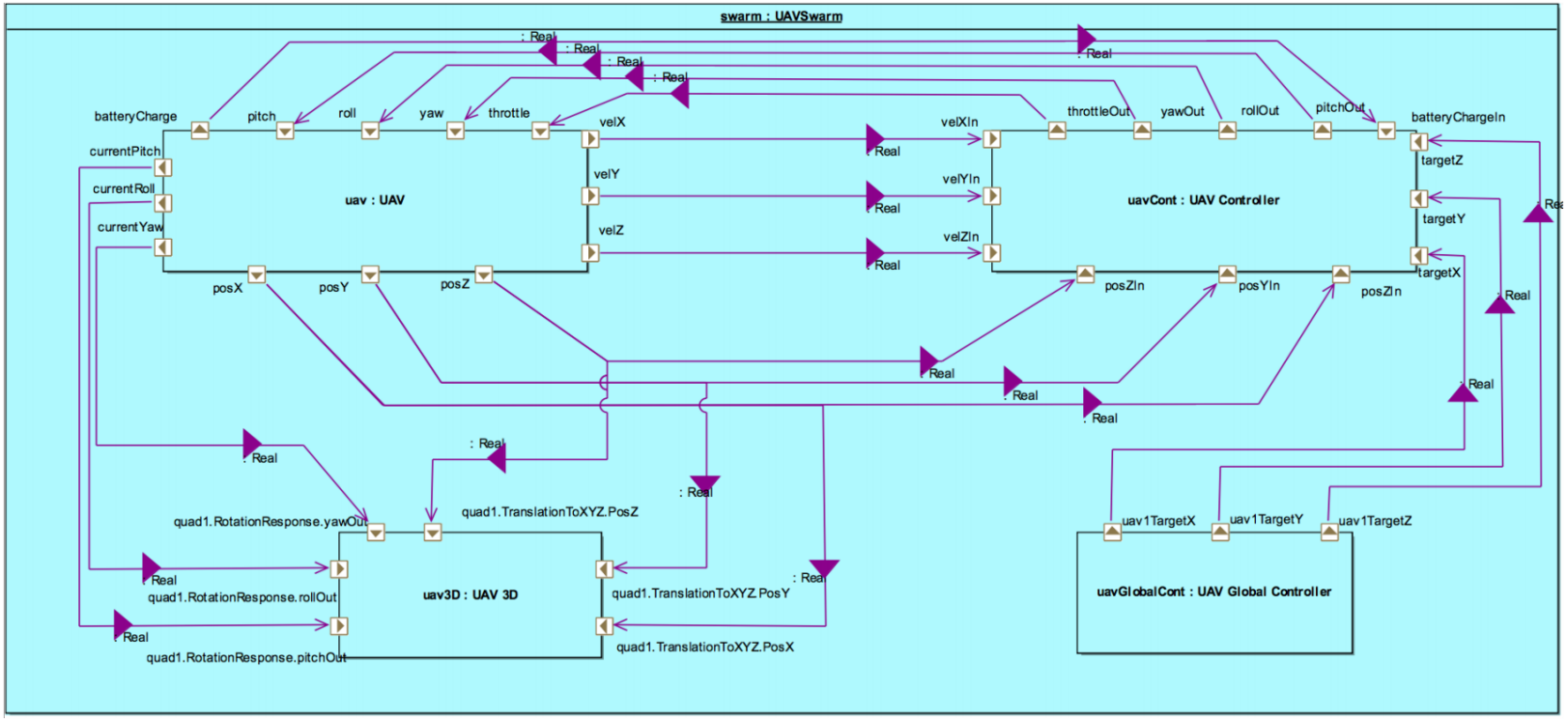
```
class Component

operations
public getMetadata : token ==> real
getMetadata(t) == is subclass responsibility;

public enforcePolicies : Service ==> ()
enforcePolicies(c) == is subclass responsibility;

end Component
```

Component.vdmrt
Generic class for system components

Newcastle University

Case study

INTO-CPS
(D3.5 - Examples Compendium 2)
“Swarm of UAV”



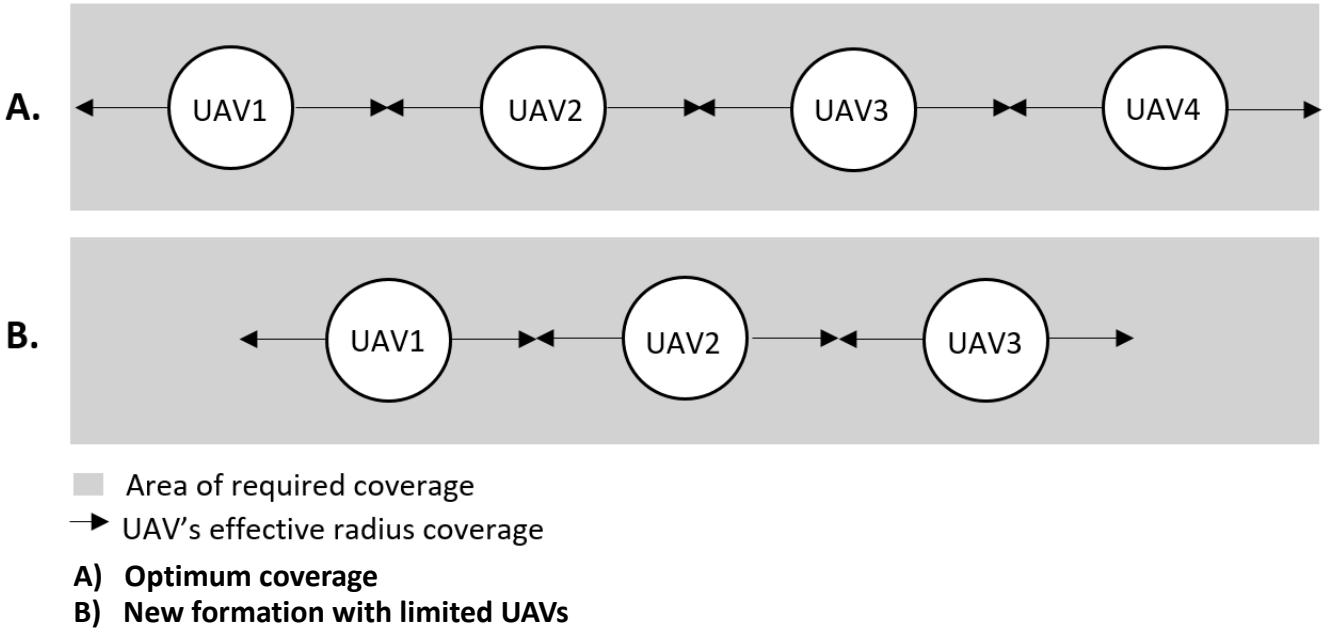
“Swarm of UAV” SysML Connection Diagram

Newcastle University

Self-Organising UAV Swarm

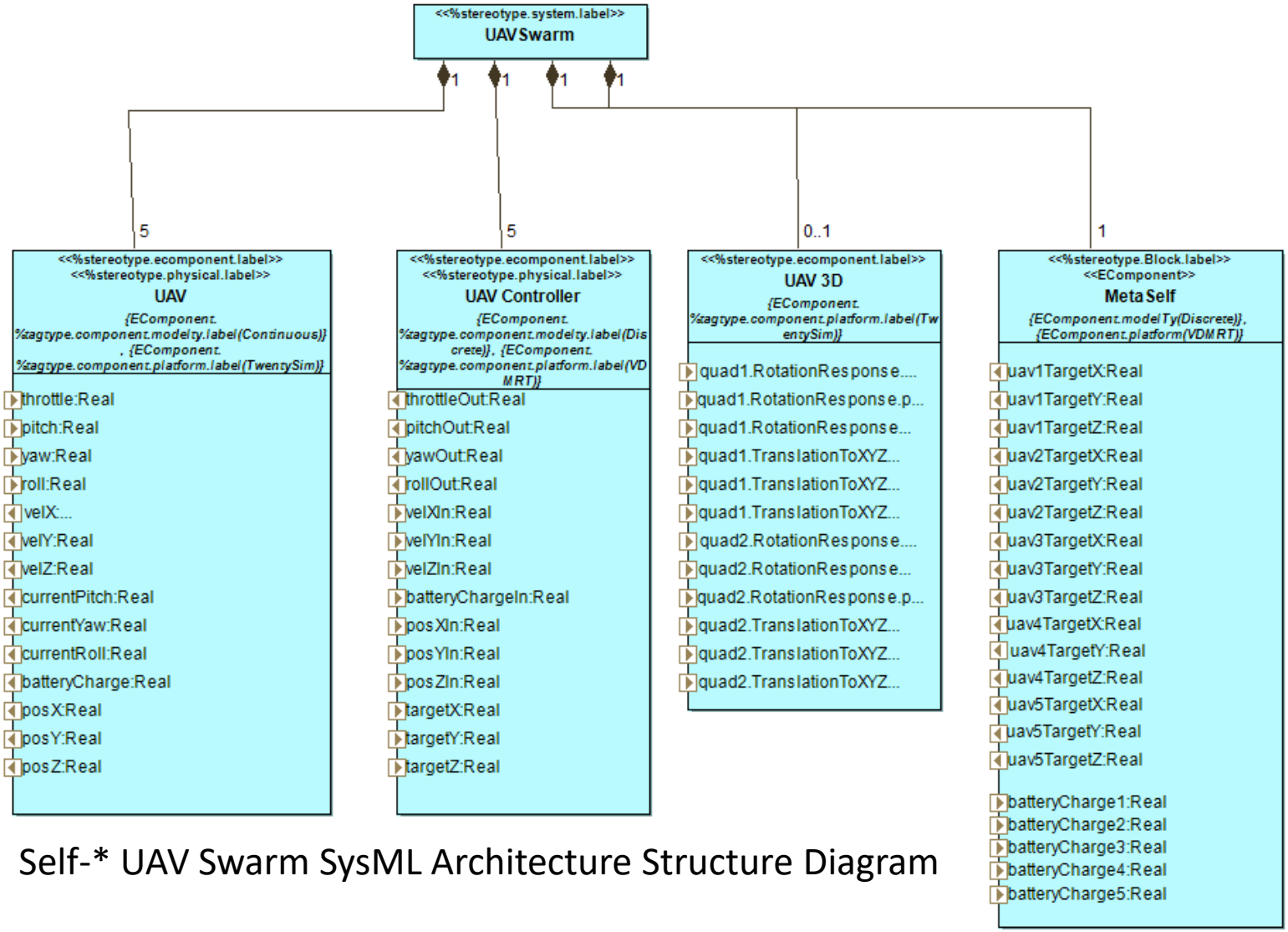
System goal: Maximize UAV coverage over a designated area (1 directional plane)

Self-* goal: Perform UAV replacement or reconfigure formation



A) Optimum coverage
B) New formation with limited UAVs

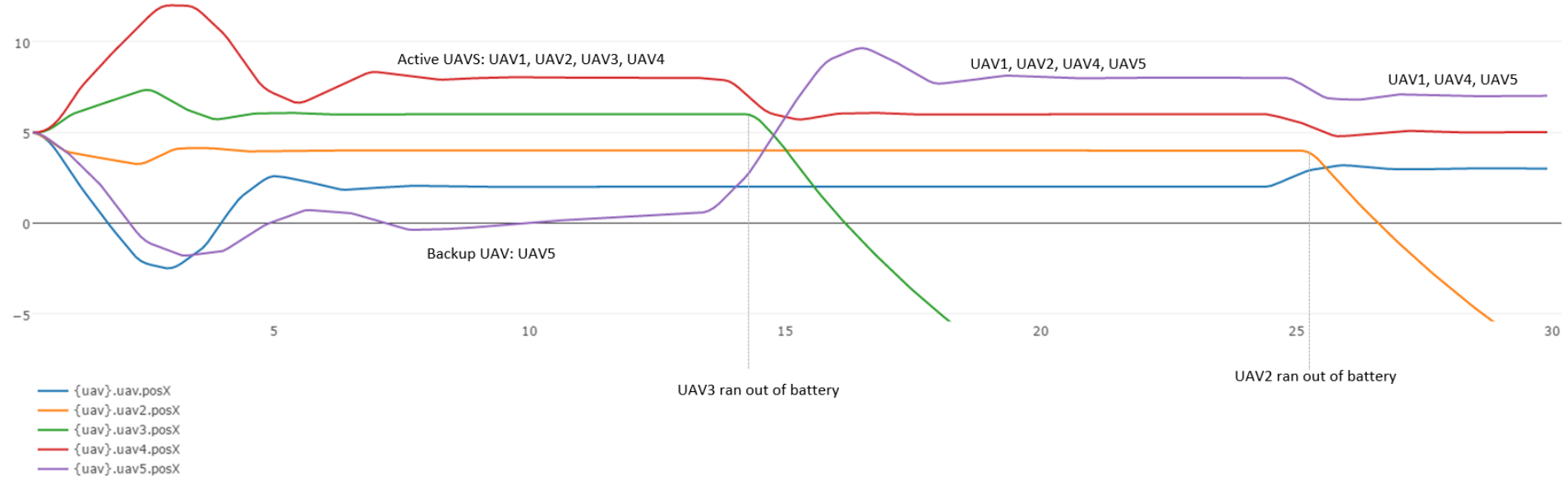
Newcastle University



Self-* UAV Swarm SysML Architecture Structure Diagram

*Might require multiple instances of the MetaSelf FMU for decentralised CPSs

Newcastle University



UAV swarm case study simulation graph – X coordinate of UAVs

Newcastle University

Conclusion

MetaSelf for CPSs

- Cyber & Physical integration handled by INTO-CPS
- Real time - no support for timely policy execution
- Autonomy - Extension required for machine learning

INTO-CPS for Self-*

- SysML profile does not describe self-* properties:
  - Architecture diagram – Architectural reconfiguration
  - Connection diagram – Dynamic connection between FMU components

Further work: case study on decentralised CPS with heterogenous components & support for real time