FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

System-Level Design Under Confidentiality and Integrity Constraints

Janos Sztipanovits
Istvan Madari
Tamas Kecskes
ISIS-Vanderbilt

Berkeley
UNIVERSITY OF CALIFORNIA

Massachusetts Institute of Technology

NSF

MICHIGAN

VANDERBILT UNIVERSITY

theory and methods

without tools

remain speculations

Content

1. **Background**
   - Goals
   - Theory
   - Validation Agenda
2. Integrated Tool Suite
   - Functional Architecture
   - WebGME-FORMULA Integration
   - Deployment Architecture

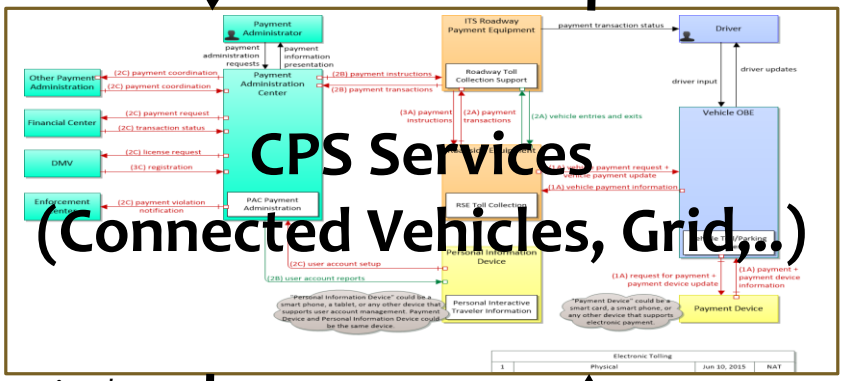
FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Notional Architecture

Human Users/Regulators

Functionalities/
Policies

Utilities



CPS Services
(Connected Vehicles, Grid,..)

Mapping/
Configuration

Services

Physical Systems/Resources

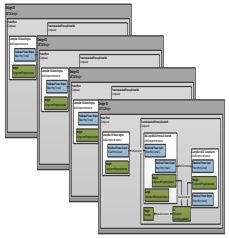
FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

What Is Our Goal?

* **The system-level synthesis problem for information flows in CPS:**
  - Derive specification for the behavior of the system components that will be implemented using networked computing
  - Derive a functional model for the information architecture and componentize the system
  - Select computing/networking platform
  - Derive deployment model assigning components of the information architecture to processing and communication platforms
  - Generate code for software components and derive WCET and WCCT
  - Perform timing analysis
* **Making security part of system-level co-design (correct-by-construction)**
  - Co-design of functionality, performance, timing and security
  - Our goal is to address security requirements as part of the design trades embedded in the system-level design process

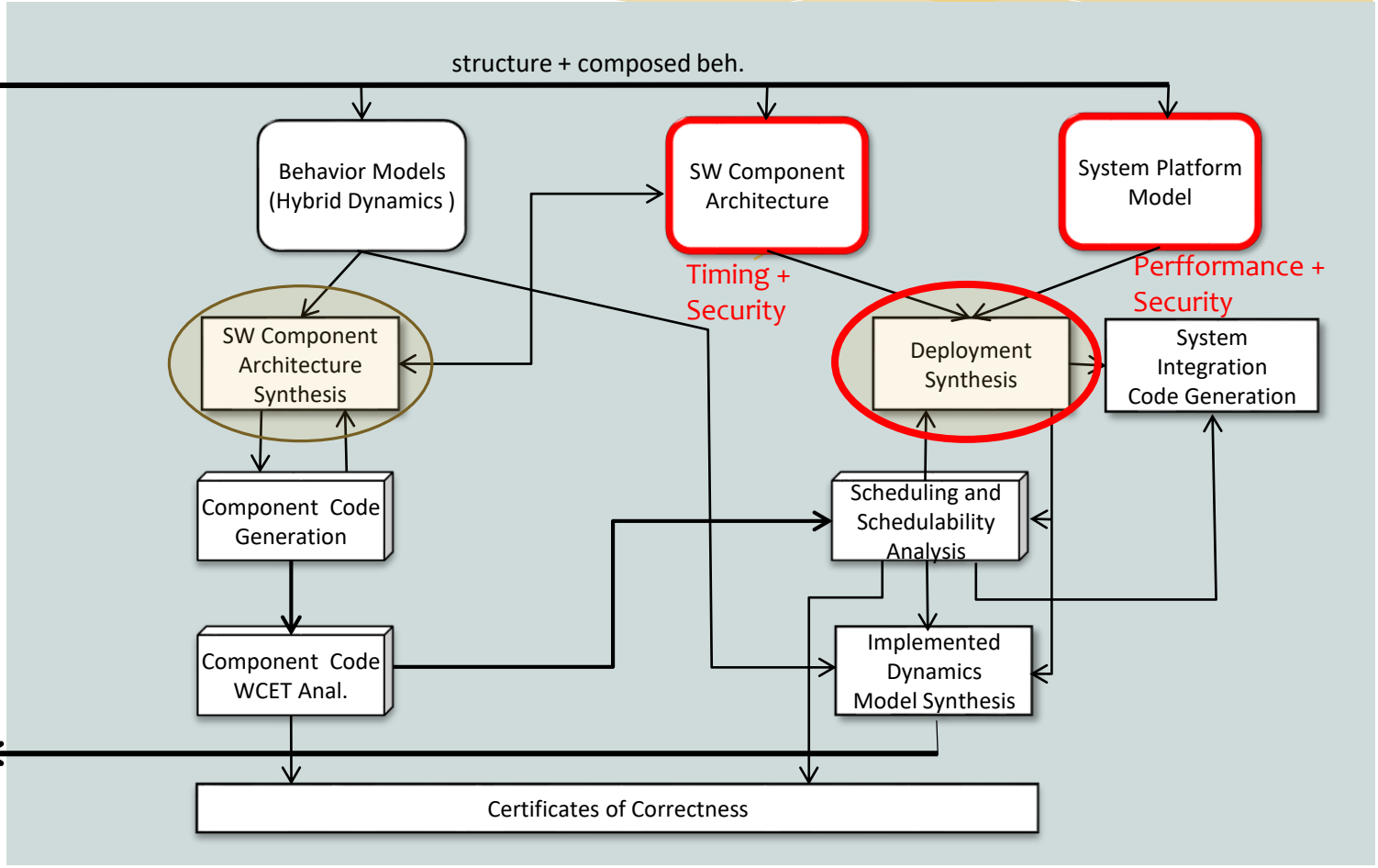
FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

System-level Synthesis Under Timing, Security and Confidentiality Constraints



Design Architectures with ideal comp. dynamics

Design Architectures with **deployed comp. dynamics**



- CAN Bus
- TT bus

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Synthesis Problem

* How to map a logical Information Architecture (components + information flows) on a physical Platform Architecture such that
  - Functional requirements (the information architecture)
  - Performance requirements (timing)
  - Security requirements (confidentiality and integrity)

  are satisfied simultaneously?

Agenda

* Modeling language suite ✓
  (behavior, information flows, SW components, architecture, timing, platform, deployment) - reuse previous work as example
* Security Requirement Modeling ✓
  (need to be composable with other modeling aspects)
* Common Semantic Domain and Formal Framework ✓
  (functional, performance and security models need to be anchored to a semantic domain suitable for synthesis)
* Synthesis Framework and Co-design flow ✓
  (mapping system-level synthesis problem on the formal framework and tools)
* **Integrated Tool Suite and Validation**
  (target domain rich enough for testing the co-design tool suite)

Content

1. Background
   - Goals
   - **Theory**
   - Validation Agenda
2. Integrated Tool Suite
   - Functional Architecture
   - WebGME-FORMULA Integration
   - Deployment Architecture

Security Concerns Addressed

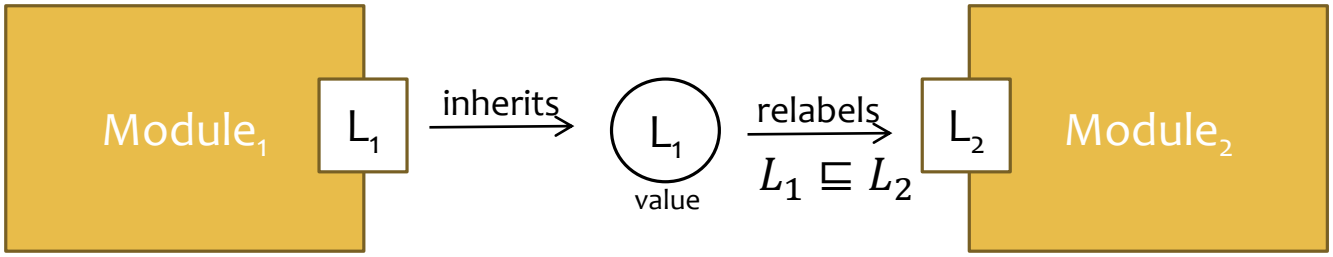
* **Integrity attacks**
  - Manipulate data (value, timestamp, source identity,..)
* **Confidentiality attack**
  - Leak critical data to unauthorized persons/systems
* **Integrity and confidentiality restrictions impose constraints on information flows.**
  - How to model these restrictions?
  - How to integrate these restrictions with others (functional and timing) and formulate a co-design problem?

Decentralized Label Model (DLM) for Information Flow Control

* Myers, Liskov (1997): Introduced **security-typed languages** by labeling variables with information flow security policies
* Method was developed for programming languages, the result is *Jif, a security-typed version of Java.*
* DLM provides mechanism for static/dynamic type checking of security labels in information flows to detect policy violations.
* Example: *Jif,* a security-typed version of Java
* **Introduce security-types in modeling languages**

Security Type Propagation Rules

* *Propagation rule-1 (restriction):*

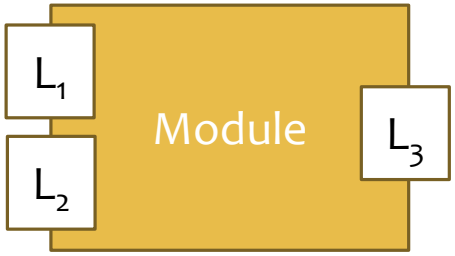


$owners(L_1) \subseteq owners(L_2)$

$\forall o \in owners(L_1), readers(L_1, o) \supseteq readers(L_2, o)$

(L_1 has more readers and fewer owners than L_2)

* *Propagation rule-2 (join):*



L_3 is the *join* of L_1 and L_2

$L_3 = L_1 \sqcup L_2$

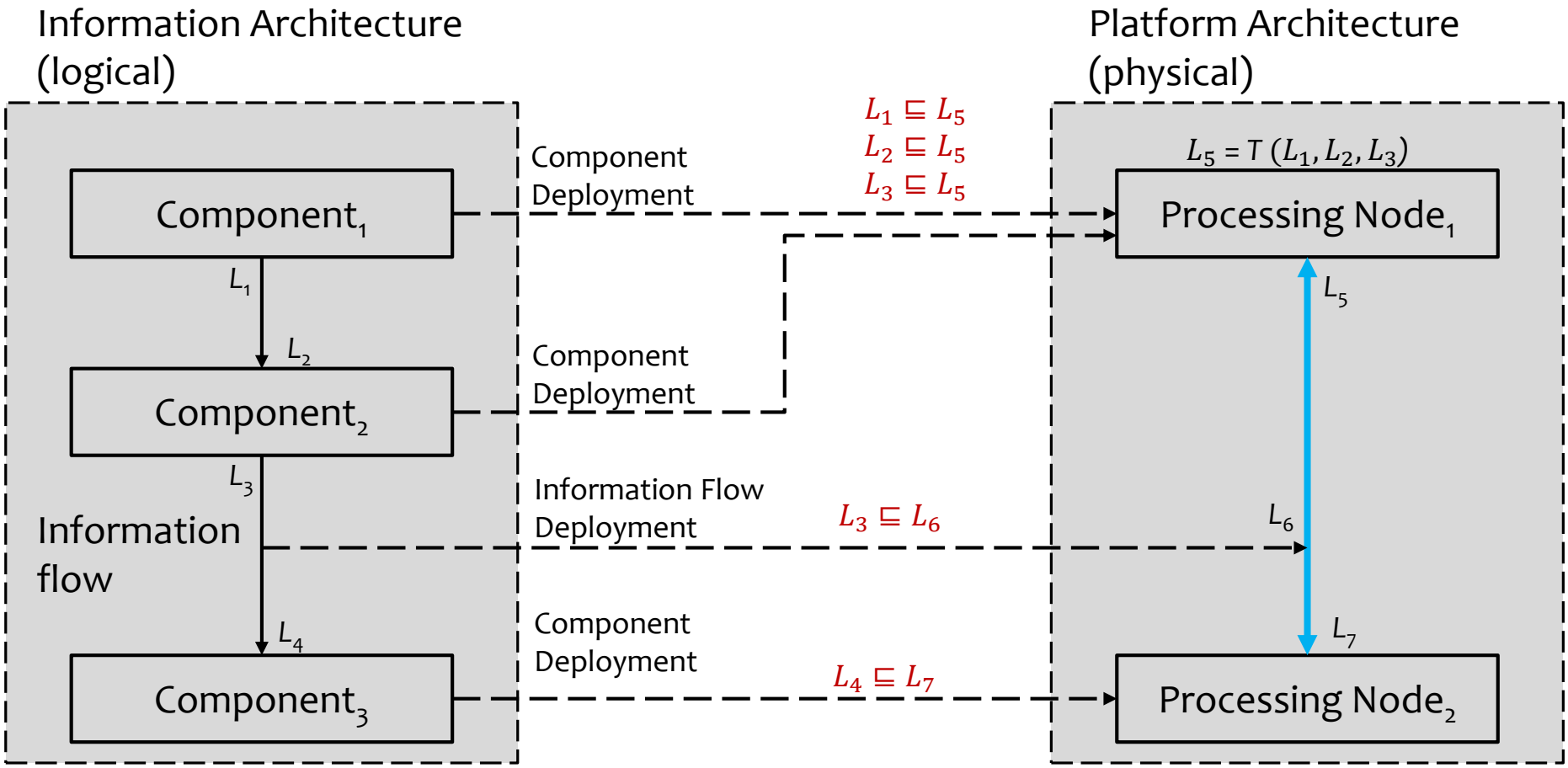
$owners(L_1 \sqcup L_2) = owners(L_1) \cup owners(L_2)$

$readers(L_1 \sqcup L_2, o) = readers(L_1, o) \cap readers(L_2, o)$

(*join* L_1 and L_2 is the least restrictive label that maintains all the flow restrictions specified by L_1 and L_2)

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Information Architecture Deployed on a Physical Platform



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Policies and Labels in FORMULA

A policy consists of an owner principal and a set of allowed reader principals:

owner: reader1 reader2

A label is a (possibly empty) set of policies:

L = { policy1; policy2; ...}

Our encoding views a label as a tree where the label's identifier is the root, the policy owners make up the second level, and the corresponding readers make up the third level :

```
Label  ::= new (name:String).
Policy ::= new (lbl:Label, owner:Principal).
Reader ::= new (pl:Policy, reader:Principal).
```

Propagation Rules Examples

We can compute the effective readers set for each label:

```
EffReader(lbl, reader) :-
  lbl is Label, reader is Principal, no CantRead(lbl, reader).
CantRead(pl.lbl, r) :-
  pl is Policy, r is Principal,
  no { r' | ActsForTR(r, r'), Reader(pl, r') }.
```

We can compare the restrictiveness of labels based on their effective reader sets:

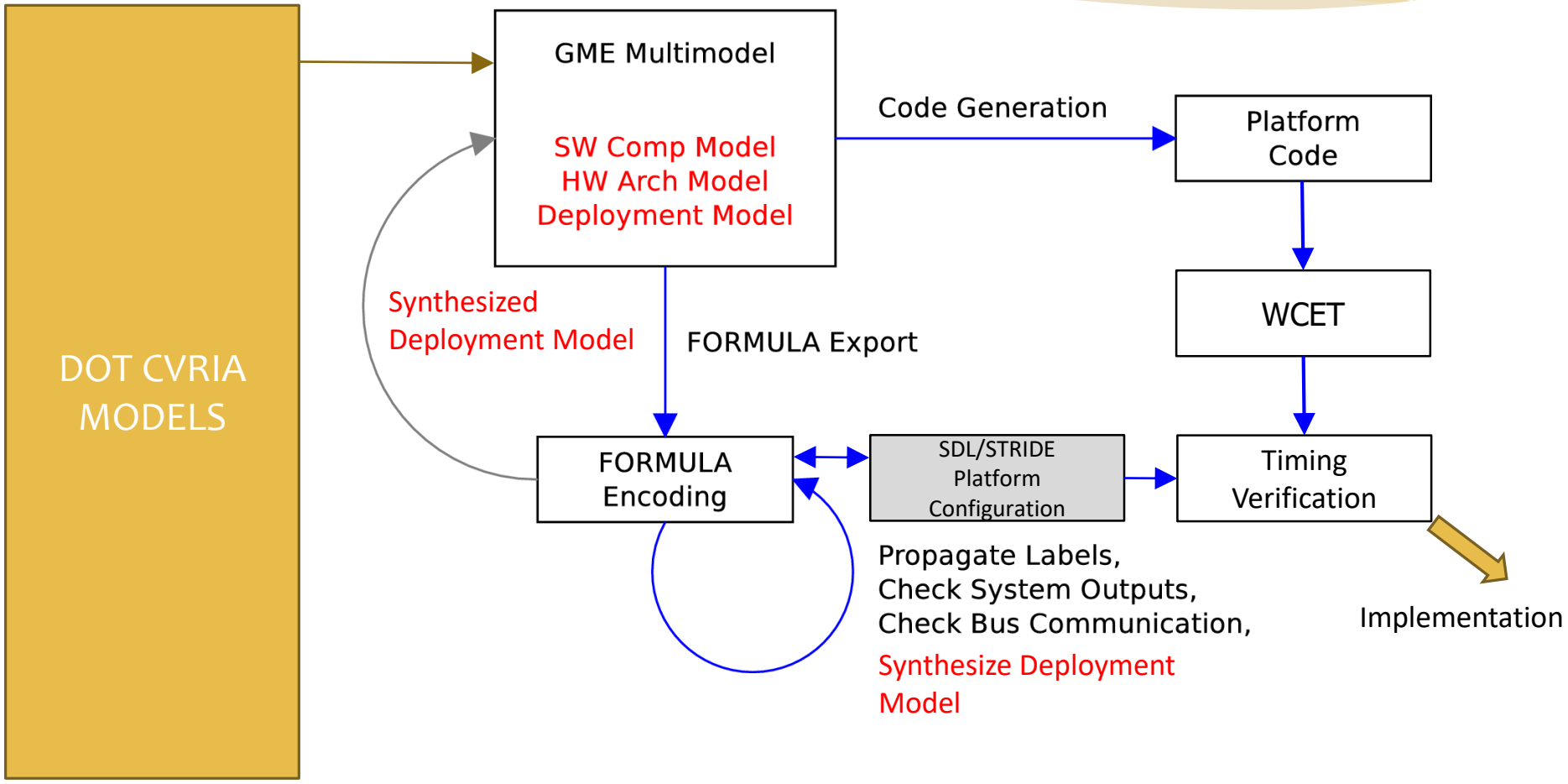
```
AtLeastAsRestrictive(lbl1, lbl2) :-
  lbl1 is Label, lbl2 is Label,
  no { x | EffReader(lbl1, x), CantRead(lbl2, x) }.
```

We can also “propagate” policies by computing the join ($\bigsqcup$) of two labels: the least restrictive label that is at least as restrictive as both labels.

Content

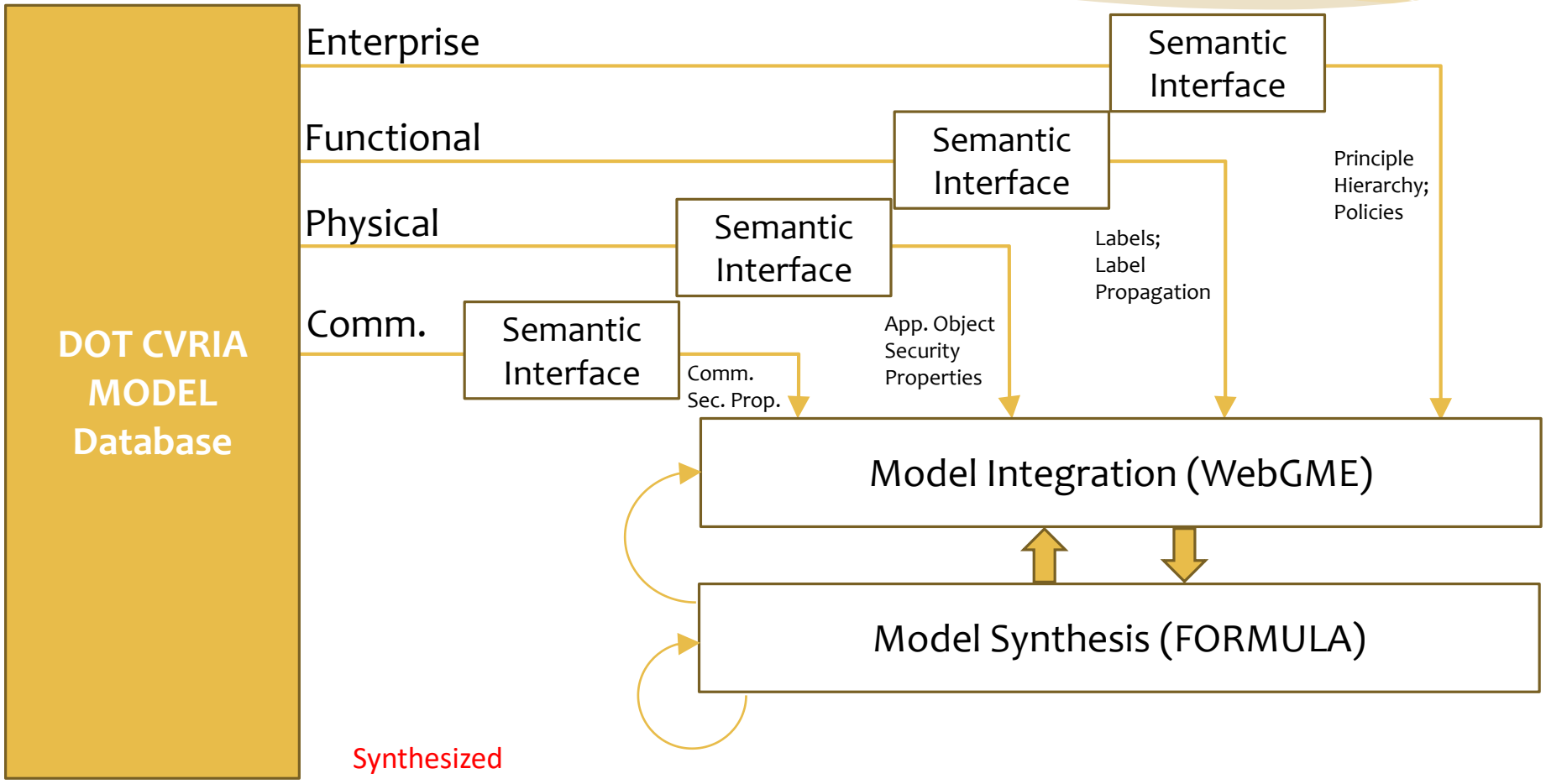
1. Background
   - Goals
   - Theory
   - **Validation Agenda**
2. Integrated Tool Suite
   - Functional Architecture
   - WebGME-FORMULA Integration
   - Deployment Architecture

Validation of the System-level Design Workflow



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Functional Architecture



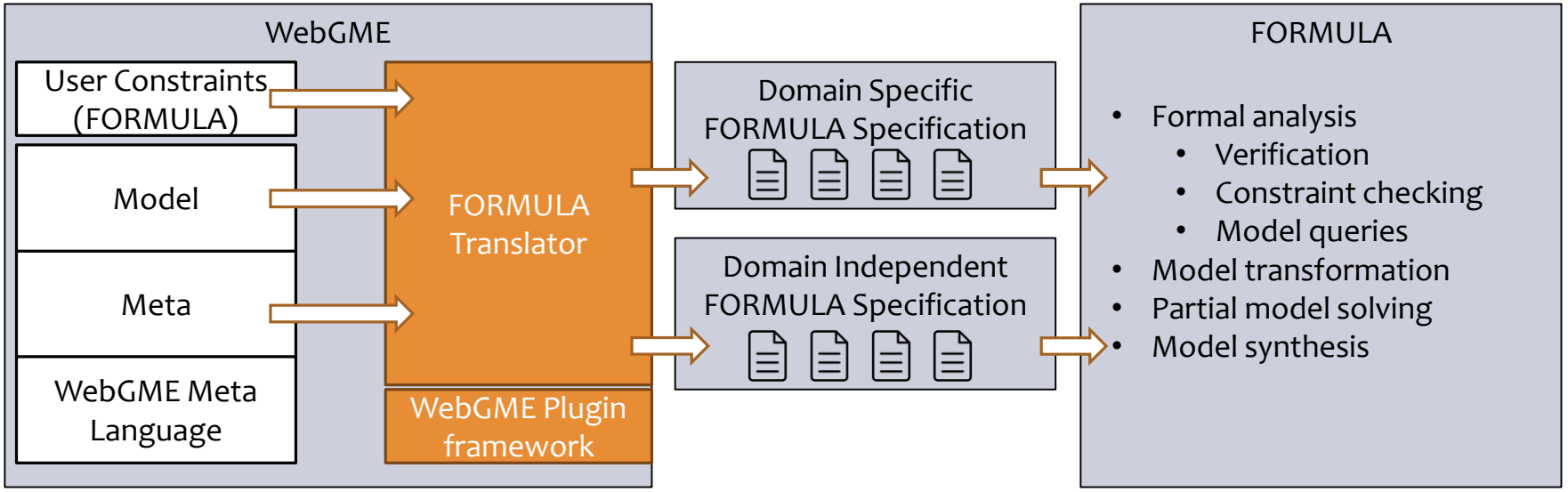
FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Content

1. Background
   - Goals
   - Theory
   - Validation Agenda
2. Integrated Tool Suite
   - **Functional Architecture**
   - WebGME-FORMULA Integration
   - Deployment Architecture

Approaches to Generate FORMULA Specifications from WebGME

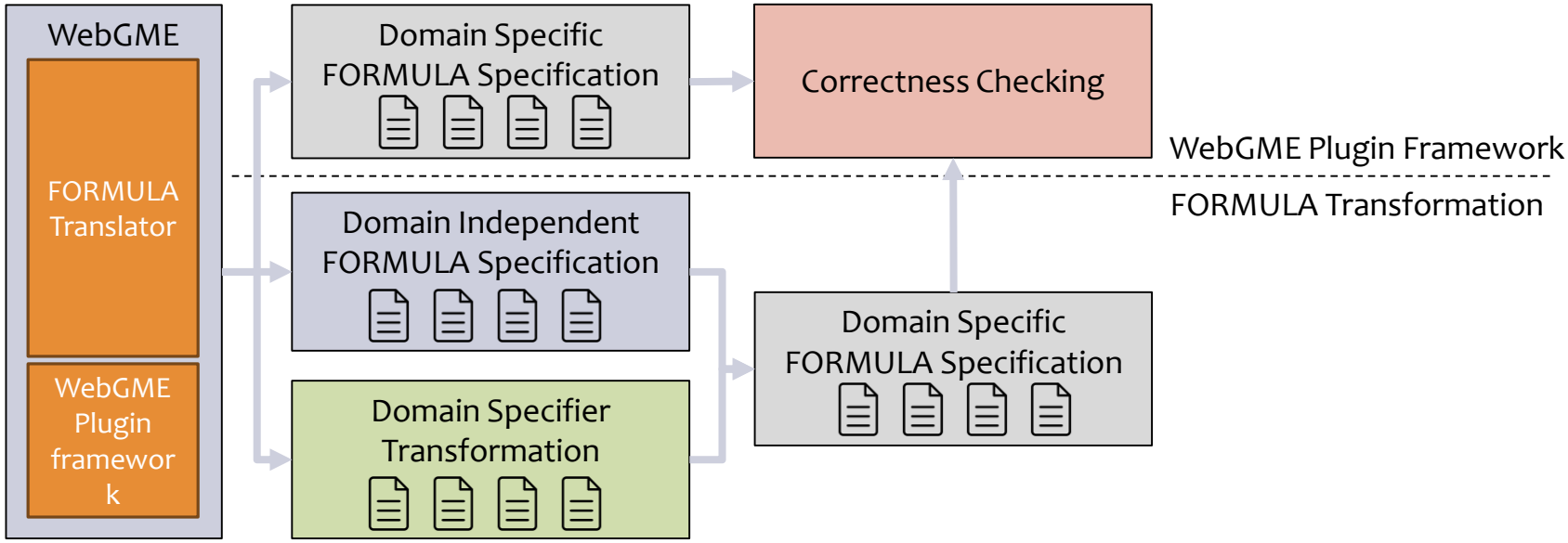
1. Domain Specific approach
2. Domain Independent approach



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

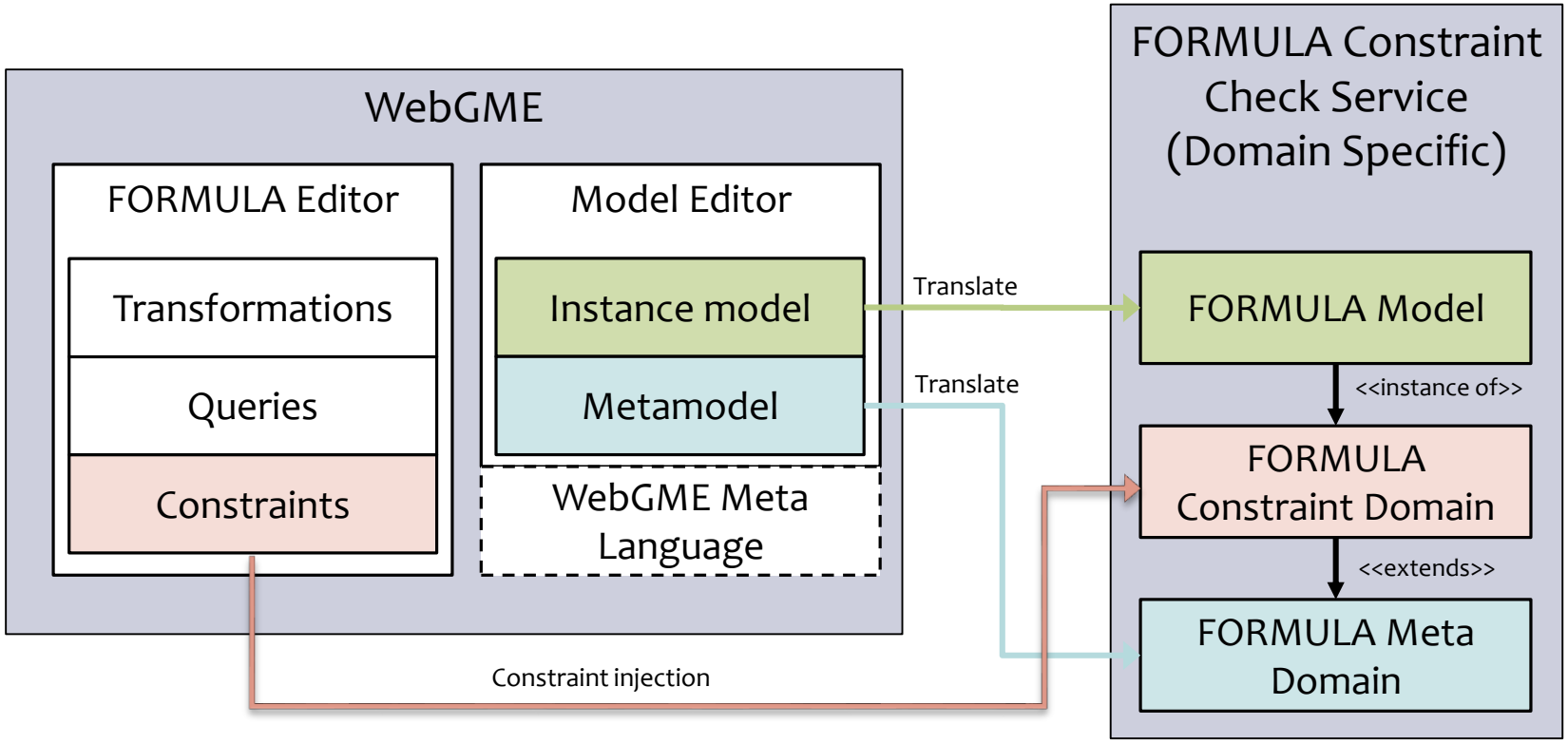
Utilize the Domain Independent Specification in Correctness Checking

- Domain specific FORMULA models can be generated from the domain independent specification
  - By FORMULA model transformation (Domain Specifier Transformation)
  - Can be slow process (One CVRIA application, with 839 nodes and 480 connections: ~145 sec)
- **OR** generate the domain specific representation by WebGME plugin framework (JavaScript)
  - If needed, compare the outputs of the plugin and the FORMULA transformation



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Translate WebGME Models Into FORMULA (Check Service)

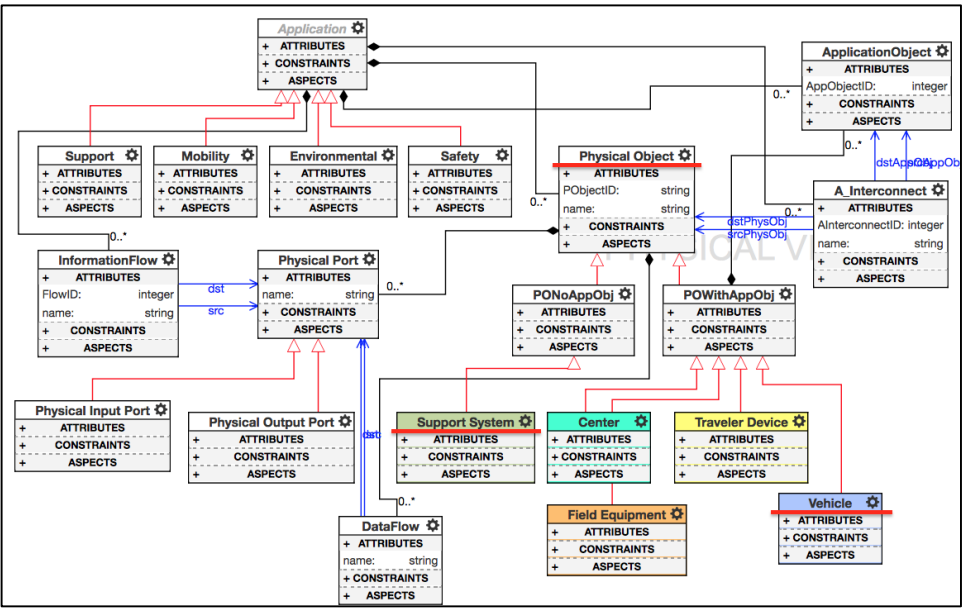


FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Content

1. Background
   - Goals
   - Theory
   - Validation Agenda
2. Integrated Tool Suite
   - Functional Architecture
   - **WebGME-FORMULA Integration**
   - Deployment Architecture

Results of WebGME -> FORMULA Translation (Metamodel)

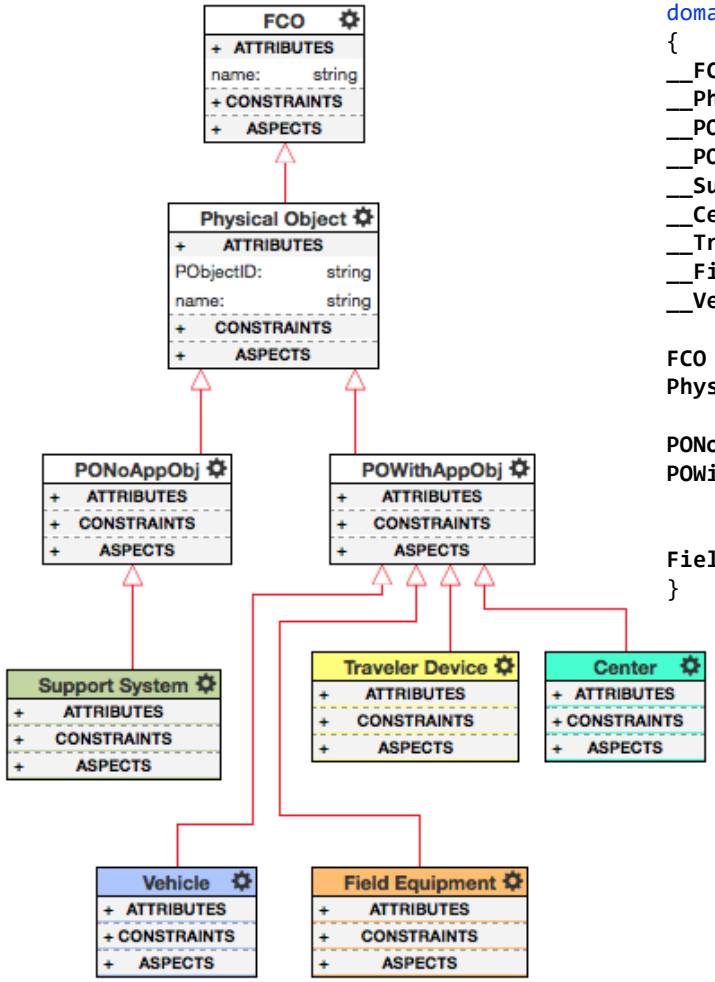



```
domain CVRIA
{
__Physical_Object ::= new (id:String,
                           base: any __Physical_Object + {NULL},
                           parent: any GMENode + {NULL},
                           attributes: Attr__Physical_Object + {NULL},
                           pointers: {NULL}).

__Support_System ::= new (id:String,
                          base: any __Support_System + {NULL},
                          parent: any GMENode + {NULL},
                          attributes: Attr__Support_System + {NULL},
                          pointers: {NULL}).

__Vehicle ::= new (id:String,
                   base: any __Vehicle + {NULL},
                   parent: any GMENode + {NULL},
                   attributes: Attr__Vehicle + {NULL},
                   pointers: {NULL}).
// Attributes
Attr__Vehicle ::= new (name:String, PObjectID:String).
Attr__Physical_Object ::= new (name:String, PObjectID:String).
Attr__Support_System ::= new (name:String, PObjectID:String).

// Unions, represent inheritance
Physical_Object ::= PONoAppObj + POWithAppObj + __Physical_Object.
PONoAppObj ::= Support_System + __PONoAppObj.
POWithAppObj ::=Field_Equipment + Center + Traveler_Device + Vehicle +
__POWithAppObj.
Support_System ::= __Support_System.
Vehicle ::= __Vehicle.
}
```


FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Represent Inheritance in FORMULA

- FORMULA doesn't support inheritance, but supports algebraic data types (equivalent with mathematical unions)
- Unions can simulate multiple-inheritance and interfaces




```
domain CVRIA
{
__FCO::= new ([params]).
__Physical_Object::= new ([params]).
__PONoAppObj::= new ([params]).
__POWithAppObj::= new ([params]).
__Support_System::= new ([params]).
__Center::= new ([params]).
__Traveler_Device::= new ([params]).
__Field_Equipment::= new ([params]).
__Vehicle::= new ([params]).

FCO ::= Physical_Object + __FCO.
Physical_Object ::= PONoAppObj + POWithAppObj +
                    __Physical_Object.
PONoAppObj ::= Support_System + __PONoAppObj.
POWithAppObj ::= Field_Equipment + Center +
                 Traveler_Device + Vehicle
                 + __POWithAppObj.
Field_Equipment ::= __Field_Equipment.
}
```

Terms with the new keyword are the building blocks (*primitives*) of instance models.
They can be *instantiated*.
The users don't work with these complicated structures, they are for the FORMULA engine.

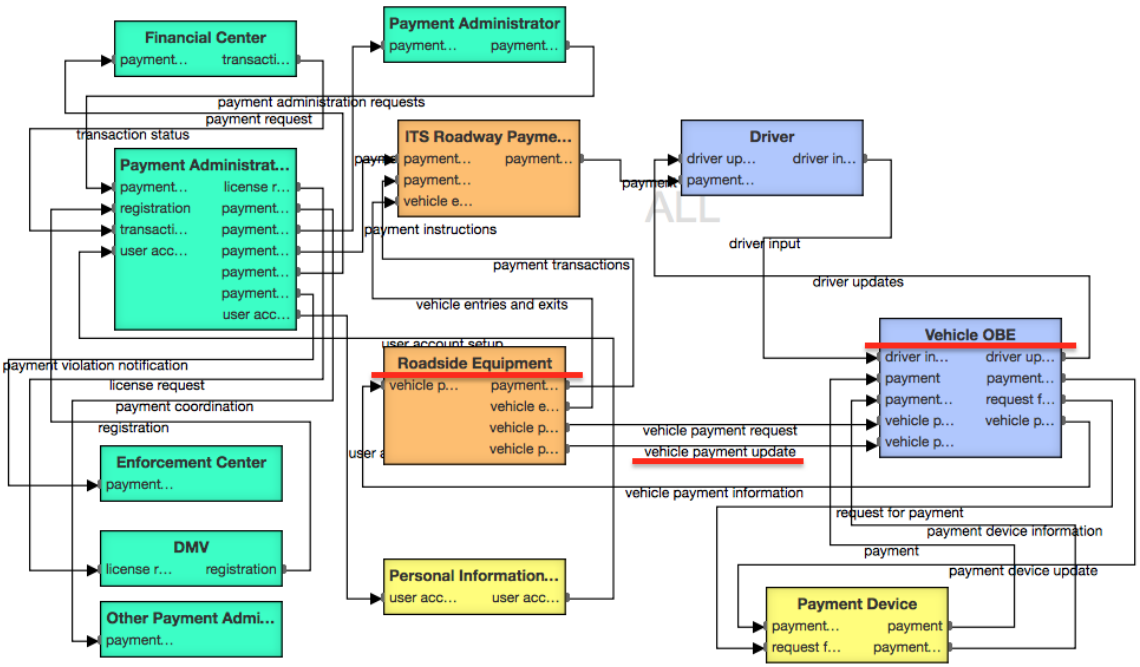
Unions, represent the listed types. Unions are abstract definitions, can be used in *rules*, *constraints*, *transformations*, but cannot be instantiated.

```
domain DConstraints extends CVRIA
{
  driverExists :- driver is Driver,
                  driver.attributes.name = "John".

  oneFinCenter :- building is Center,
                  building.attributes.id = "46".
}
```

Users write their code using the unions (and not the complicated primitives).

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Results of WebGME -> FORMULA Translation (Instance model – Electronic Toll Collection App)




```
model CVRIA of DConstraints
{
EpFG4zcpjI is __Vehicle("/817592481/U/P/q",
                        YCcryQqBUL,
                        qsakawH193,
                        x6MNWVCk6P, NULL).
x6MNWVCk6P is Attr__Vehicle("Vehicle OBE", "4").

ZbZCjvrXcE is __Field_Equipment("/817592481/U/P/O",
                                eFqxjm2S0Y,
                                qsakawH193,
                                spZ9Q6ZPLI, NULL).


spZ9Q6ZPLI is Attr__Field_Equipment("Roadside
Equipment", "11").

7EkawrMiB is __Physical_Input_Port("/817592481/U/P/q/s",
                                   NULL, EpFG4zcpjI,
                                   l21cLfaRB1, NULL).

aDozrEJx is __Physical_Output_Port("/817592481/U/P/O/7",
                                   NULL, ZbZCjvrXcE,
                                   QpOGstwkSj, NULL).

QpOGstwkSj is Attr__Physical_Output_Port("vehicle
payment update").

l21cLfaRB1 is Attr__Physical_Input_Port("vehicle payment
update").

niKfMfhuje is __InformationFlow("/817592481/U/P/0W",
                                JNaDozrEJx, K7EkawrMiB,
                                XSfylRKaPp, qsakawH193,
                                NJbT2zfEAl, NULL).
}
```


FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Results of WebGME -> FORMULA Translation (Constraint injection)

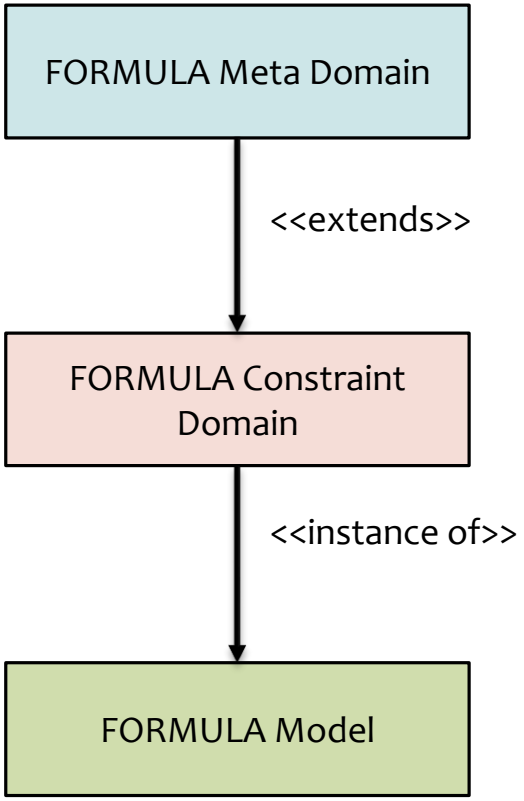
```
1  // Collection of added readers
2  AddedReader ::= (port:Physical_Port, policy:Policy, reader:Principal).
3  AddedReader(dstPort, dstPol, reader) :-
4      if is InformationFlow,
5      srcLabel is __SecurityLabel(_,_,if.src,_),
6      dstLabel is __SecurityLabel(_,_,if.dst,_),
7      dstPol is Policy, dstPol.parent=dstLabel,
8      srcPol is Policy, srcPol.parent=srcLabel,
9      srcReader is Reader, srcReader.parent=srcPol,
10     dstReader is Reader, dsrReader.parent=dstPol,
11
12     // The reader is new reader if it is not specified on the source port.
13     reader = dstReader.pointers.referTo,
14     no srcReader.pointers.referTo.
15
16 // The propagation constraint is satisfied if no readers are added
17 Propagation :- no AddedReader.
```

```
domain DConstraints extends CVRIA
{
// Collection of the added readers
AddedReader ::= (port:Physical_Port, policy:Policy,
reader:Principal).

AddedReader(dstPort, dstPol, reader) :-
  if is InformationFlow,
  srcLabel is __SecurityLabel(_,_,if.src,_),
  dstLabel is __SecurityLabel(_,_,if.dst,_),
  dstPol is Policy, dstPol.parent=dstLabel,
  srcPol is Policy, srcPol.parent=srcLabel,
  srcReader is Reader, srcReader.parent=srcPol,
  dstReader is Reader, dsrReader.parent=dstPol,

// The reader is new reader if it is not specified
// on the source port.
  reader = dstReader.pointers.referTo,
  no srcReader.pointers.referTo.

// The propagation constraint is satisfied
// if no readers are added
Propagation :- no AddedReader.
}
```


FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

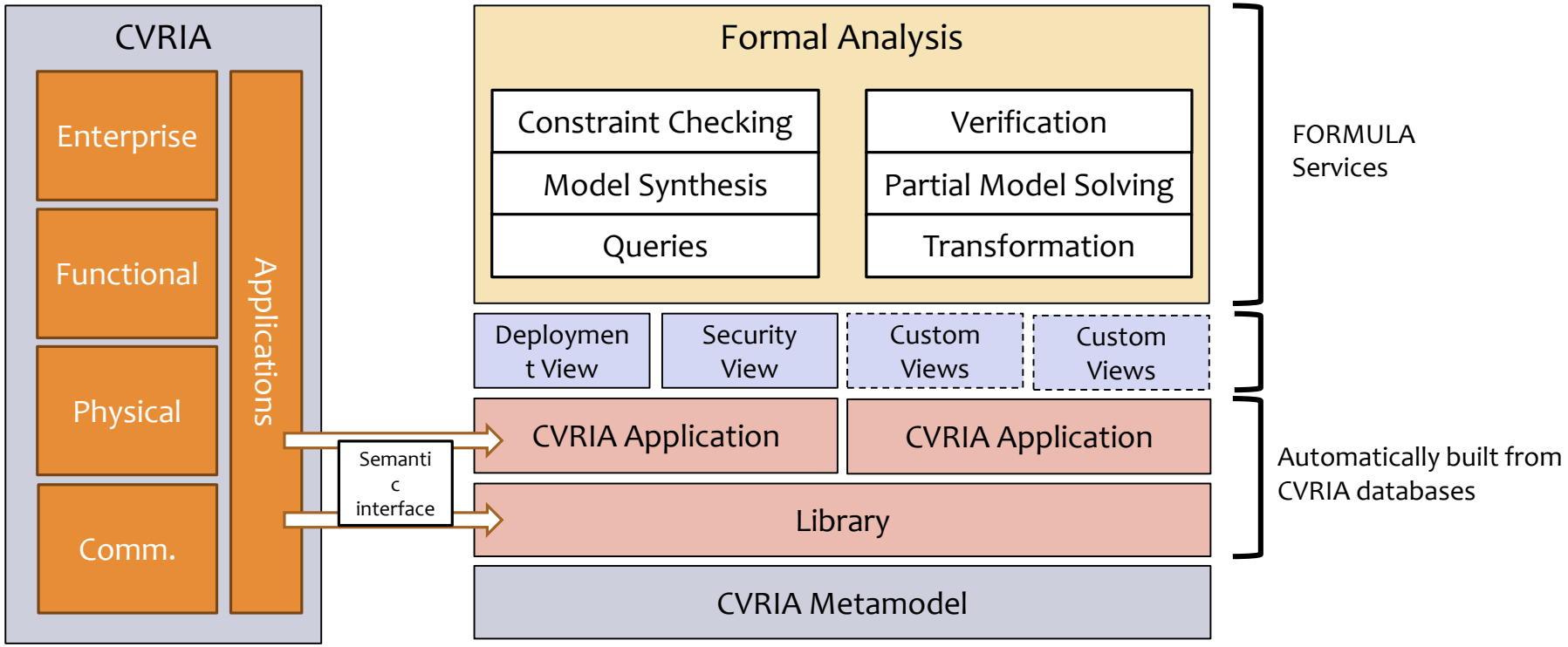
All FORMULA Pieces Together

```
// Metamodel specification
domain CVRIA
{
// …
__Physical_Object ::= new (id:String,
                           base: any __Physical_Object + {NULL},
                           parent: any GMENode + {NULL},
                           attributes: Attr__Physical_Object + {NULL},
                           pointers: {NULL}).
// …
}

// Injected constraints
domain DConstraints extends CVRIA
{
// …
AddedReader ::= (port:Physical_Port, policy:Policy, reader:Principal).
// See full constraint on previous slide
// …

Propagation :- no AddedReader.
}

// Instance Model
model CVRIA of DConstraints
{
// …
EpFG4zcpjI is __Vehicle("/817592481/U/P/q",YCcryQqBUL,qsakawH193, x6MNWVCk6P,
NULL).
x6MNWVCk6P is Attr__Vehicle("Vehicle OBE", "4").
// …
}
```



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Validation Workflow for CVRIA

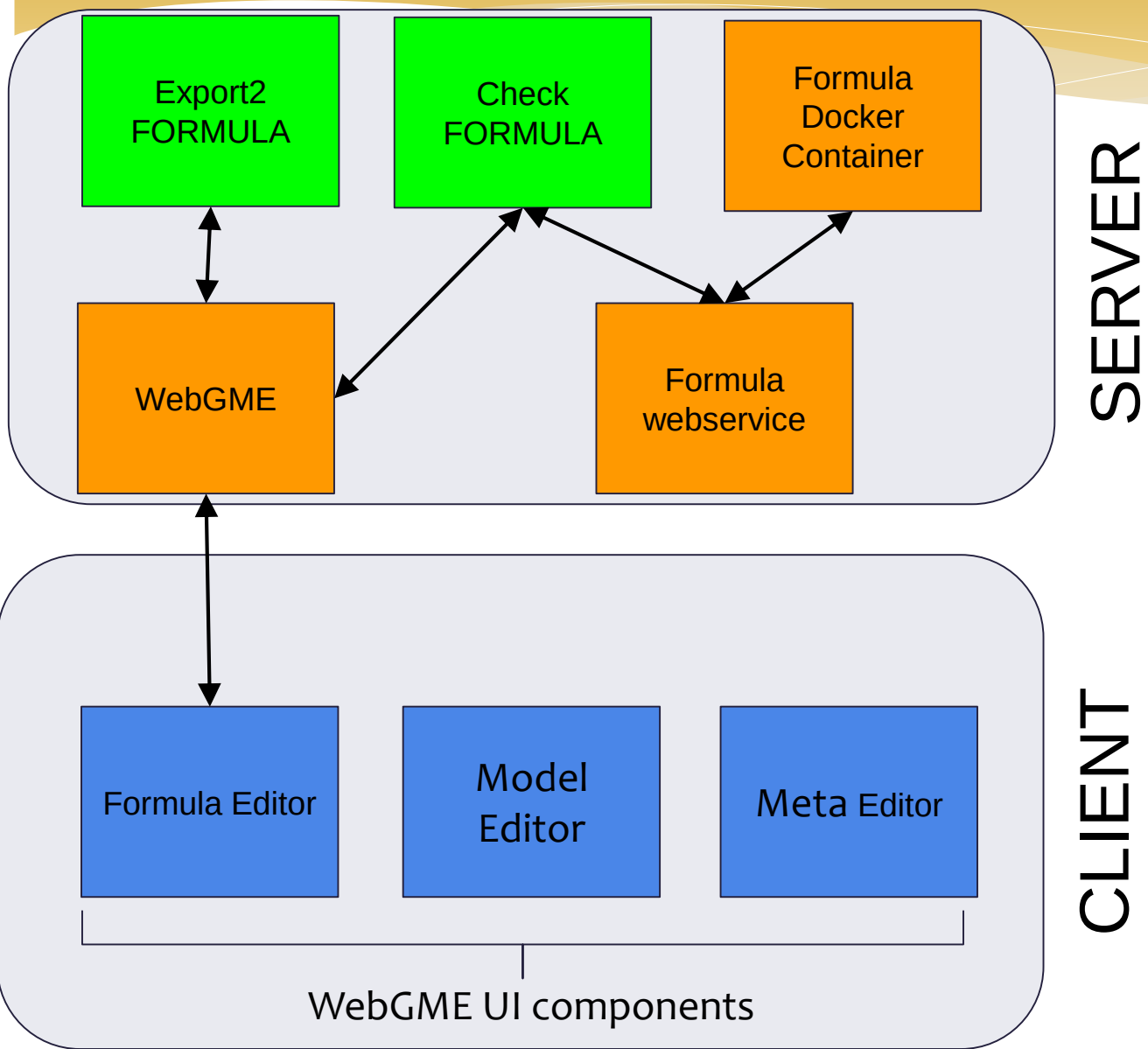


FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Content

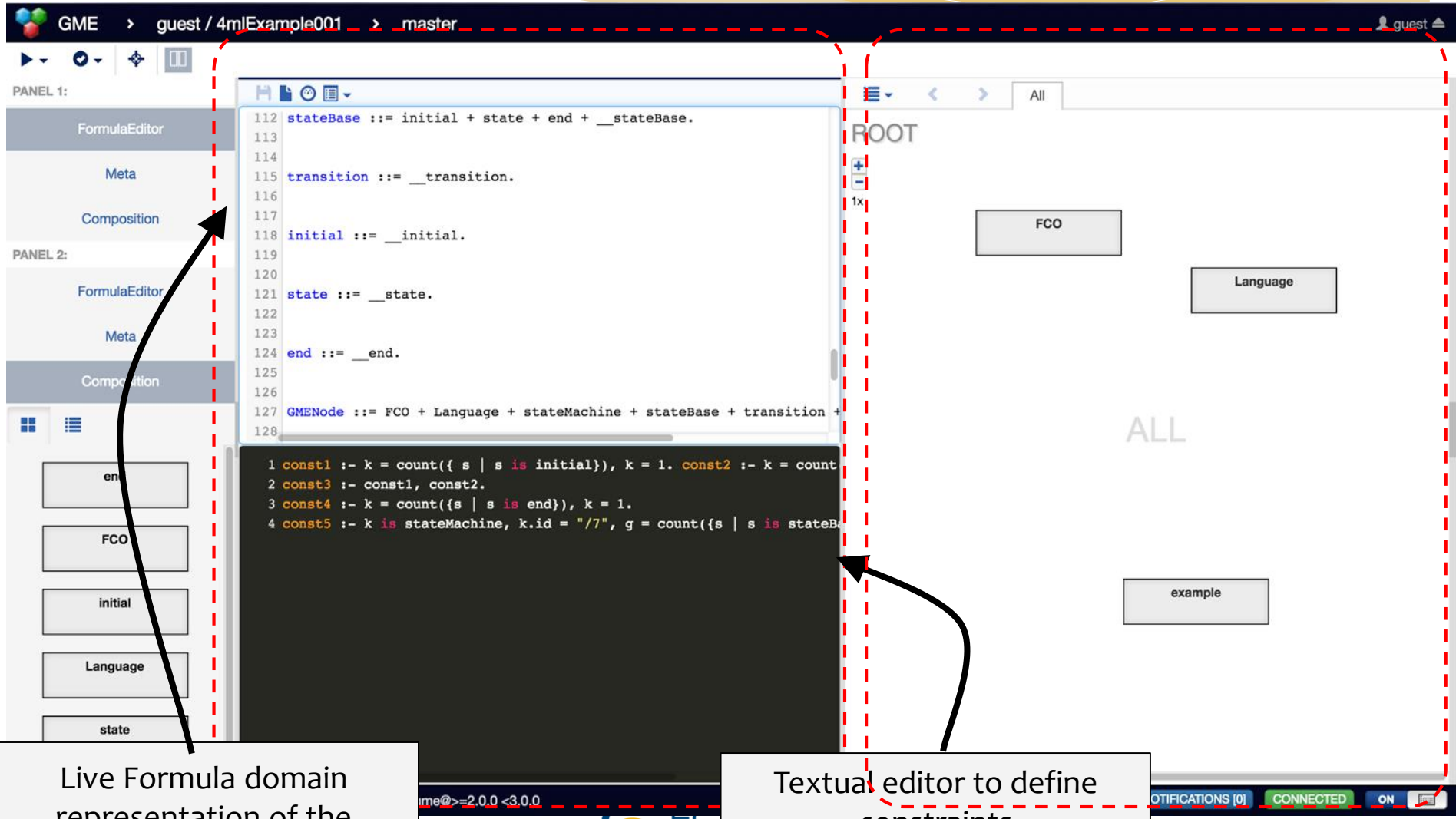
1. Background
   - Goals
   - Theory
   - **Validation Agenda**
2. Integrated Tool Suite
   - Functional Architecture
   - WebGME-FORMULA Integration
   - **Deployment Architecture**

Tool Architecture



1. **Model translation** (Formula Editor initiates towards WebGME that executes Export2FORMULA plugin and respond with a link to the Formula domain and model with the user defined constraints)
2. **Constraint checking** (Formula Editor initiates towards WebGME that executes CheckFORMULA which uses the Formula webservice that gets the result of step 1 and runs it on a docker container and responds with the true/false result per constraint)

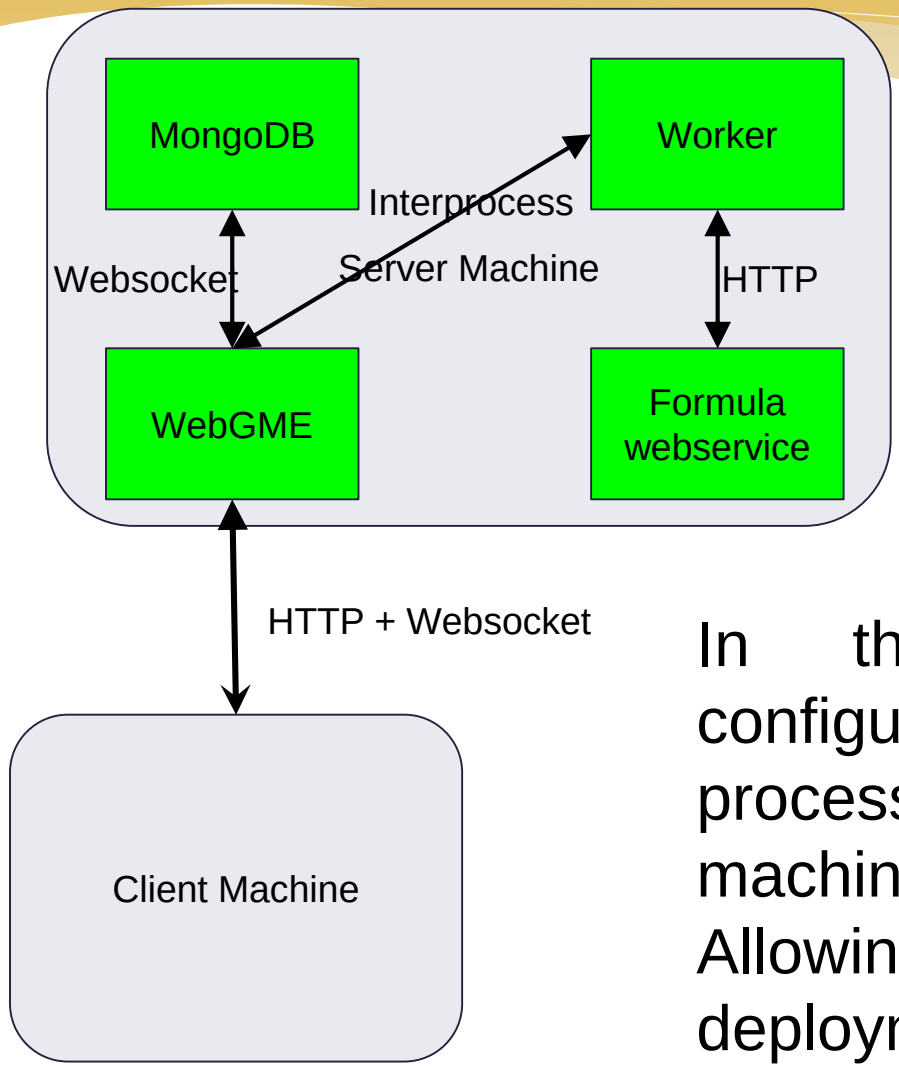
Client Layout



Live Formula domain representation of the WebGME projects' language

Textual editor to define constraints

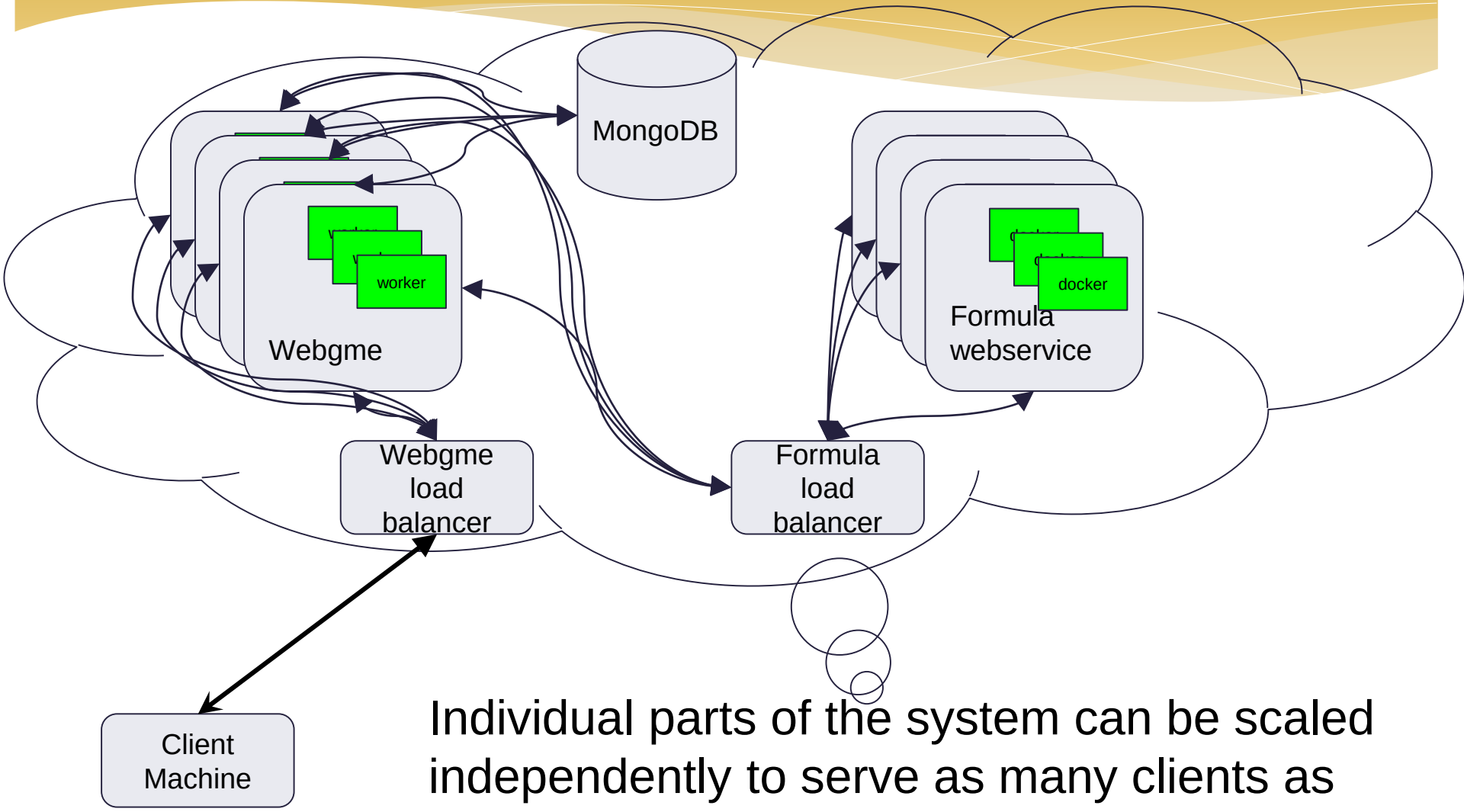
Minimal Deployment



In the smallest deployment configuration all the necessary processes can live on the same machine on the server side. Allowing a quick yet fully functional deployment.

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Scalable Deployment



Individual parts of the system can be scaled independently to serve as many clients as needed.

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Next Steps

* Completions of the WebGME-FORMULA integration
* Synthesis use cases
* CPS-VO deployment