

Semantic Backplane for Model-Based Development

*Dr. Daniel Balasubramanian, Senior Research Scientist
Institute for Software Integrated Systems, Vanderbilt University*

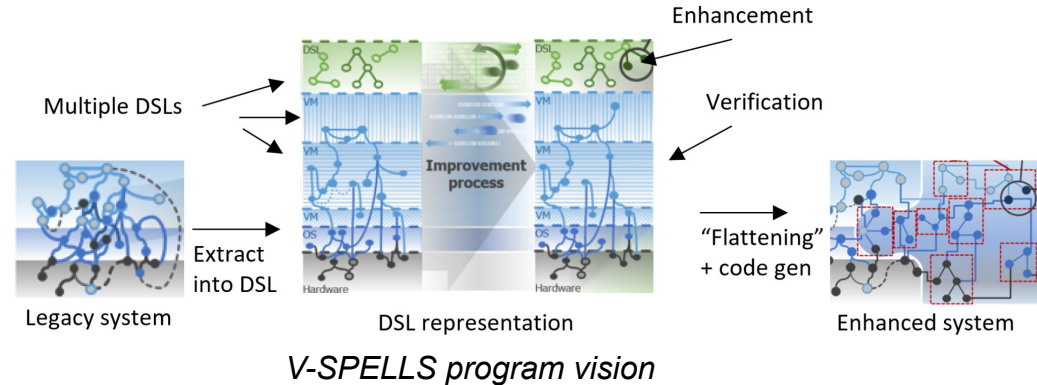
*Collaborators: Dr. Janos Sztipanovits, Dr. Qishen Zhang, Vanderbilt University
Dr. Eunsuk Kang, Carnegie Mellon University*

Outline

- **Background**
- Semantic backplane uses
- Open problems

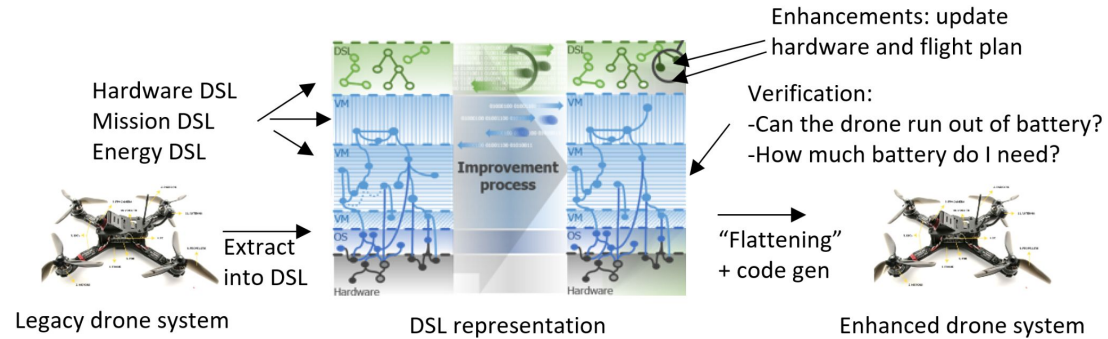
Background: DARPA V-SPELLS Program

- Goal: enable the verification of *enhancements* to legacy systems using domain-specific languages (DSLs)
 - Ensure updates and security patches are *compatible* with existing system



Background: DARPA V-SPELLS Program

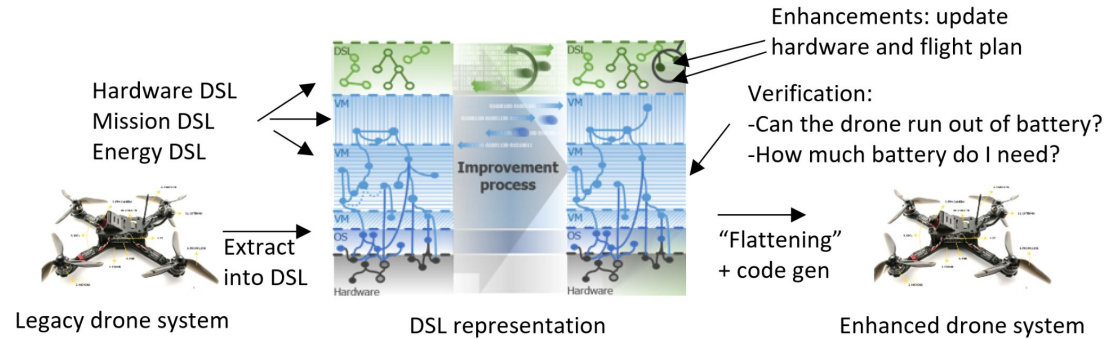
- Goal: enable the verification of *enhancements* to legacy systems using domain-specific languages (DSLs)
 - Ensure updates and security patches are *compatible* with existing system



Example V-SPELLS enhancement

Background: DARPA V-SPELLS Program

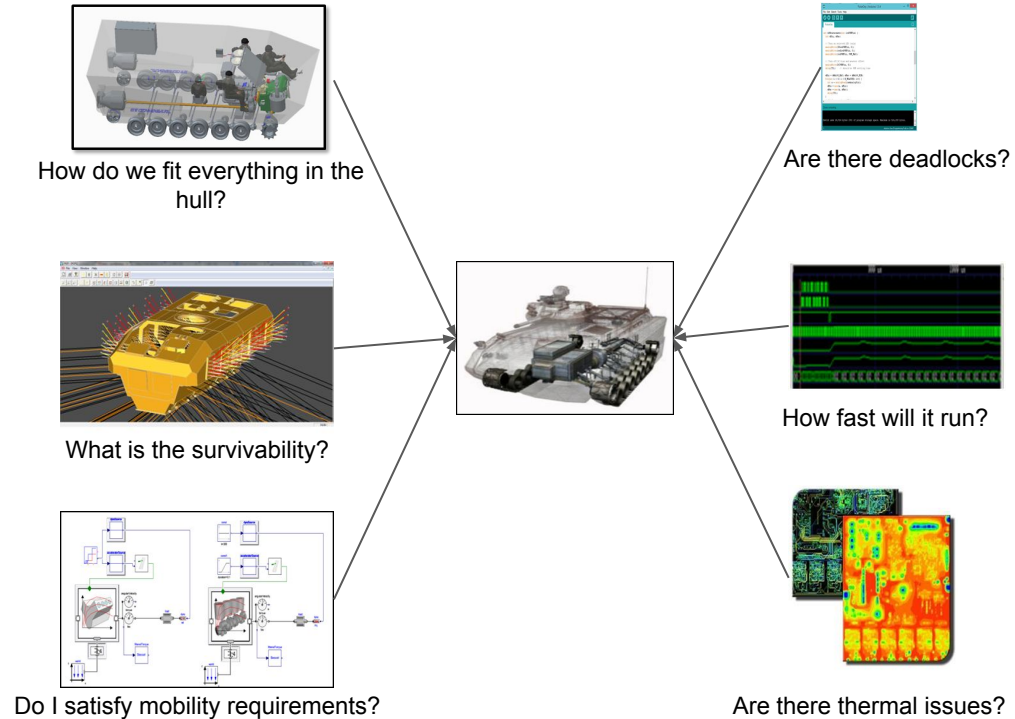
- Goal: enable the verification of *enhancements* to legacy systems using domain-specific languages (DSLs)
 - Ensure updates and security patches are *compatible* with existing system
- Challenges:
 - How do we verify cross-domain properties?
 - How do we “complete” the missing pieces of a system?



Example V-SPELLS enhancement

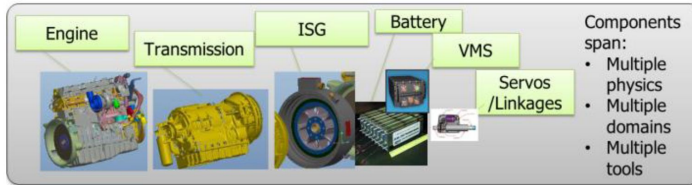
Background: DARPA AVM Program

- Goal: revolutionize the development of complex CPS
 - META program: create a tool chain for model-based design

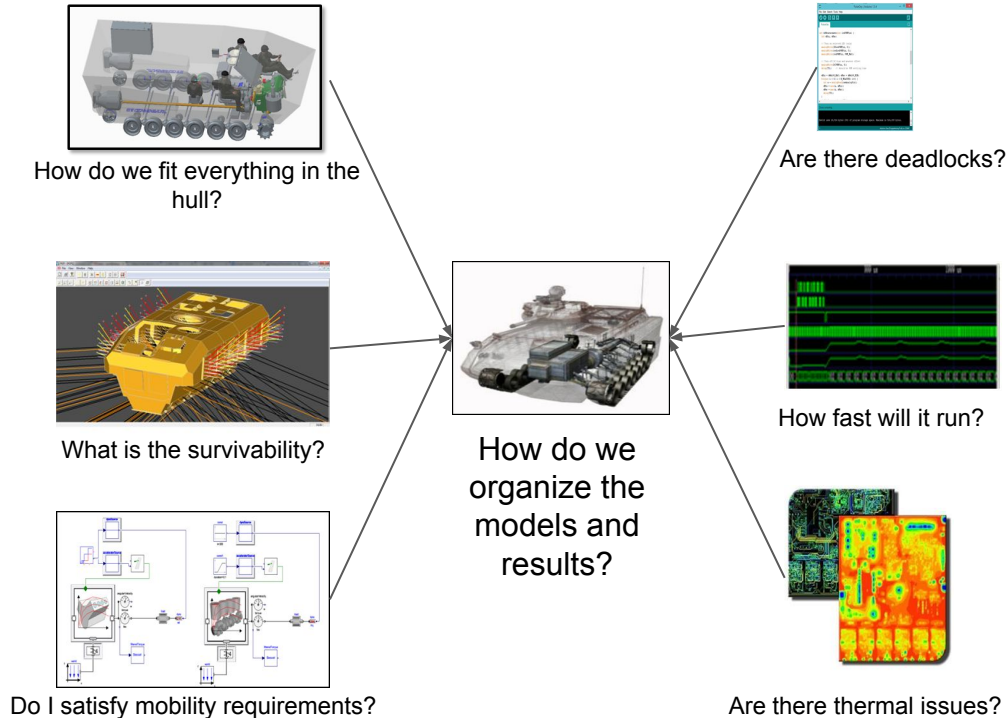


Background: DARPA AVM Program

- Goal: revolutionize the development of complex CPS
 - META program: create a model-based design flow and tool chain
- Challenges:
 - How do we integrate heterogeneous models and languages?

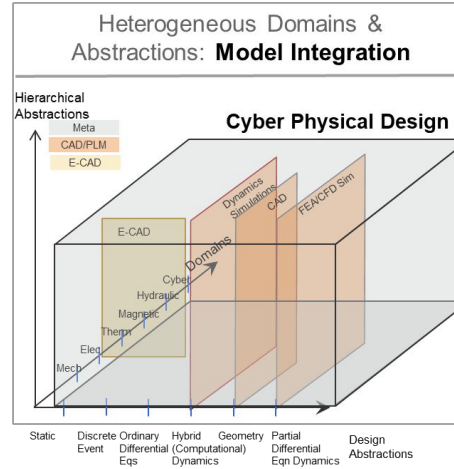


Components are heterogeneous and span multiple domains and tools

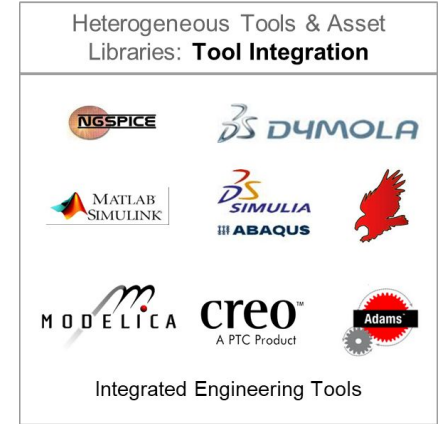


Background: DARPA AVM Program

- Goal: revolutionize the development of complex CPS
 - META program: create a model-based design flow and tool chain
- Challenges:
 - How do we integrate heterogeneous models and languages?
 - How do we verify cross-domain properties?



Models are different

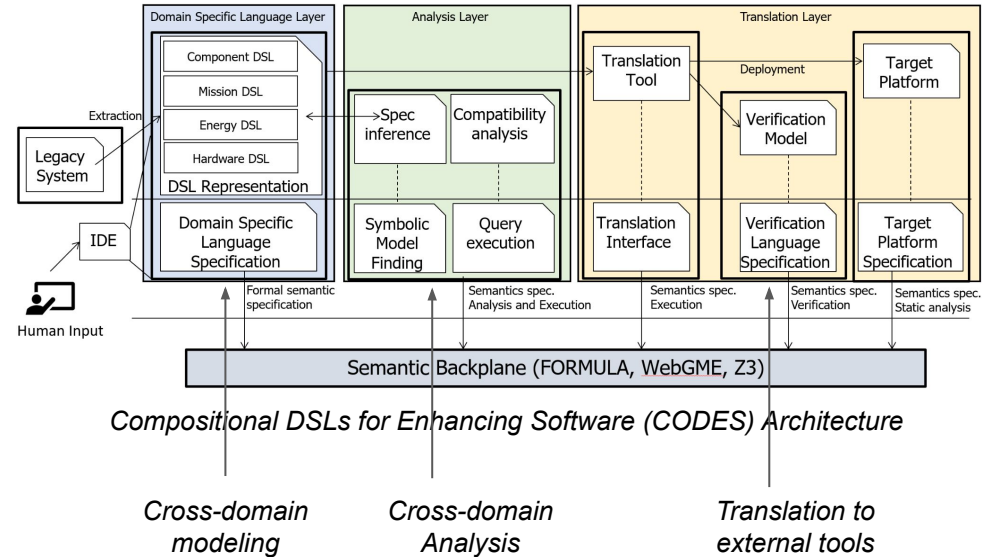


Tools are different

A design tool chain covering all CPS modeling abstractions is unrealistic

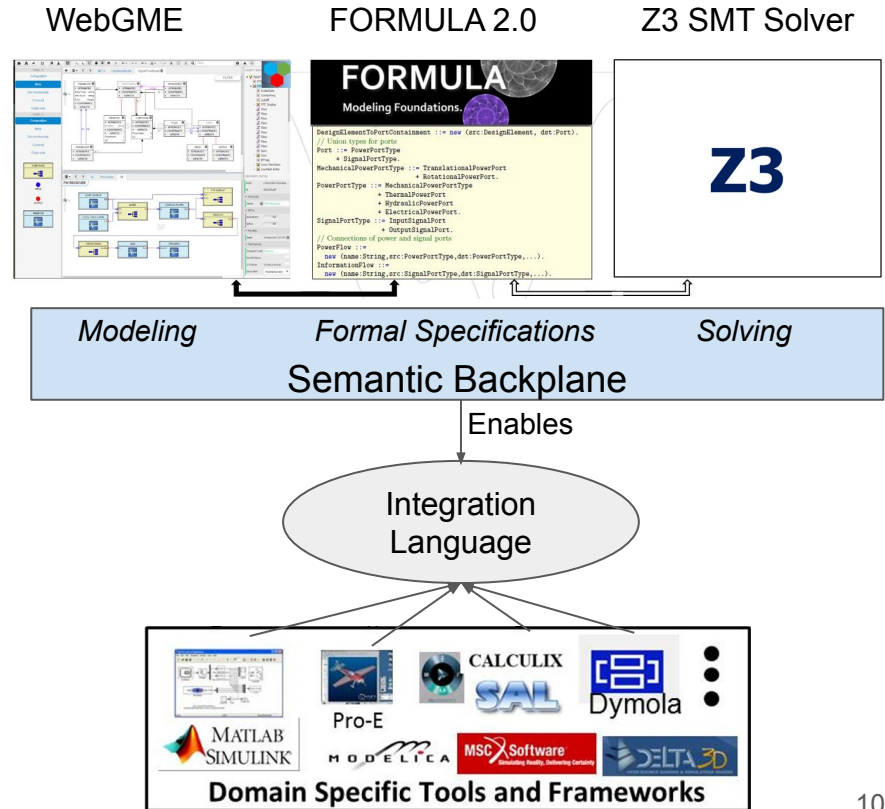
V-SPELLS Semantic Backplane

- System architecture on the right
- Semantic backplane:
 - FORMULA 2.0: formal specifications of DSLs
 - WebGME: graphical user interface for modeling
- Enables:
 - Analysis of cross-domain properties
 - Complete the missing pieces of the system



AVM Semantic Backplane

- Semantic backplane:
 - WebGME: metamodeling/modeling
 - FORMULA 2.0: formal specifications
 - Z3 is the solver for FORMULA's model finding procedure
- Enables:
 - Composition of heterogeneous models and languages through a model-integration language
 - Verification of properties across domains



Outline

- Background
- **Semantic backplane uses**
- Open problems

FORMULA 2.0

- Language and tool for formally specifying DSLs
- Originally developed at Microsoft Research
 - Fork actively maintained at Vanderbilt
- Open-world Logic Programming (OLP) with
 - Algebraic data types
 - First-order logic with fix-point operations
- Automated reasoning is enabled by symbolic execution and Z3

```
1 domain Mapping
2 {
3   Component ::= new (id: Integer, utilization: Real).
4   Processor  ::= new (id: Integer).
5   Mapping    ::= new (c: Component, p: Processor).
6
7   // The utilization must be > 0
8   invalidUtilization :- c is Component, c.utilization <= 0.
9
10  badMapping :- p is Processor,
11             s = sum(0.0, { c.utilization |
12                 | c is Component, Mapping(c, p) }), s > 100.
13
14  conforms no badMapping, no invalidUtilization.
15 }
```

Given this software component domain

```
1 model m of Mapping
2 {
3   c1 is Component(0, 10).
4   c2 is Component(1, 90).
5   p1 is Processor(0).
6   Mapping(c1, p1).
7   Mapping(c2, p1).
8 }
```

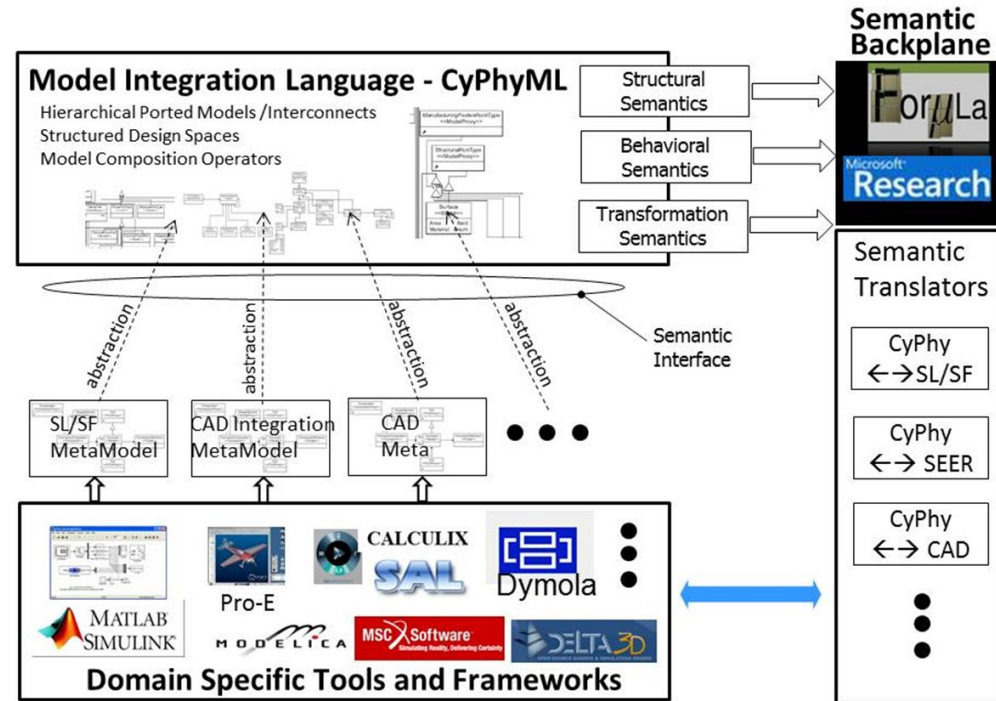
We can check whether this concrete model is valid

```
1 partial model pm of Mapping
2 {
3   c1 is Component(0, x).
4   c2 is Component(1, y).
5   p1 is Processor(0).
6   Mapping(c1, p1).
7   Mapping(c2, p1).
8 }
```

We can generate values for x and y that make this *partial* model valid

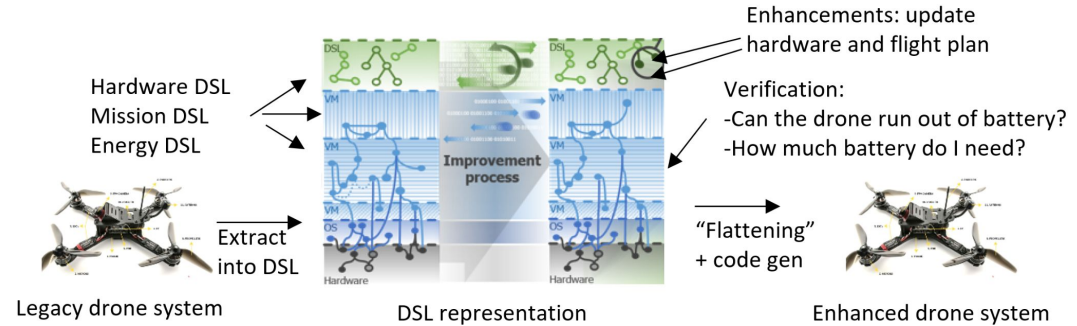
AVM Model Integration Language

- A design tool chain covering all CPS modeling abstractions is unrealistic
- Instead, we created a Model Integration Language
 - MIL changed frequently because component models are built with different tools
- Created FORMULA specs for:
 - Interface semantics
 - Model integration constructs
 - Model transformations



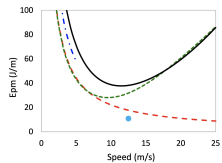
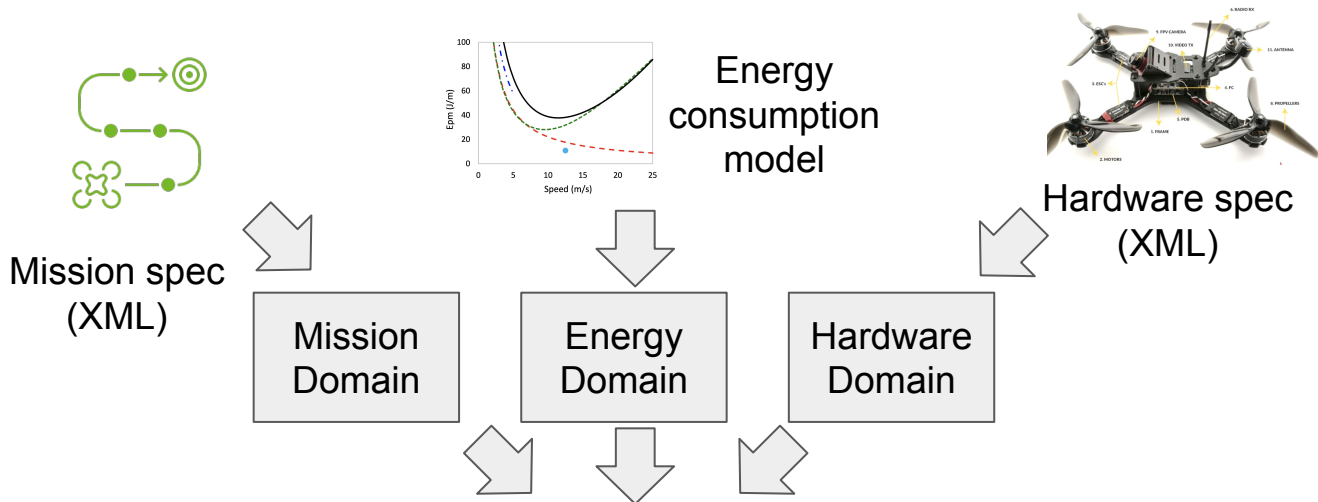
V-SPELLS: Cross-Domain Reasoning

- Use case: I want to update the hardware and flight plan
- Questions:
 - Can the drone fly without running out of battery?
 - How much battery do I need?
- Requires:
 - Reasoning over multiple, cross-cutting domains
 - Symbolic analysis to “fill-in” the required amount of battery



V-SPELLS: Cross-Domain Reasoning

Use case: I want to change the hardware and flight plan.



Energy consumption model



Hardware spec (XML)

```

domain BatteryChecker extends Energy, Mission
{
  missionConsumption ::= (String, Real).
  batteryExceeded ::= (String, Real, Real).

  batteryExceeded(missionName, consumed, available) :-
    missionConsumption(missionName, consumed),
    batteryCapacity(available),
    consumed > available.

  conforms no batteryExceeded(n, c, a).
}
    
```

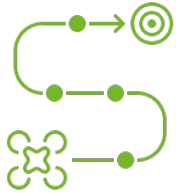
“Can the drone run out of battery during the mission?”

Developer queries



Cross-domain composition in FORMULA

Mission Domain Model in FORMULA



Mission spec

```
domain Mission {
  Loc ::= new (x : Real, y : Real).
  MissionItem ::= new (label : String, src : Loc,
                       dest : Loc, dist : Real, vel : Real).
  Mission ::= new (m : MissionItem,
                  remainder : any Mission + {NIL}).
  itemDuration ::= (String, Real).

  itemDuration(name, t) :- MissionItem(name, _, _, dist, vel),
                          t = dist/vel.
}
```

- Each mission is a sequence of mission items (travel between waypoints)
- Each mission item is associated with its estimated duration
- Representative of the Mission API in MAVSDK; could be extracted from source code

Battery & HW Models in FORMULA

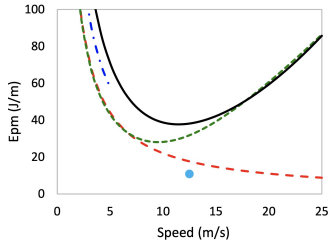


Hardware spec

```
domain WeightSpec {
  Component ::= new (label : String, weight : Real).
}
```

```
domain Battery includes WeightSpec {
  Battery ::= new (label : String, weight : Real,
                  capacity : Real).
  rate ::= (Real).
  batteryCapacity ::= (Real).
```

```
rate(r) :- x = sum(0, {cw | cw = w*(9.8), Component(_, w)}),
           r = (x + bw*9.8)/(3*0.7), Battery(_, bw, _).
batteryCapacity(c) :- c = sum(0, {bc | Battery(_, _, bc)}).
```



Energy consumption model

$$\frac{\sum_{k=1}^3 m_k g}{r \eta}$$

where m_k : drone + battery + payload mass
 r : lift-to-drag ratio
 η : power transfer efficiency

Cross-Domain Composition in FORMULA

```
domain BatteryAnalysis includes Battery, Mission {  
  missionConsumption ::= (String, Real).  
  batteryExceeded ::= (String, Real, Real).  
  
  missionConsumption(name, c) :- Mission(MissionItem(name, _, _, _, _),  
    Mission(MissionItem(name2, _, _, _, _), _)),  
    itemDuration(name, t1),  
    missionConsumption(name2, c2),  
    rate(r),  
    c = t1*r + c2.  
  
  ...  
  batteryExceeded(missionName, consumed, available) :-  
    missionConsumption(missionName, consumed),  
    batteryCapacity(available),  
    consumed > available.  
  
  conforms no batteryExceeded(n, c, a).  
}
```

Computes the total energy consumed for mission "name"

What it means for a mission to exceed available battery

Check to ensure mission "n" does not exceed battery

Specifying Instances in FORMULA

```
model sample_drone of BatteryAnalysis
{
  Component("payload1", 5).
  Component("payload2", 3).
  Component("body", 10).
  Battery("battery1", 5, 200).

  t1 is MissionItem("task1", Loc(40.00, 5.00), Loc(47.00, 8.00), 7.62, 0.4).
  t2 is MissionItem("task2", Loc(47.00, 8.00), Loc(52.00, 2.00), 7.81, 0.2).

  m2 is Mission(t2, NIL).
  m1 is Mission(t1, m2).
}
```

- FORMULA checks the conformance constraints against the given instance (model)

Instance Finding in FORMULA

```
partial model sample_drone of BatteryAnalysis
```

```
{
```

```
  Component("payload1", 5).
```

```
  Component("payload2", 3).
```

```
  Component("body", 10).
```

```
  Battery("battery1", 5, x).
```

Symbolic variable "x";
defines a partial instance

```
t1 is MissionItem("task1", Loc(40.00, 5.00), Loc(47.00, 8.00), 7.62, 0.4).
```

```
t2 is MissionItem("task2", Loc(47.00, 8.00), Loc(52.00, 2.00), 7.81, 0.2).
```

```
m2 is Mission(t2, NIL).
```

```
m1 is Mission(t1, m2).
```

```
}
```

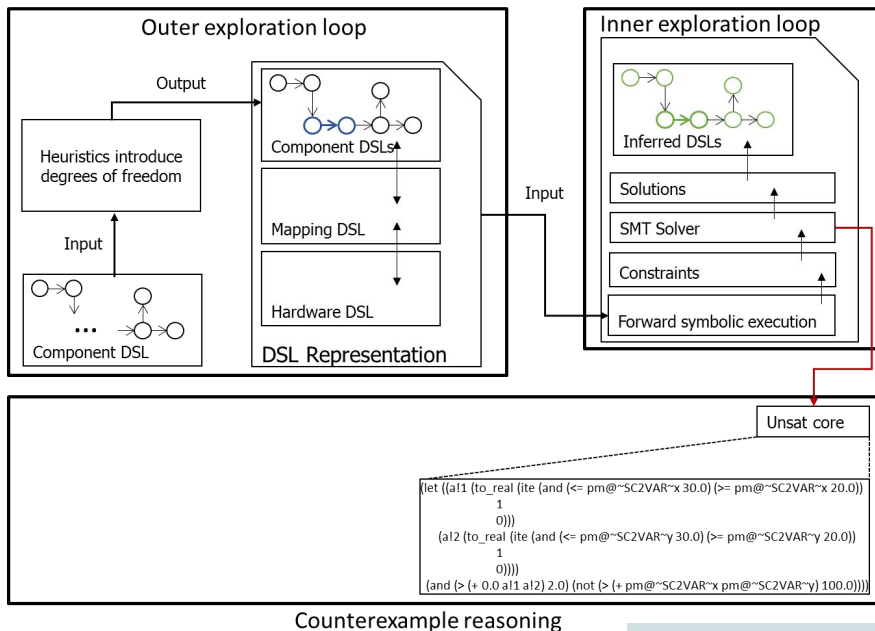
- Given a partial model, we can generate values for symbolic variables
 - e.g., "Find me the battery capacity sufficient to support the given mission"

Outline

- Background
- Semantic backplane uses
- **Open problems**

Open problems

2. A heuristic search “guesses” new elements
Example: Component (“a”, x)



3. Model finding fails
-> Constraints not satisfiable

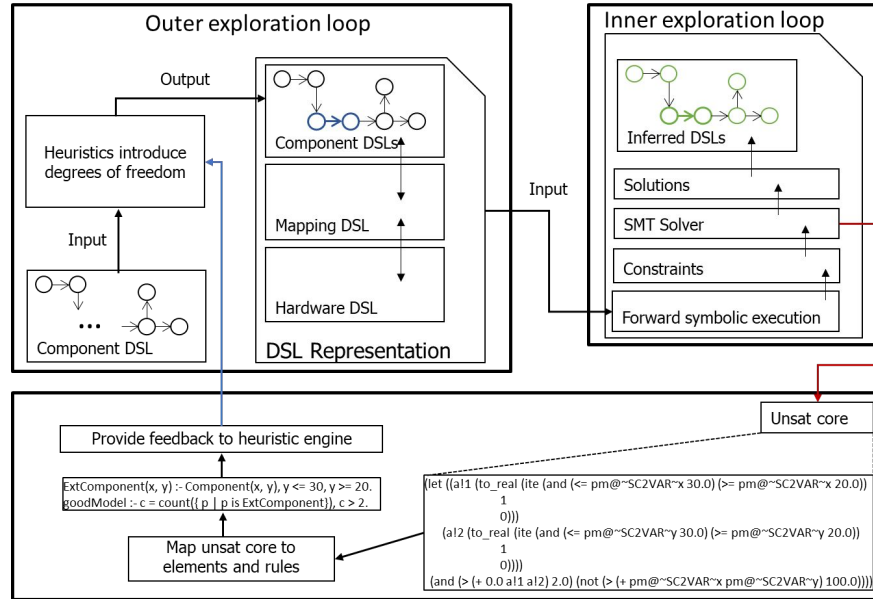
4. Solver output extracted

1. System model is extracted. Model may be “missing” elements

5. How can we help users (1) understand, (2) debug, and (3) repair models?

Open problem: explanation generation

2. A heuristic search “guesses” new elements
Example: Component (“a”, x)



3. Model finding fails
 -> *Constraints not satisfiable*

4. Solver output extracted

1. System model is extracted. Model may be “missing” elements

Counterexample reasoning: explanation generation + automated heuristic guidance

Solution 1: map core to terms and rules

```

let ((a11 (to_real (ite (and (<= pm@~SC2VAR~x 30.0) (>= pm@~SC2VAR~x 20.0))
    1
    0)))
    (a12 (to_real (ite (and (<= pm@~SC2VAR~y 30.0) (>= pm@~SC2VAR~y 20.0))
    1
    0))))
(and (> (+ 0.0 a11 a12) 2.0) (not (> (+ pm@~SC2VAR~x pm@~SC2VAR~y) 100.0))))
    
```

Open problem: debugging models

Idea: use a debugger-style interface to understand model execution

The screenshot shows a solver interface with several panels. A yellow box labeled "Constraints for selected term to be present" points to the "Direct Constraints" panel, which contains the constraint $1: ((y + x) > 100)$. A yellow box labeled "Derived terms after execution" points to the "Current Terms" panel, where item 10, "Mapping.badMapping", is highlighted in blue. A yellow box labeled "Solution" points to the "Solution/Unsat* Core" panel, which displays two solutions. A red dashed arrow points from the "Solution/Unsat* Core" panel back to the "Direct Constraints" panel.

```
domain Mapping
{
  Component ::= new (id: Integer, utilization: Real, processor: Processor)
  Processor ::= new (id: Integer)
  Mapping ::= new (c: Component, p: Processor)

  // The utilization must be > 0
  invalidUtilization :- c is Component, c.utilization <= 0

  badMapping :- p is Processor,
    s = sum(0.0, { c.utilization | c is Component, M
    Utilization.
```

File Manager: MappingExample.4ml

File: solve pm 1 Mapping.conforms

```
[> load /Users/daniel/work/formula/formula/Tst/
(Compiled) MappingExample.4ml
0.68s.

[> solve pm 1 Mapping.conforms
0.05s.

[> Use buttons in solver view to init, execute, a

Solve initialize task completed.

[> Solve execution task completed.

[> Solve start t
Solveable: true
22ms

[>
```

Solver: Symbolic Variables

Vars: Ops: Digit/Real

+ Constraint

All Constraints

Solution

Solution/Unsat* Core

Solution number 0
Processor(0)
Component(0, 1)
Component(1, 1)
Mapping(Component(0, 1), Processor(0))
Mapping(Component(1, 1), Processor(0))

Solution number 1
Processor(0)

Counter Example

Inference Rules

- invalidUtilization
- badMapping

Current Terms

- 3: Component(1, y)
- 4: Mapping(Component(0, x), Processor(0))
- 5: Mapping(Component(1, y), Processor(0))
- 6: Mapping.invalidUtilization
- 7: Mapping.invalidUtilization
- 8: pm.ensures
- 9: _Query_0000000000000000.ensures
- 10: Mapping.badMapping
- 11: Mapping.~conforms0

Direct Constraints

- 1: ((y + x) > 100)

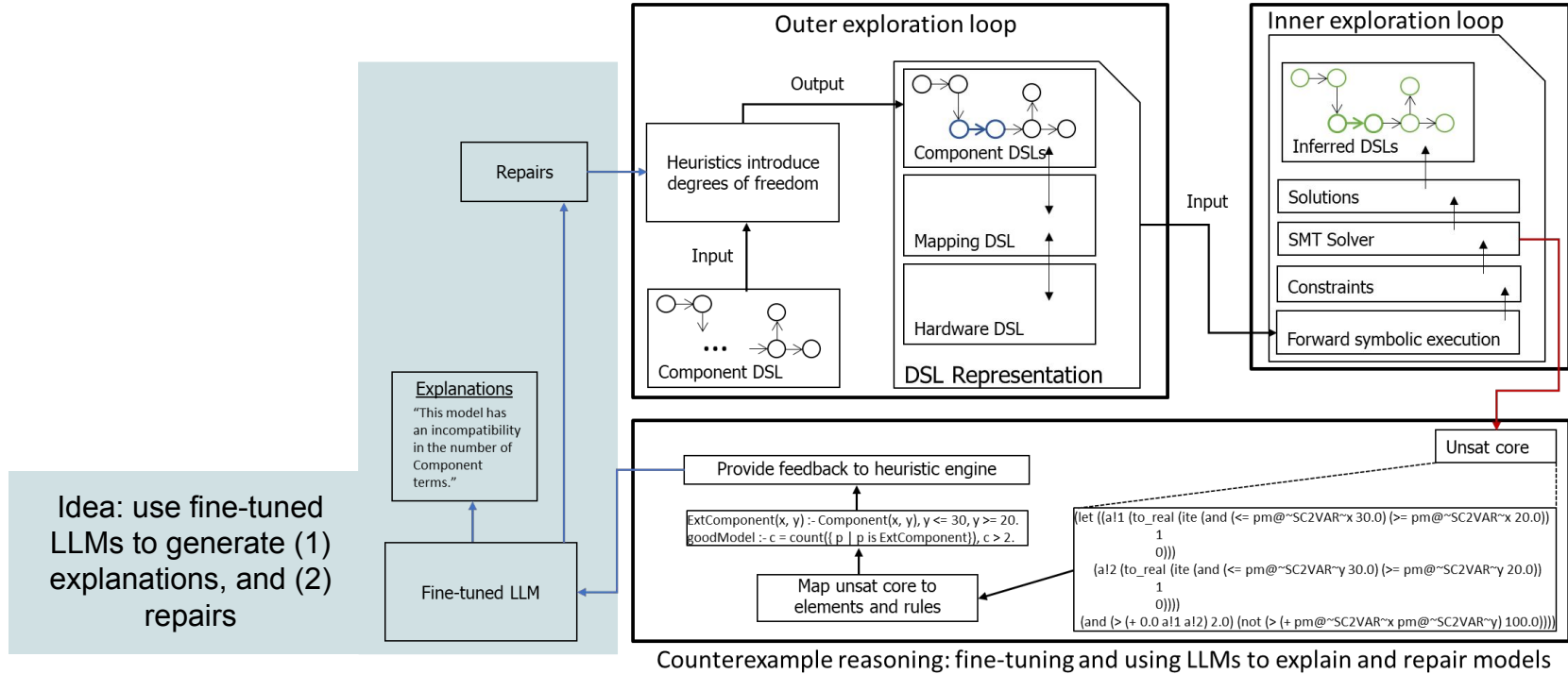
Positive Constraints

Negative Constraints

Flattened Constraints

- 1: ((y + x) > 100)

Open problem: model repairs



Examples available online

- Drone example (and others) available online at:

<https://formula.isis.vanderbilt.edu>

- Periodically taken down for updates
- Please report issues to

<https://github.com/VUISIS/formula/issues>

```
1 domain Battery
2 {
3   Component := new (label : String, weight : Real).
4   Battery := new (label : String, weight : Real, capacity : Real).
5
6   // Energy consumption rate
7   rate := (Real).
8
9   // Duration to complete the MissionItem with the given name
10  itemDuration := (String, Real).
11
12   // Amount of energy consumed to carry out the Mission with given name
13  missionConsumption := (String, Real).
14
15   // Total battery capacity
16  batteryCapacity := (Real).
17
18   // (x,y) location of the drone
19  loc := new (x : Real, y : Real).
20
21   // Each mission item involves moving from source to destination locs at g
22  MissionItem := new (label : String, src : Loc, dest : Loc, dist : Real, v
23
24   // Each mission is a list of mission items
25  Mission := new (n : MissionItem, remainder : any Mission + (NIL)).
26
27  rate(r) := x = sum(0, {cw | cw = we(9.8), Component(_, w)}),
28            r = (x + bw*9.8)/(3*0.7), Battery(_, bw, _).
29
30  batteryCapacity(c) := c = sum(0, {bc | Battery(_, _, bc)}).
31
32  itemDuration(name, t) := MissionItem(name, _, _, dist, vel), t = dist/vel.
33
34  missionConsumption(name, c) := Mission(MissionItem(name, _, _, _, _),
35  Mission(MissionItem(name2, _, _, _, _), _),
36  itemDuration(name, t1),
37  missionConsumption(name2, c2),
38  rate(r),
```

```
[>] (Compiled) tmp_file.4ml
0.94s.
[]> solve pm | Battery.conforms
Parsing text took: 1
Visiting text took: 0
Started solve task with Id 0.
0.07s.
[]> ls

Environment variables
Programs in file root
+-- /
+-- tmp
+-- formula-DnbAeQ [1 file(s)]
| tmp_file.4ml
Programs in env root
+-- /

All tasks
Id | Kind | Status | Result | Started | Duration
-----|-----|-----|-----|-----|-----
Common Commands

Solving:
solve pm | Mapping.conforms // Try to solve the partial model named
pm
ls // Check the result of a solve task
ex 0 0 0 // Extract solution 0 from solve task 0 ex 0 1 0 //
```

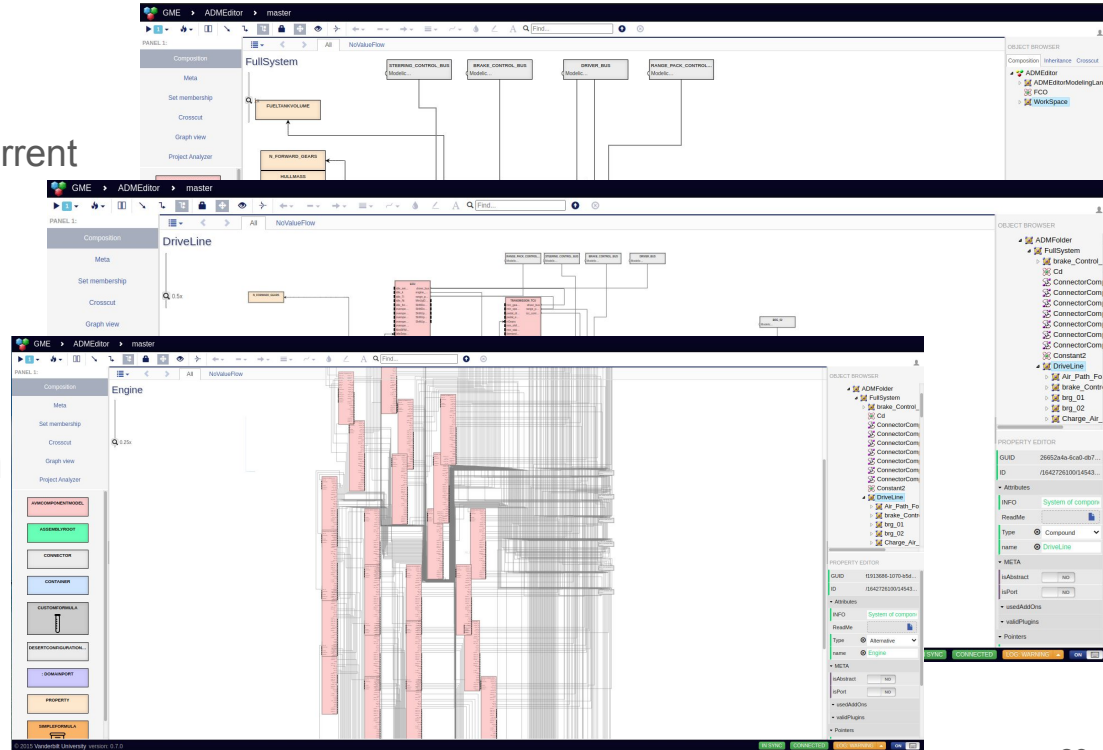
Contributions

- Presented our experiences building a semantic backplane for CPS
 - Tools: WebGME + FORMULA 2.0
 - Tight integration between “traditional” modeling tools and formal specification languages is essential
 - Cross-domain reasoning is essential
- Semantic backplane enables:
 - Integration of heterogeneous modeling languages
 - Cross-domain verification and symbolic analysis
- Open problems
 - Explanation generation, model debugging, model repair

Additional slides

WebGME

- Meta-programmable, visual modeling tool
 - Multi-user, collaborative, concurrent modeling
- Web application (thick browser-based client)
- Graphical interface can be customized to match domain notations



FORMULA 2.0 example: specification and verification

- Example: is every list of four integers sortable via adjacent compare and swaps?

Defines types →

Define what a counterexample is: an input that does not generate a sorted trace →

Recursively generate new traces →

We want to try to find a counterexample →

Our input model contains one list →

```
domain SymbolicOLP
{
  input    ::= new (w: Integer, x: Integer, y: Integer, z: Integer).
  trace    ::= (w: Integer, x: Integer, y: Integer, z: Integer).
  cntrexp  ::= (Integer, Integer, Integer, Integer).

  cntrexp(w, x, y, z) :- input(w, x, y, z),
    c = count({ t | t is trace, t.w <= t.x, t.x <= t.y, t.y <= t.z}),
    c = 0.

  trace(w, x, y, z) :- input(w, x, y, z).
  trace(x, w, y, z) :- trace(w, x, y, z), w > x.
  trace(w, y, x, z) :- trace(w, x, y, z), x > y.
  trace(w, x, z, y) :- trace(w, x, y, z), y > z.

  conforms cntrexp(a, b, c, d).
}

partial model pm of SymbolicOLP
{
  input(a, b, c, d).
}
```

FORMULA 2.0 example: specification and verification

- Example: is every list of four integers sortable via adjacent compare and swaps?
- Partial execution trace:

input(a, b, c, d)

trace(a, b, c, d)

trace(b, a, c, d), a > b

trace(a, b, c, d), a > b, b > a

Z3



Term not generated;
Recursion terminates

```
domain SymbolicOLP
{
  input    ::= new (w: Integer, x: Integer, y: Integer, z: Integer).
  trace    ::= (w: Integer, x: Integer, y: Integer, z: Integer).
  cntrexp  ::= (Integer, Integer, Integer, Integer).

  cntrexp(w, x, y, z) :- input(w, x, y, z),
    c = count({ t | t is trace, t.w <= t.x, t.x <= t.y, t.y <= t.z}),
    c = 0.

  trace(w, x, y, z) :- input(w, x, y, z).
  trace(x, w, y, z) :- trace(w, x, y, z), w > x.
  trace(w, y, x, z) :- trace(w, x, y, z), x > y.
  trace(w, x, z, y) :- trace(w, x, y, z), y > z.

  conforms cntrexp(a, b, c, d).
}

partial model pm of SymbolicOLP
{
  input(a, b, c, d).
}
```