# On the Challenges of Model Decorations for Capturing Complex Metadata


Horacio Hoyos Rodriguez
University of York, UK

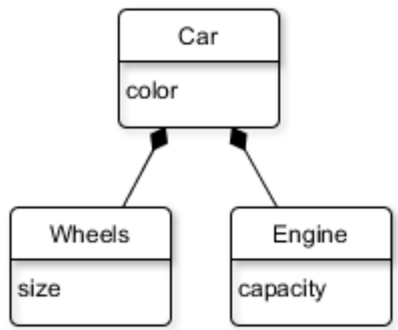
Athanasios Zolotas
University of York, UK

Dimitris Kolovos
University of York, UK

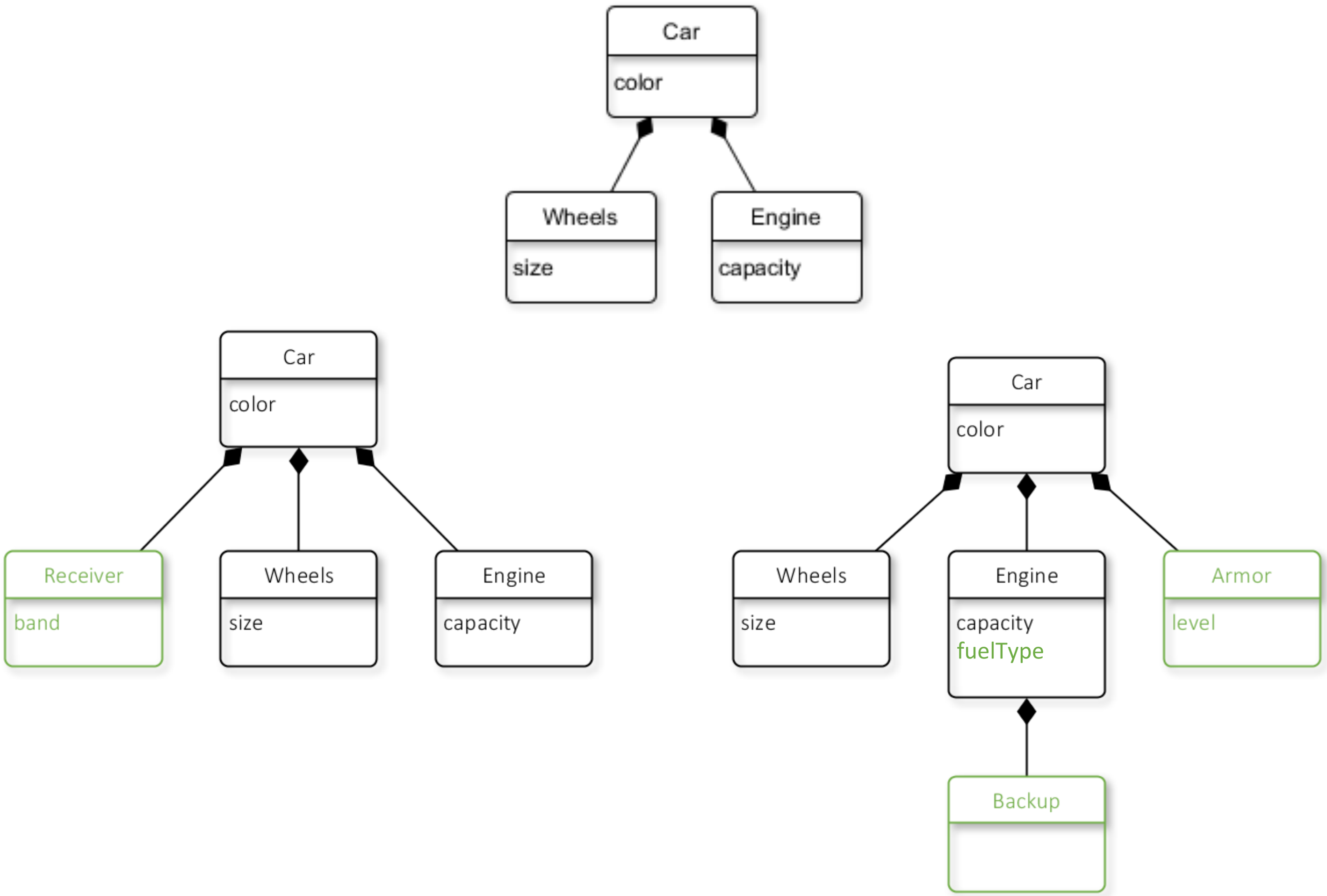
Richard F. Paige
University of York, UK
McMaster University, Canada

# The Problem

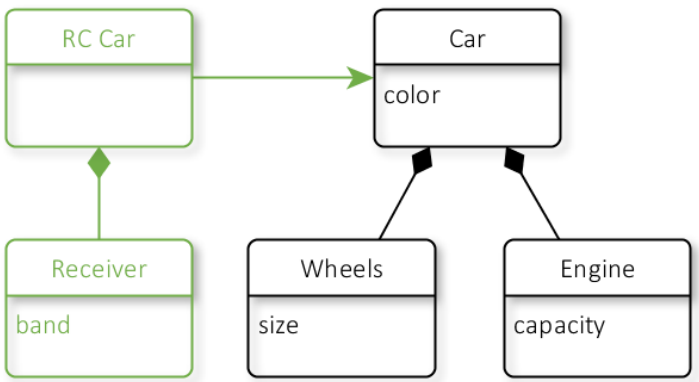
# Planning for the future



# Planning for the future

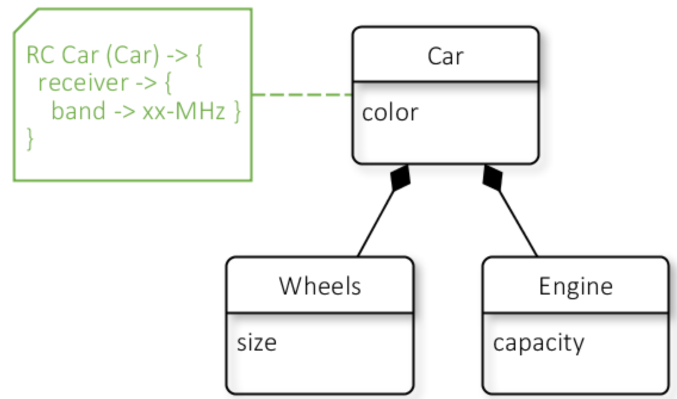


# Decorator languages




## Same technologies

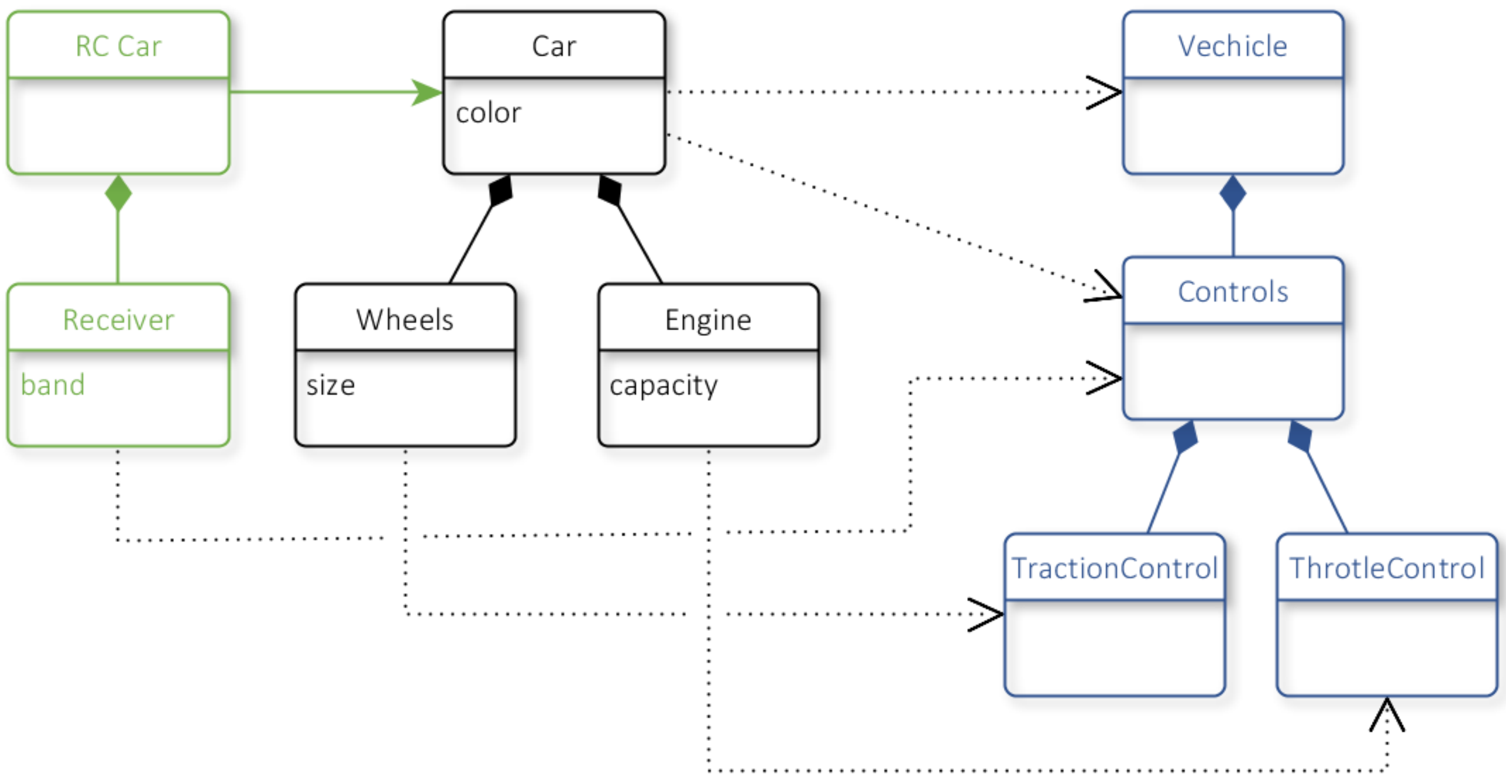
- Additional information is seamlessly captured in the model, i.e. its feasible to navigate between a **Car** and its **RC Car**.
- Tooling can be reused (with work)



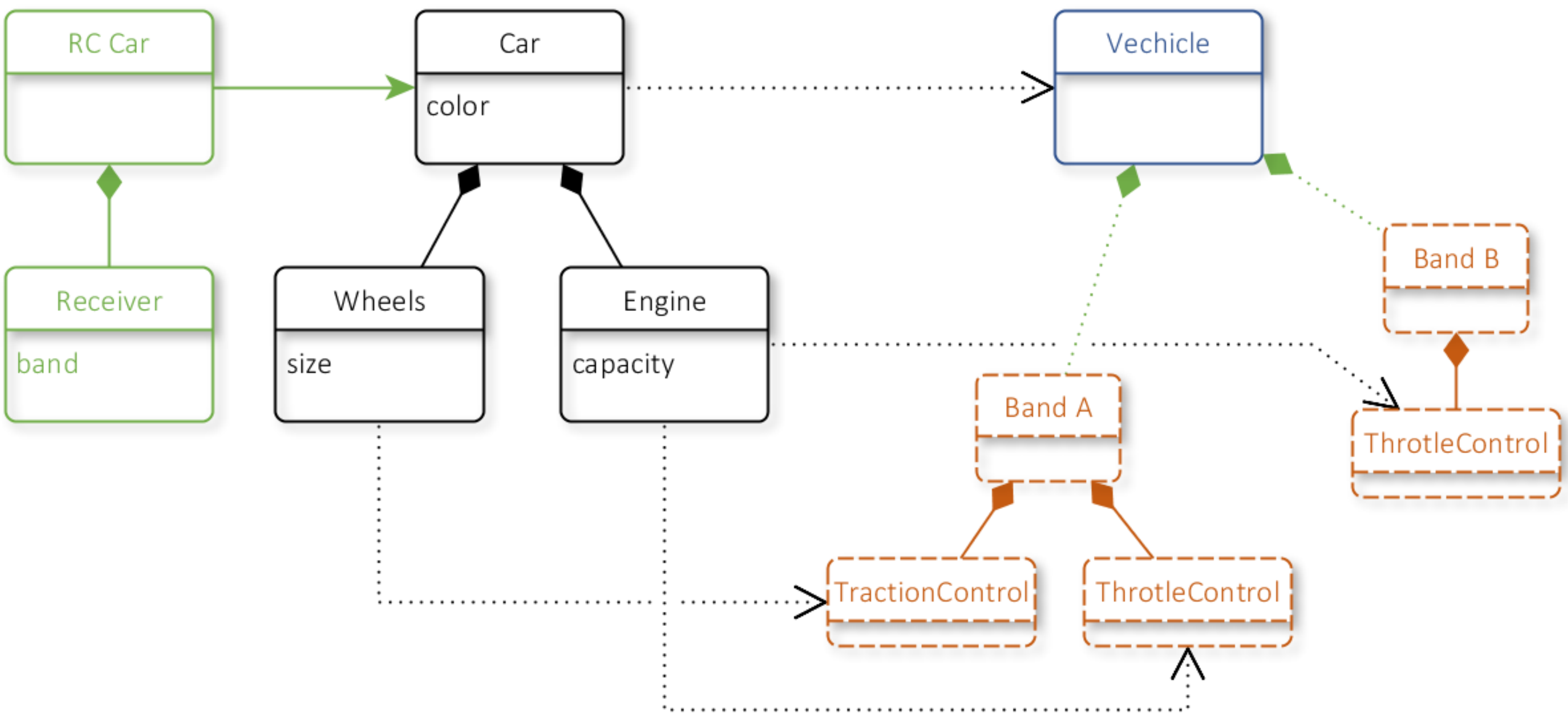

## Other technologies

- Separate artefact available during model management (i.e. as other model)
- Seamlessly navigation is not possible
- Tooling is not reusable

# Purpose of metadata



# Purpose of metadata

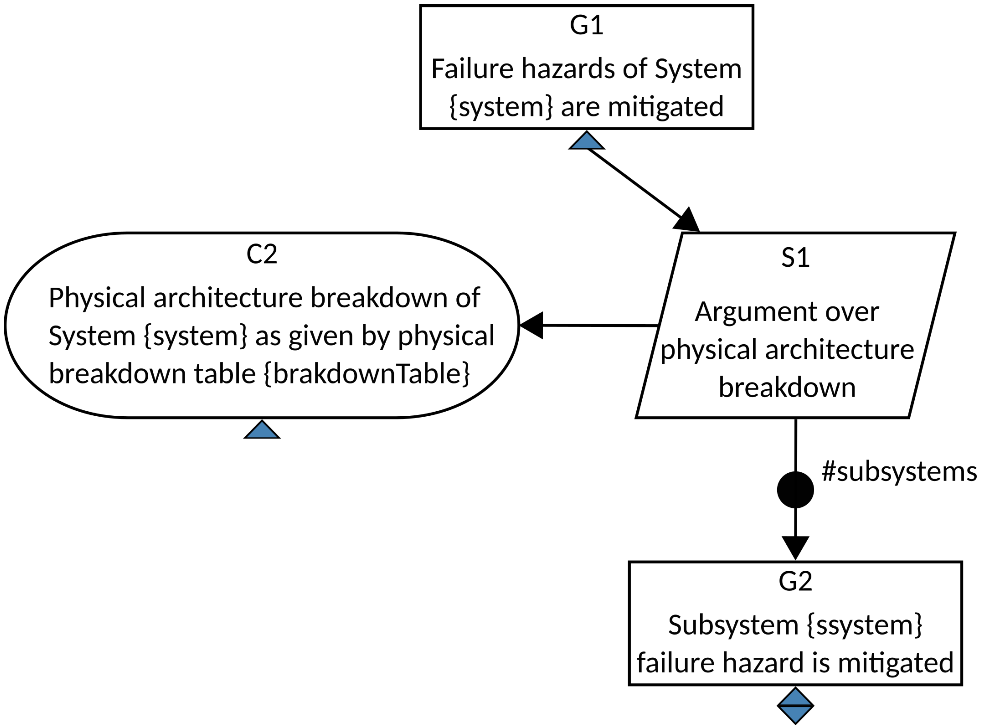


# Goal Structuring Notation (GSN)

- The development of assurance cases is a key part of engineering safety critical systems.
- An assurance case presents a structured argument aimed at ensuring that the safety or security of a system can be demonstrated with respect to evidence


- Safety cases are typically constructed manually, since many tools rely on diagrammatic drawing support input
- Interested in how the safety cases could be auto-generated and how the information required to generate them could be captured

# GSN Patterns

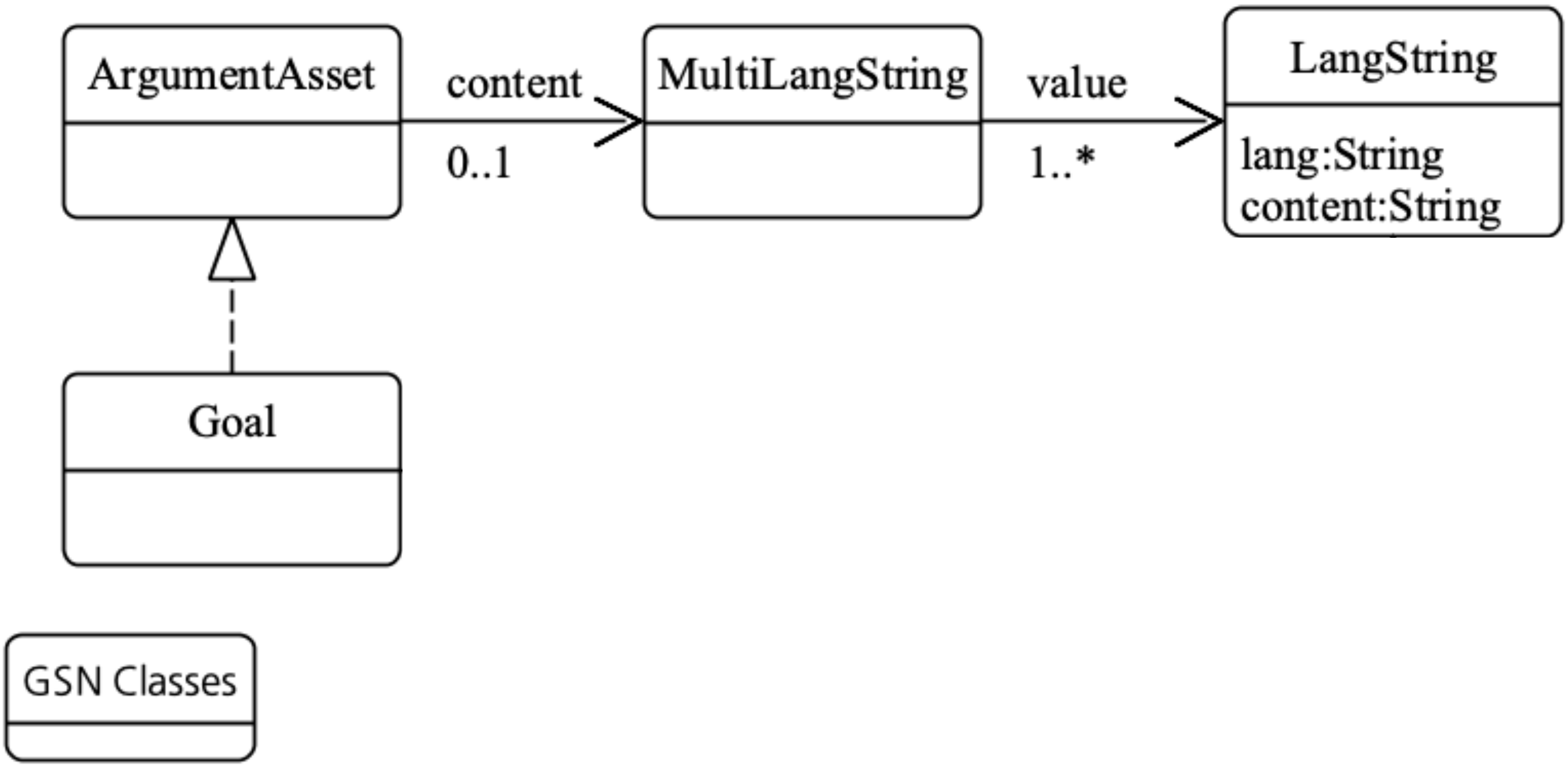



Pattern Instantiation

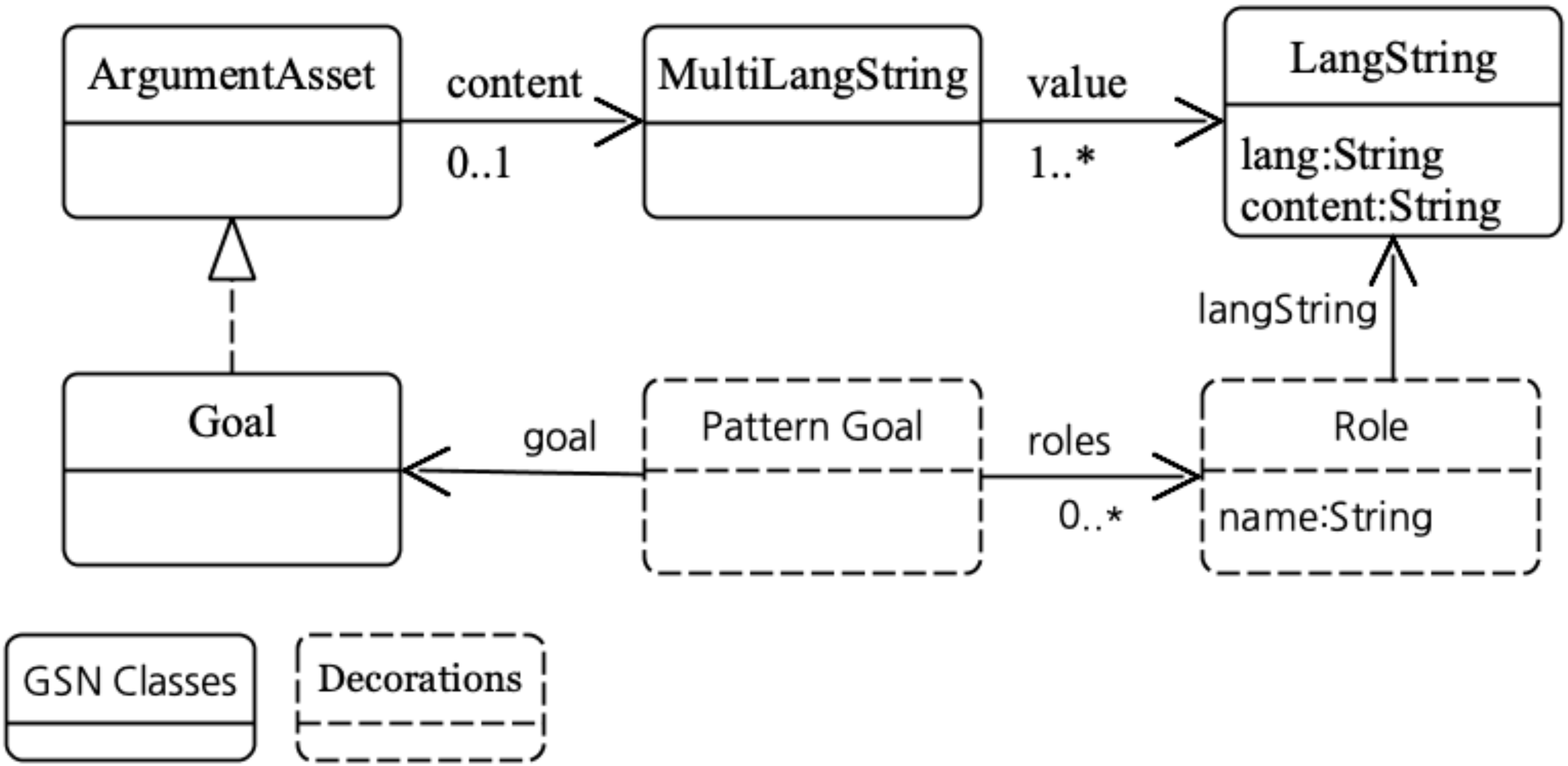
- Pattern elements are copied
- Roles replaced
- Multiplicities "unfolded"

{system} -> CarControl
{breakdownTable} -> ControlTable
#Subsystems -> 2
1. {ssystem} -> Throttle
2. {ssystem} -> Traction

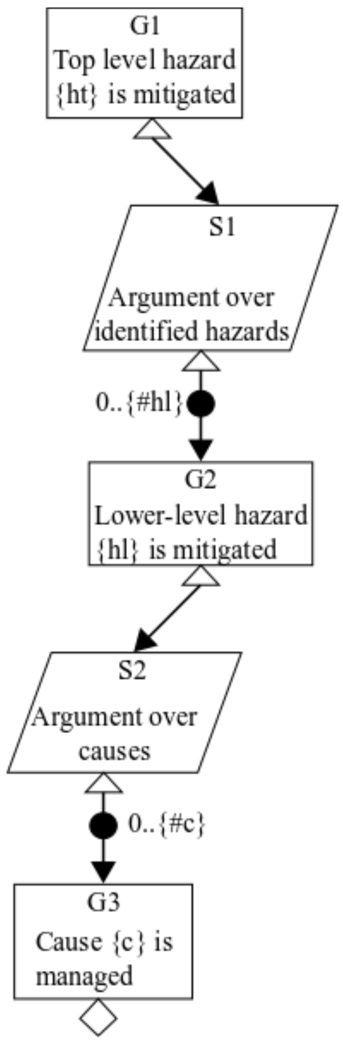
# Challenge 1: Capturing role metadata



# Challenge 1: Capturing role metadata

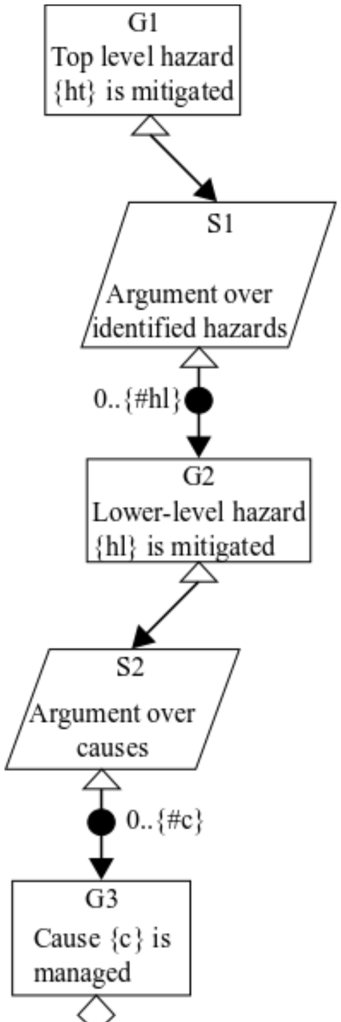


# Challenge 2: Capturing multiplicity metadata

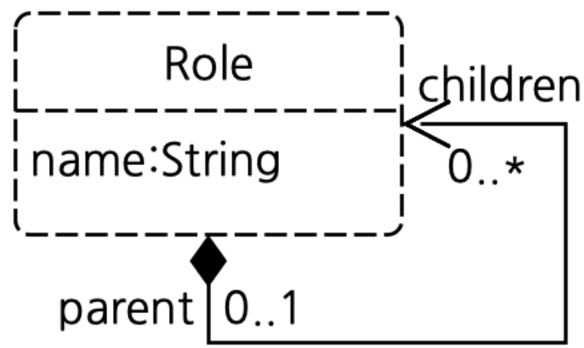



- Nested multiplicities cause combinatorial role values: for each **ht** there are many **hls**, and for each **hl** there are many **c** values.

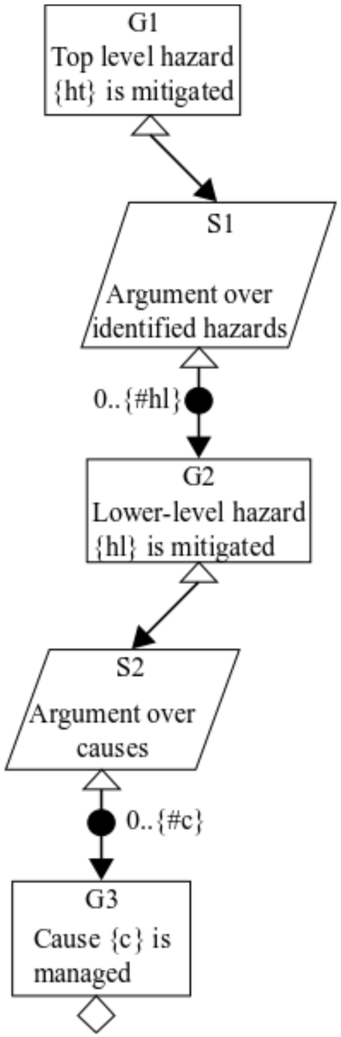
# Challenge 2: Capturing multiplicity metadata



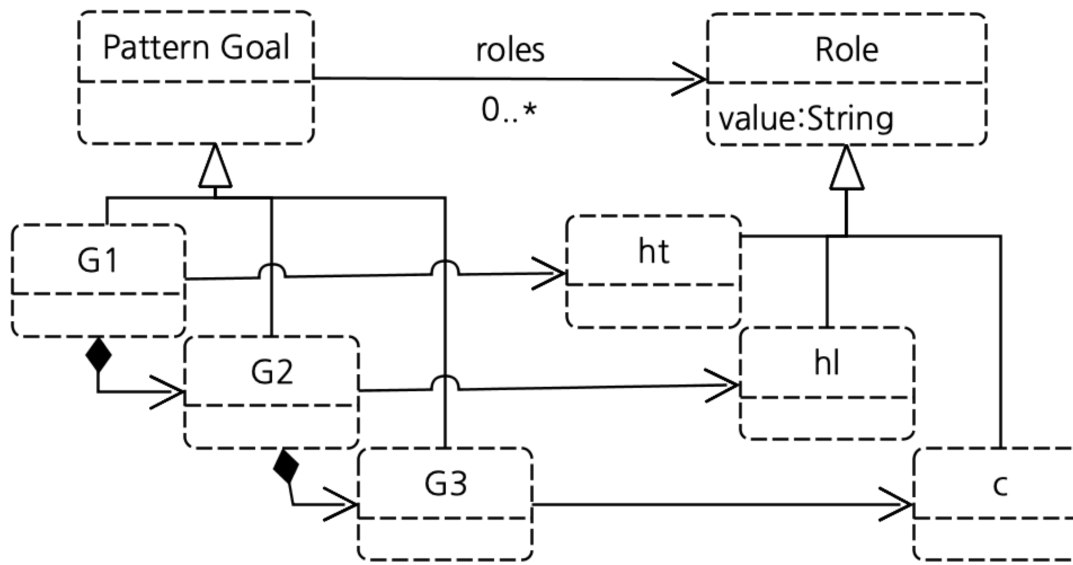

- Nested multiplicities cause combinatorial role values: for each **ht** there are many **hls**, and for each **hl** there are many **c** values.



# Challenge 2: Capturing multiplicity metadata




- Nested multiplicities cause combinatorial role values: for each **ht** there are many **hls**, and for each **hl** there are many **c** values.



# Challenges

- Complex metadata places additional requirements on the decoration language.
- Decorations can be required *per-model basis.* The decoration activity is much more time-consuming.
- A side effect of this fine-grained granularity is that reuse of decoration languages is reduced.
- In a nutshell, when metadata is tightly coupled to the semantics of model operations, a different approach to define more fine-grained decorations and model more complex relations is required.

# Generating decorators

Dealing with fine-grained decorations and complex metadata

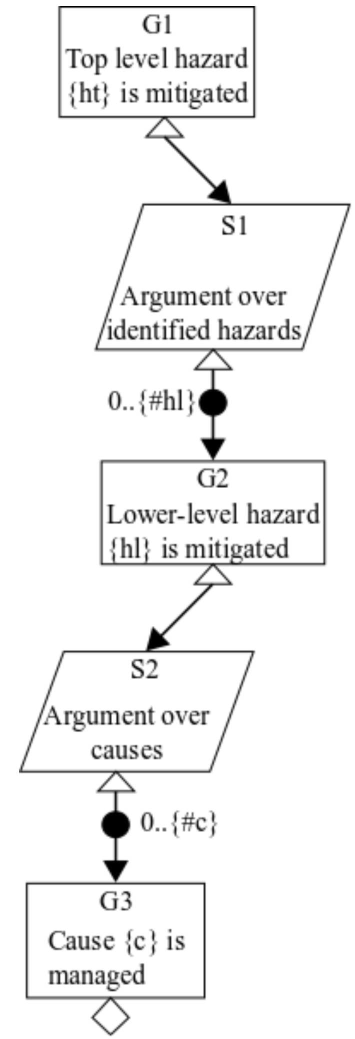
# GSN Observations

- Complex metadata is structured as a tree where
  - branches are related to the SupportedBy relations: multiplicity, optionality or selection,
  - each node can capture specific role information.
- Roles are often reused throughout the pattern.
- We call the *role:value* pairs a *link* and each node in the tree can have 0 or more links.

- Given that the nature of links is to capture text values, we opted for creating decoration languages that use textual notation.

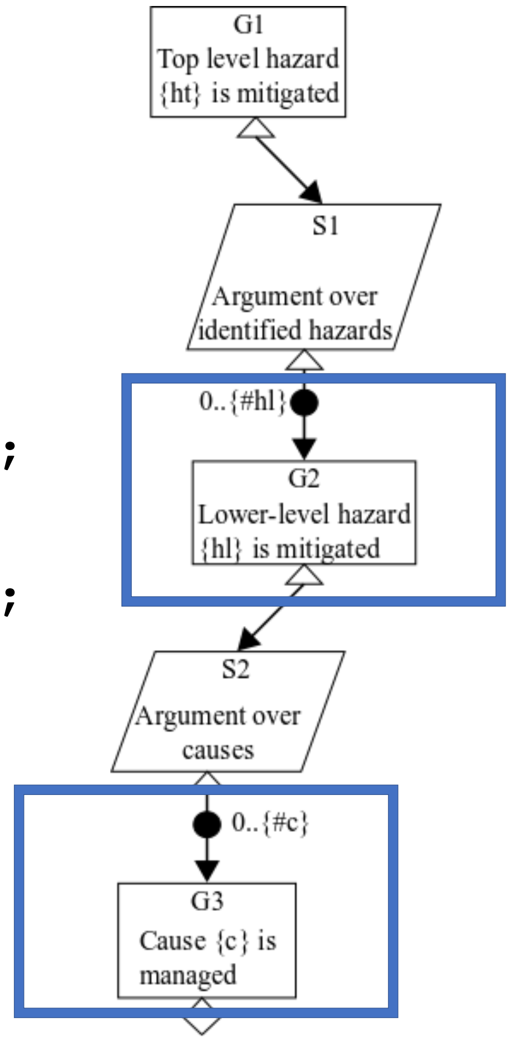
# GSN Decorator template (BNF)

```
data ::= variable* <gsnnode>*;
variable ::= ID '=' STRING;
<gsnnode> ::= (count <branchnode>) | <node>;
<branchnode> ::= '<name>_br' ':' <node>*;
<node> ::= '<node_name>'':'<link>* gsnnode*;
<link> ::= '*<role>' '=' ID | STRING;
count ::= <max> | (<min>,<max>)
```



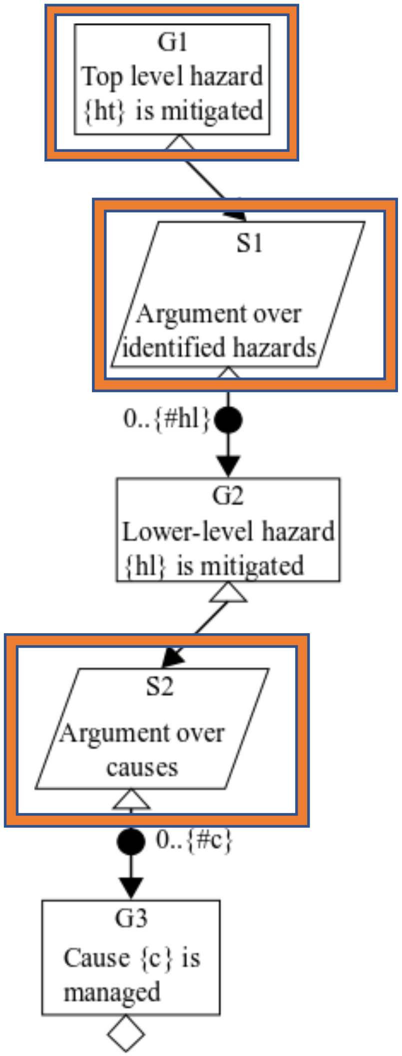
# GSN Decorator template (BNF)

```
data ::= variable* <gsnnode>*;
variable ::= ID '=' STRING;
<gsnnode> ::= (count <branchnode>) | <node>;
<branchnode> ::= '<name>_br' ':' <node>*;
<node> ::= '<node_name>'':'<link>* gsnnode*;
<link> ::= '*<role>' '=' ID | STRING;
count ::= <max> | (<min>,<max>)
```



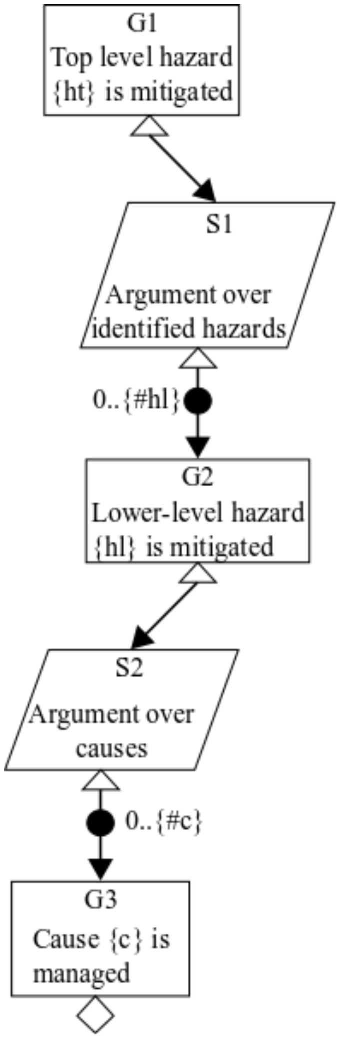
# GSN Decorator template (BNF)

```
data ::= variable* <gsnnode>*;
variable ::= ID '=' STRING;
<gsnnode> ::= (count <branchnode>) | <node>;
<branchnode> ::= '<name>_br' ':' <node>*;
<node> ::= '<node_name>'':'<link>* gsnnode*;
<link> ::= '*<role>' '=' ID | STRING;
count ::= <max> | (<min>,<max>)
```



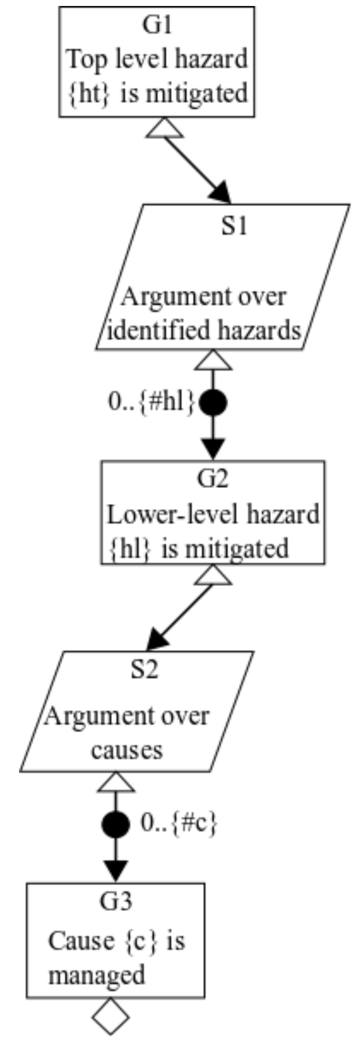
# GSN Decorator template (BNF)

```
data ::= 'G1' ':' g1 s1;
g1 ::= '*ht' '=' ID | STRING;
s1 ::= 'S1' ':' hl g2_br*;
g2_br ::= 'G2' ':' g2 s2;
hl ::= 'hl_count' '=' INT;
g2 ::= '*hl' '=' ID | STRING;
s2 ::= 'S2' ':' c g3_br*;
g3_br ::= 'G3' ':' g3;
c ::= 'c_count' '=' INT;
g3 ::= '*c' '=' ID | STRING;
```

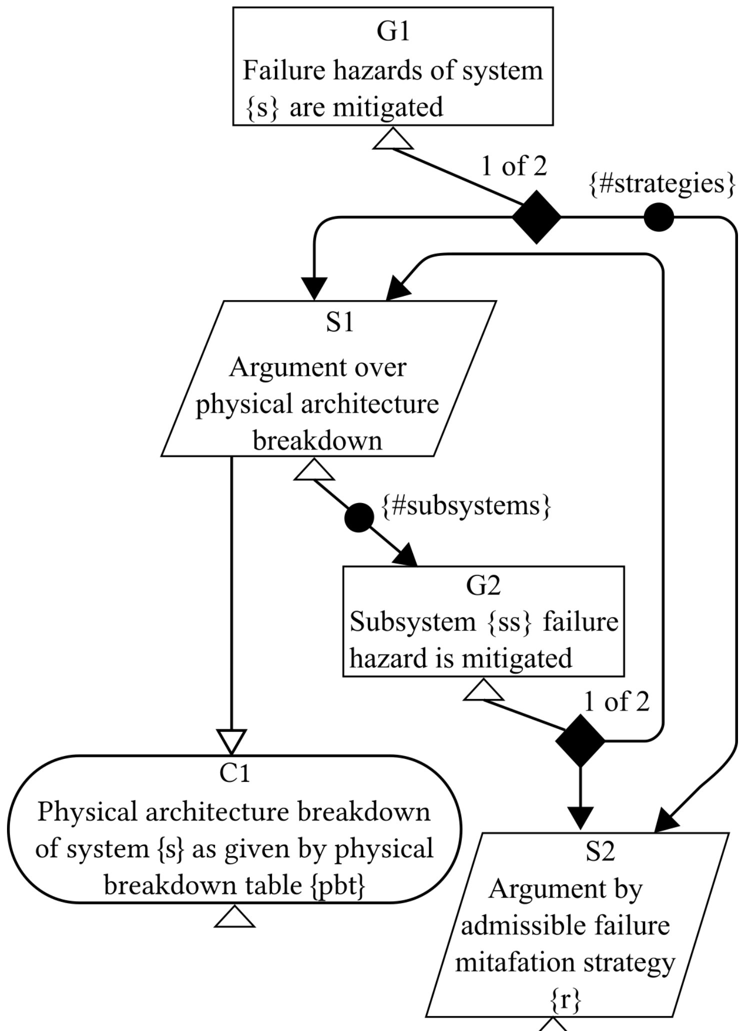


# GSN Decorator template (BNF)

```
G1:
 *ht = 'Hazard1'
 S1:
  hl_count = 2
  G2:
   *hl = 'LowHazard1'
   S2:
    c_count = 1
    G3:
     *c = 'LowHazard1 cause'
  G2:
   *hl = 'LowHazard2'
   S2:
    c_count = 2
    G3:
     *c = 'LowHazard2 cause1'
    G3:
     *c = 'LowHazard2 cause2'
```

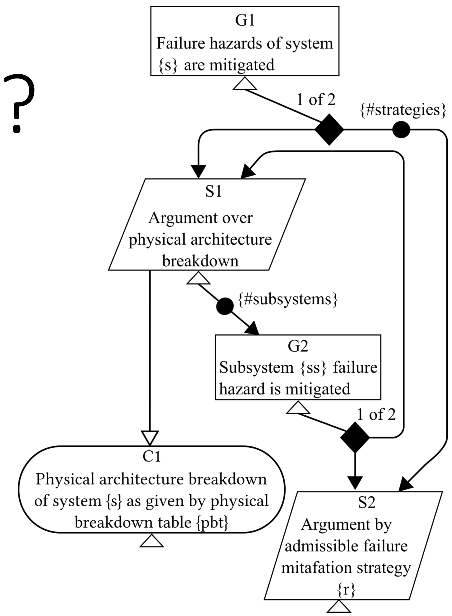


# What about more complex patterns?



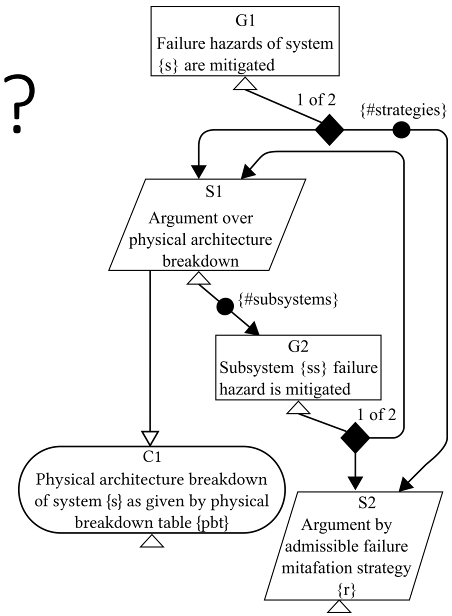
# What about more complex patterns?

```
data:
  (vars+=Var)*
  top=g1;
g1:
  'G1'':'
    BEGIN
    '*s' '=' s=(STRING | VarRef);
    g1s1=s1 | (count=s2_count g1s2+=s2_br*);
    END;
```



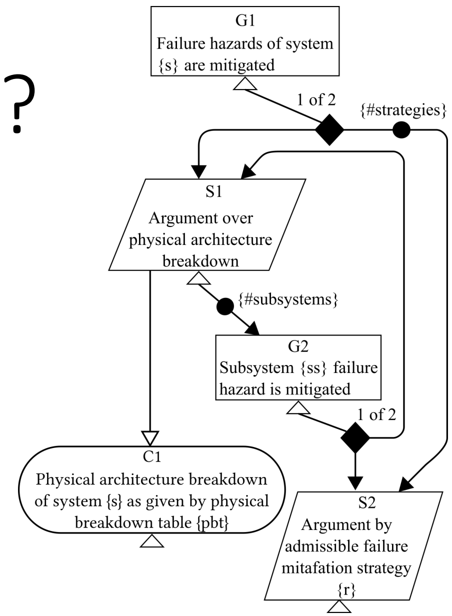
# What about more complex patterns?

```
s1:
  'S1:'
    BEGIN
    s1c1=c1 (count=g2_count s1g2+=g2_br)*;
    END;
c1:
  'C1'':'
    BEGIN
    '*s' '=' s=(STRING | VarRef);
    '*pbt' '=' pbt=(STRING | VarRef);
    END
g2_br:
  BEGIN
    top=g2 (g2s1=s1 | g2s2=s2);
  END
g2_count:
  'subsystems_count' '=' INT;
```

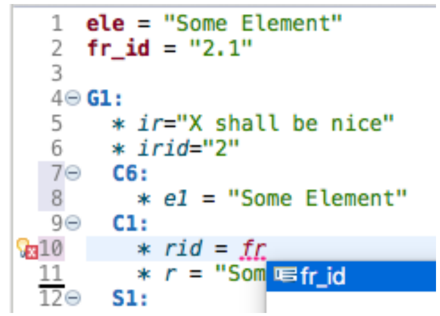


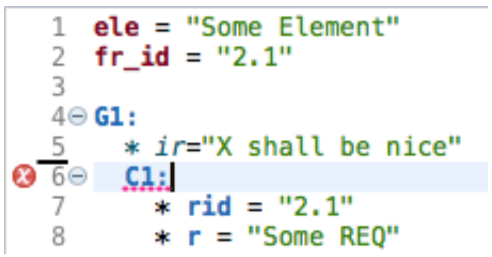
# What about more complex patterns?

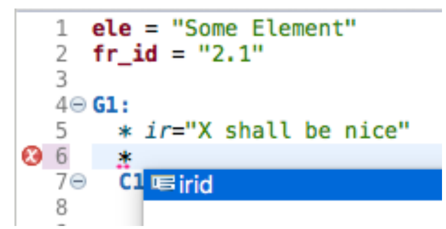
```
g2:
  'G2'':'
    BEGIN
    '*ss' '=' ss=(STRING | VarRef);
    END
s2_gb:
  top=s2
s2_count:
  'strategies_count' '=' INT;
s2:
  'S2'':'
    BEGIN
    '*r' '=' r=(STRING | VarRef);
    END
Var:
  name = ID '=' value=STRING;
VarRef:
  ref = [Var];
```

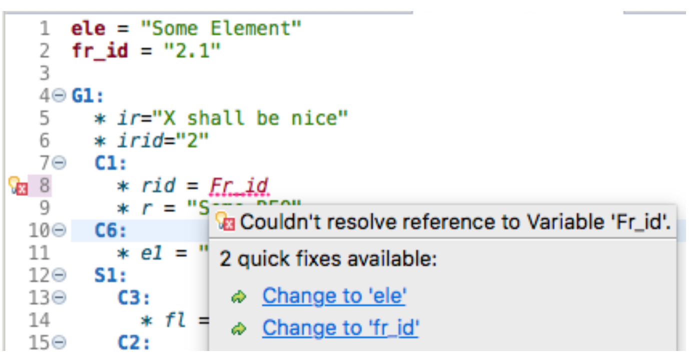


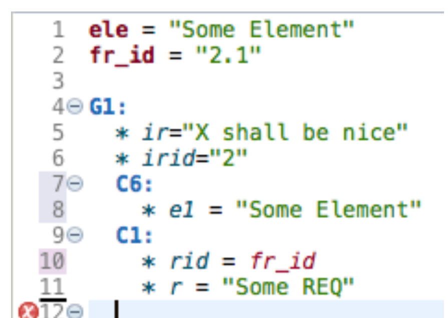
# The Editor


ele = "Some Element"
fr_id = "2.1"
G1:
* ir="X shall be nice"
* irid="2"
C6:
* el = "Some Element"
C1:
* rid = fr
* r = "Som
fr_id
S1:


ele = "Some Element"
fr_id = "2.1"
G1:
* ir="X shall be nice"
C1:
* rid = "2.1"
* r = "Some REQ"


ele = "Some Element"
fr_id = "2.1"
G1:
* ir="X shall be nice"
*
C1
irid


ele = "Some Element"
fr_id = "2.1"
G1:
* ir="X shall be nice"
* irid="2"
C1:
* rid = Fr_id
* r = "S
C6:
* el = "
S1:
C3:
* fl =
C2:
Couldn't resolve reference to Variable 'Fr_id'.
2 quick fixes available:
Change to 'ele'
Change to 'fr_id'


ele = "Some Element"
fr_id = "2.1"
G1:
* ir="X shall be nice"
* irid="2"
C6:
* el = "Some Element"
C1:
* rid = fr_id
* r = "Some REQ"

# Questions?

Acknowledgments

This work was supported by the SECT-AIR project, funded by the Aerospace Technology Institute and In- novate UK, as project number 113099 This work was partially supported by the EU through the DEIS project (#732242).