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

- Goal
- Strategy
- Solution
- Assumption
- Context
- Modifiers

• 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 $\text{ht}$ there are many $\text{hls}$, and for each $\text{hl}$ there are many $\text{c}$ 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 \textbf{ht} there are many \textbf{hls}, and for each \textbf{hl} there are many \textbf{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.
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?

- **G1**
  - Failure hazards of system \( \{s\} \) are mitigated

- **S1**
  - Argument over physical architecture breakdown

- **G2**
  - Subsystem \( \{ss\} \) failure hazard is mitigated

- **C1**
  - Physical architecture breakdown of system \( \{s\} \) as given by physical breakdown table \( \{pbt\} \)

- **S2**
  - Argument by admissible failure mitigation strategy \( \{r\} \)
What about more complex patterns?

data:

\[(vars+=Var)\]*

top=g1;

g1:

'G1' :'

BEGIN

'*s' '=' s=(STRING | VarRef);

gls1=s1 | (count=s2_count gls2+=s2_br*);

END;
What about more complex patterns?

s1:
'\texttt{S1}:'
BEGIN
s1c1=c1 (count=g2\_count s1g2+=g2\_br)*;
END;

c1:
'\texttt{C1}:'
BEGIN
'\texttt{*s} ' '=' s=(\text{STRING } | \text{VarRef});
'\texttt{*pbt} ' '=' pbt=(\text{STRING } | \text{VarRef});
END

g2\_br:
BEGIN
\text{top}=g2 (g2s1=s1 | g2s2=s2);
END

g2\_count:
'\texttt{subsystems\_count}' '=' INT;
What about more complex patterns?

\[
g2:\quad \text{'}G2'\text{' :} \\
\begin{align*}
\text{BEGIN} \\
\text{'*ss' } &= \text{ ss=(STRING | VarRef);} \\
\text{END}
\end{align*}
\]

\[
s2_gb: \\
\quad \text{top=s2}
\]

\[
s2_count: \\
\quad \text{'}strategies_count' } &= \text{ INT;}
\]

\[
s2: \\
\quad \text{'}S2'\text{' :} \\
\begin{align*}
\text{BEGIN} \\
\text{'*r' } &= \text{ r=(STRING | VarRef);} \\
\text{END}
\end{align*}
\]

\[
\text{Var:} \\
\quad \text{name = ID } &= \text{ value=STRING;}
\]

\[
\text{VarRef:} \\
\quad \text{ref = [Var];}
\]
The Editor

1. \( \text{ele} = "\text{Some Element}" \)
2. \( \text{fr}_\text{id} = "2.1" \)
3. \( \text{G1:} \)
4. * \( \text{ir} = "X \text{ shall be nice}" \)
5. * \( \text{irid} = 2 \)
6. \( \text{C6:} \)
7. * \( \text{el} = "\text{Some Element}" \)
8. \( \text{C1:} \)
9. * \( \text{rid} = \text{fr}_\text{id} \)
10. * \( r = "\text{Some REQ}" \)
Questions?

Acknowledgments

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