Heterogeneous Megamodel Slicing for Model Evolution

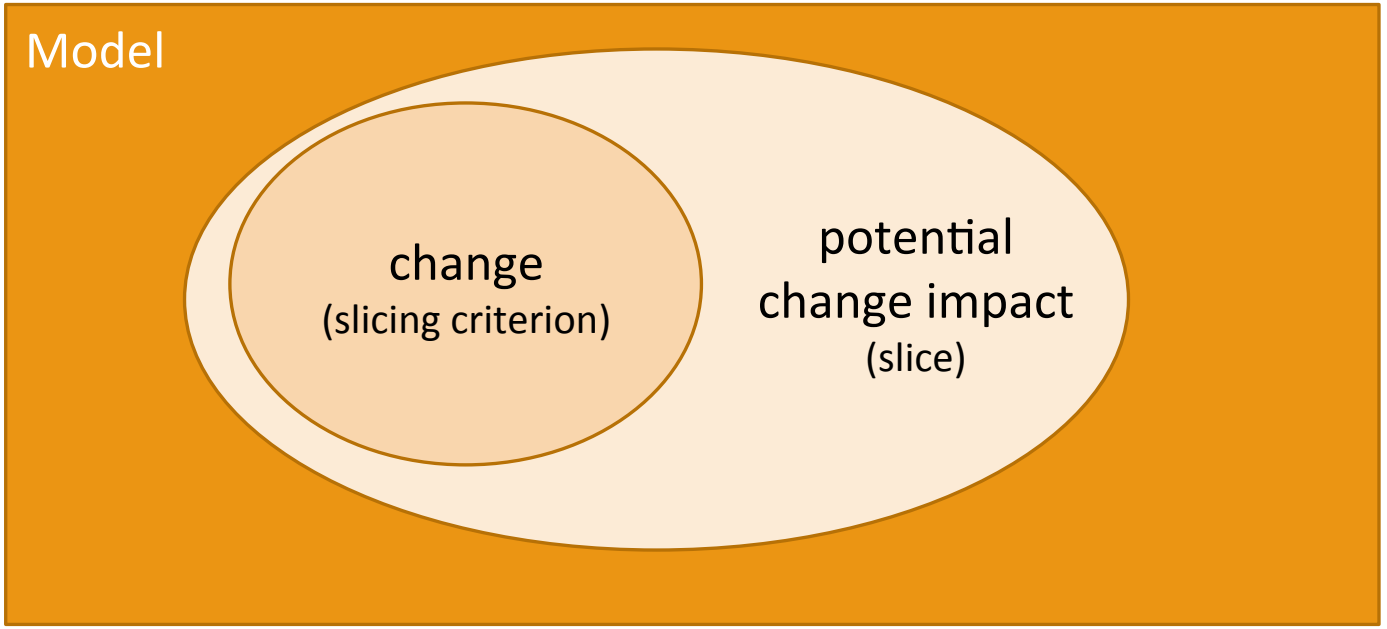
RICK SALAY, SAHAR KOKALY, MARSHA CHECHIK AND TOM MAIBAUM

ME 2016 @ MODELS'16

OCT. 2, 2016, SAINT MALO, FRANCE

Motivation

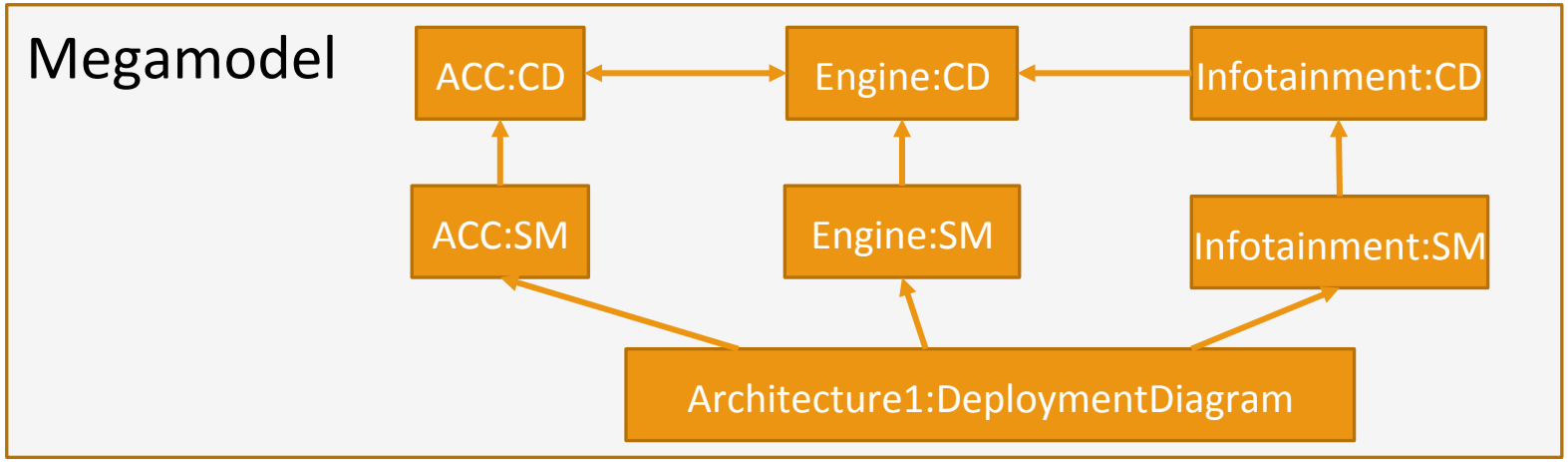
Model slicing to identify change impact is a key technique for supporting model evolution



Motivation

Slicing is well studied for individual models ..

.. but not for heterogeneous collections of related models (megamodels) which are common in large projects



(deep) megamodel slicing can be useful for identifying impact due to evolution across multiple models

A Model Management Approach for Assurance Case Reuse due to System Evolution

Sahar Kokaly
McMaster Centre for Software Certification
McMaster University
Hamilton, Canada
kokalys@mcmaster.ca

Rick Salay
Department of Computer Science
University of Toronto
Toronto, Canada
rsalay@cs.toronto.edu

Valentin Cassano
McMaster Centre for Software Certification
McMaster University
Hamilton, Canada
cassanv@mcmaster.ca

Tom Maibaum
McMaster Centre for Software Certification
McMaster University
Hamilton, Canada
maibaum@mcmaster.ca

Marsha Chechik
Department of Computer Science
University of Toronto
Toronto, Canada
chechik@cs.toronto.edu

Friday, Oct 7 in the "Testing and Analysis" Session

ABSTRACT

Evolution in software systems is a necessary activity that occurs due to fixing bugs, adding functionality or improving system quality. Systems often need to be shown to comply with regulatory standards. Along with demonstrating compliance, an artifact, called an assurance case, is often produced to show that the system indeed satisfies the property imposed by the standard (e.g., safety, privacy, security, etc.). Since each of the system, the standard, and the assurance case can be presented as a model, we propose the extension and use of traditional model management operators to aid in the reuse of parts of the assurance case when the system undergoes an evolution. Specifically, we present a model management approach that eventually produces a partial evolved assurance case and guidelines to help the assurance engineer in completing it. We demonstrate how our approach works on an automotive subsystem regulated by the ISO 26262 standard.

Keywords

Evolution, reuse, model management, regulatory compliance, assurance cases, certification.

1. INTRODUCTION

The pervasiveness of software in all aspects of human activity has created special concerns regarding issues such as safety, security and privacy. Governments and standard organizations (e.g., ISO) have responded to this trend by creating regulations and standards that software must comply with. Compliance is a complex and costly goal to achieve. They may have to comply with multiple standards due to multiple jurisdictions and track the changes in standards. *Assurance cases* – collections of arguments and evidence to support the claims of compliance – must be developed and managed. Finally, maintaining families of related software products further multiplies the effort. Increasingly, models and model-driven engineering are being used as means to facilitate communication and collaboration between the stakeholders in the compliance value chain and further to introduce automation into regulatory compliance tasks.

In a position paper [21], we a laid out a research agenda for applying model management to address the software compliance problem and sketched its use in particular compliance management scenarios. In this paper, we focus on one of these scenarios – assurance case reuse due to system evolution – and develop it in detail. Fig. 1 illustrates the scenario at a high level. Assume that a current specification S describes the specification for the software in a vehicle. In addition, a type of assurance case A, called a *safety* case, has been developed complying with the ISO 26262 vehicle functional safety standard [16]. Safety case A contains perhaps thousands of safety claims about different components of the vehicle, as well as arguments and evidence to support these claims. Now if S is evolved to S' – for example, as a result of a new requirement or a bug fix – a corresponding safety case A' for S' must be developed. Due to complexity and effort required to develop a safety case, there is strong incentive to reuse as much of A as possible in the creation of A'. We address this problem using a model management strategy. Specifically, we make the following contributions:

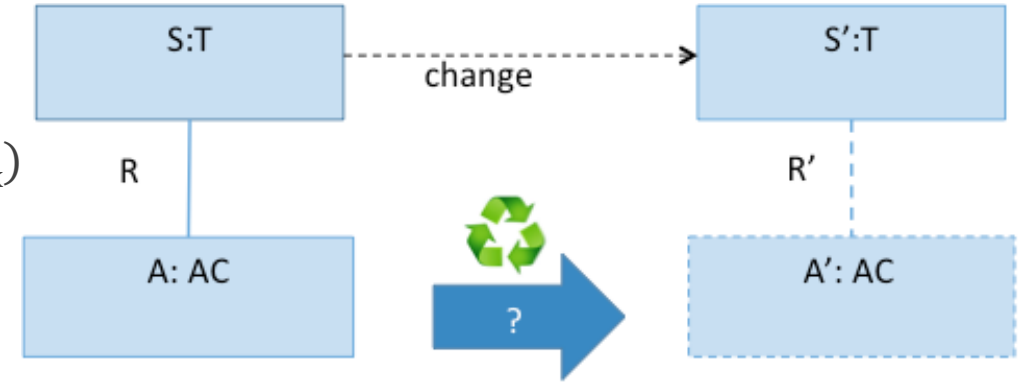
1. We define a generic model management framework for

Assurance Case Impact Assessment algorithm [MODELS'16]

Friday, Oct 7 in the "Testing and Analysis" Session

Params: <$Slice_T$; $Merge_T$>
Input: initial spec S : T, assurance case A : AC, traceability map R, changed spec S' : T, delta D = <C0a;C0d;C0c>
Output: Impact set estimate A_{RMM}, impact kind annotation k_{RMM}

1: $R'_A \leftarrow Restrict(R, D)$
2: $dc \leftarrow Slice_T(S, Merge_T(d,c))$
3: $ac \leftarrow Slice_T(S', Merge_T(a,c))$
4: $C2_{recheck} \leftarrow Merge_{AC}(Trace(R, dc), Trace(R'_A, ac))$
5: $C2_{revise} \leftarrow Trace(R, d)$
6: $C3_{revise} \leftarrow Slice_{AC}(M, C2_{revise})$
7: $C3_{recheck} \leftarrow Slice_{AC}(M, C2_{recheck})$
8: $A_{RMM} \leftarrow Merge_{AC}(C3_{revise}, C3_{recheck})$
9: $k_{RMM}(C3_{recheck}) \leftarrow$ 'recheck'
10: $k_{RMM}(C3_{revise}) \leftarrow$ 'revise'
11: return A_{RMM}, k_{RMM}



Proposed Slicing Algorithm

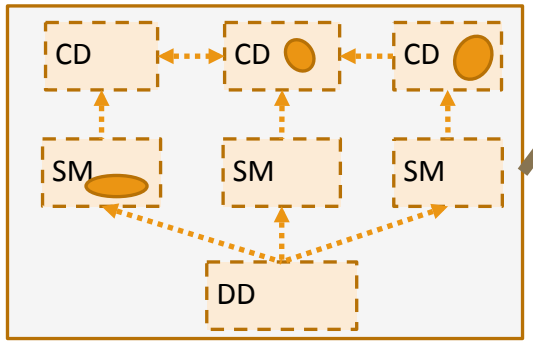
- ✓Operates on Megamodels: general modeling technique to represent collections of interrelated models
- ✓Works with arbitrary model types (e.g., conceptual, behavioural, goal models, test models, etc.)
- ✓ Uses widely adopted notion of traceability relations to assess change impact between models.

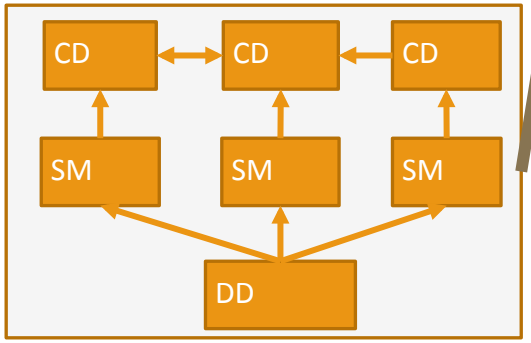
Assumptions:

1. (Slicers) There is a slicer available for each model type represented in the megamodel
2. (Dependencies) The relationships express all and only the inter-model dependencies

Slicing algorithm

criterion megamodel fragment



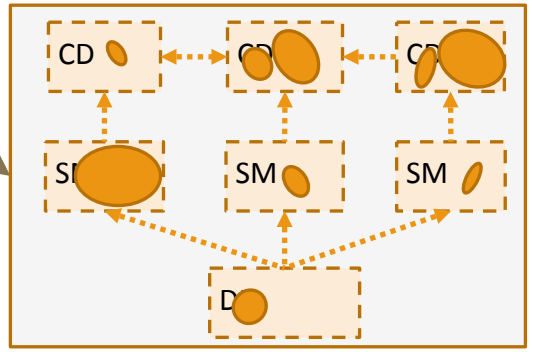


megamodel

Algorithm: Forward Megamodel Slice
Input: megamodel X, criterion megamodel fragment $S_c[X]$
Output: slice megamodel fragment $S[X]$

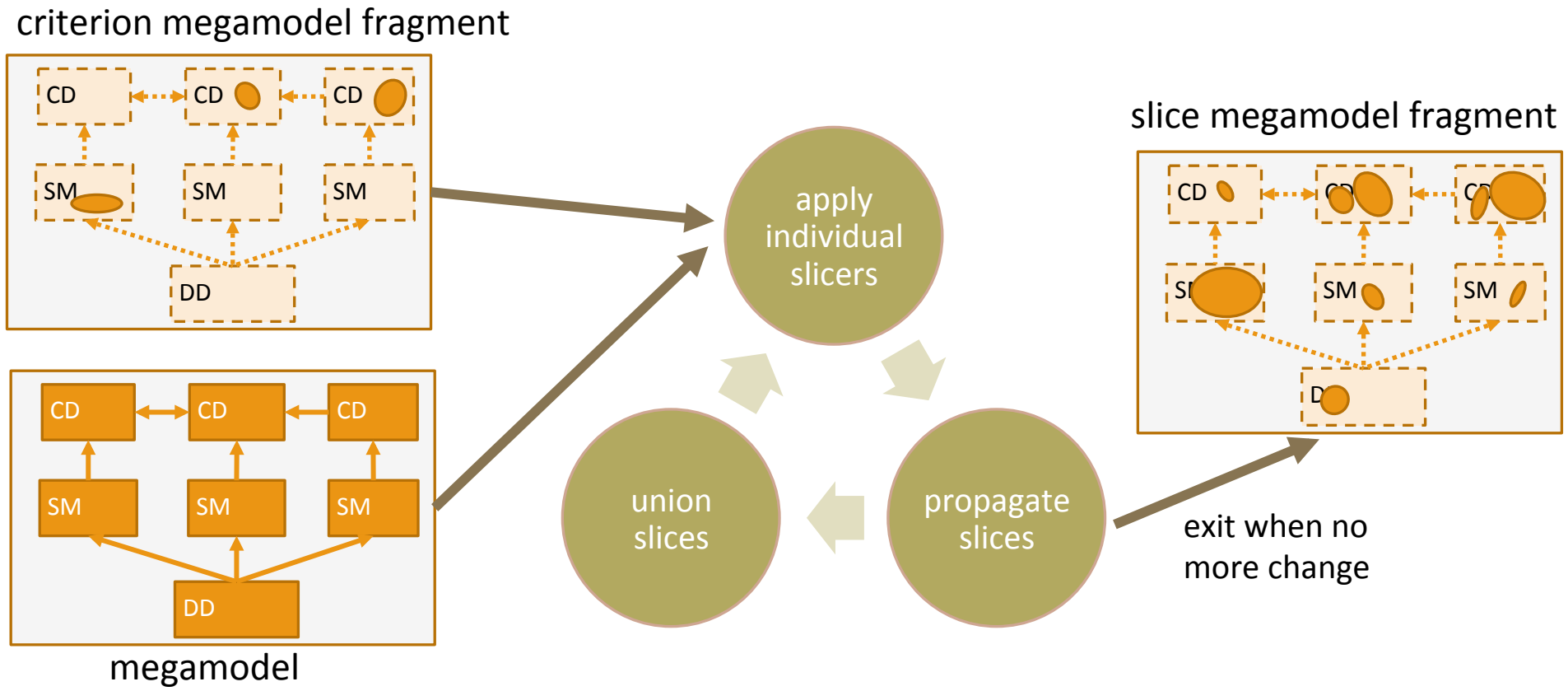
```
1:  S[X] := S_c[X]
2:  do {
3:      S'[X] := S[X]
4:      S_1[X] := ∅
5:      for (S[M] ∈ S[X]) {
6:          T := M.type
7:          S_1[M] := Slice_T(M, S[M])
8:          S_1[X] := Union(S_1[X], {S_1[M]})
9:      }
10:     S_2[X] := ∅
11:     for (S_1[M] ∈ S_1[X]) {
12:         for (R ∈ M.end) {
13:             M' := OppEnd(R, M)
14:             S_2[M'] := Trace(R, S_1[M])
15:             S_2[X] := Union(S_2[X], {S_2[M']})
16:         }
17:     }
18:     S[X] := Union(S_1[X], S_2[X])
19: } until (S[X] ⊑ S'[X])
20: return S[X]
```

slice megamodel fragment

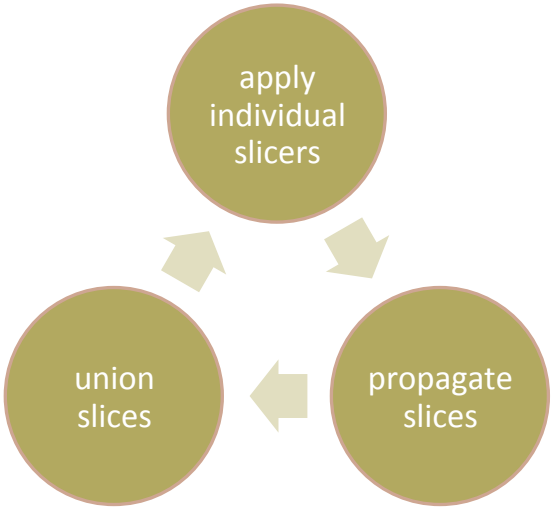


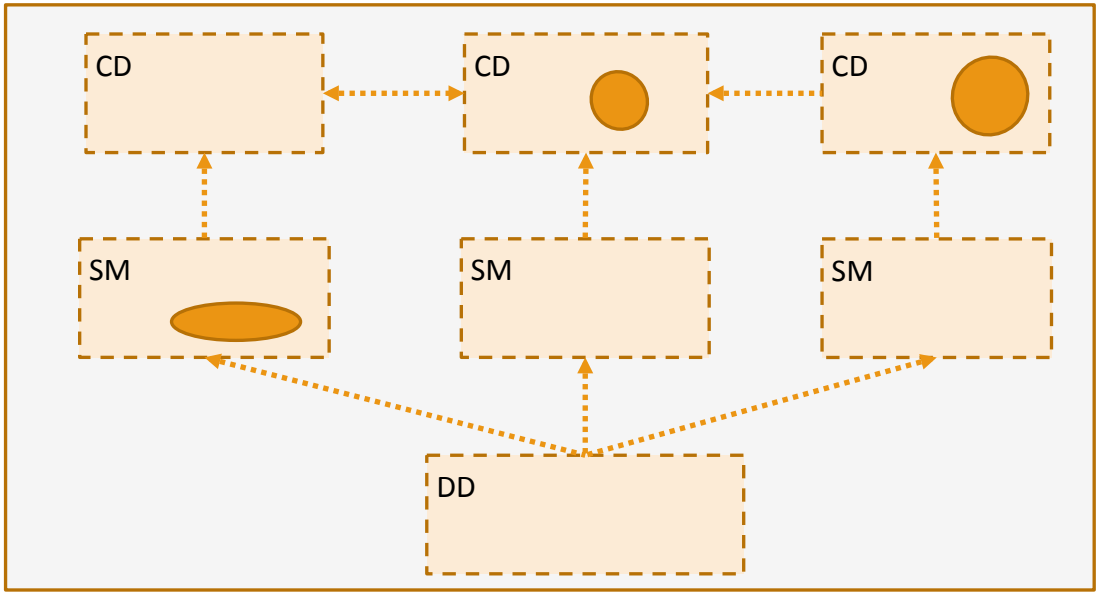
a slicer for each model type is assumed to be available

Slicing algorithm

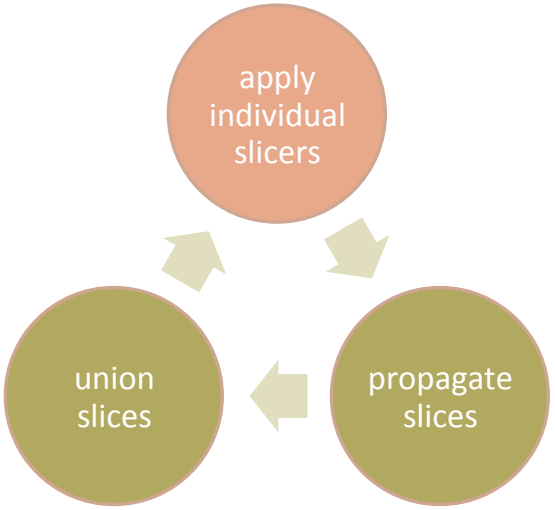


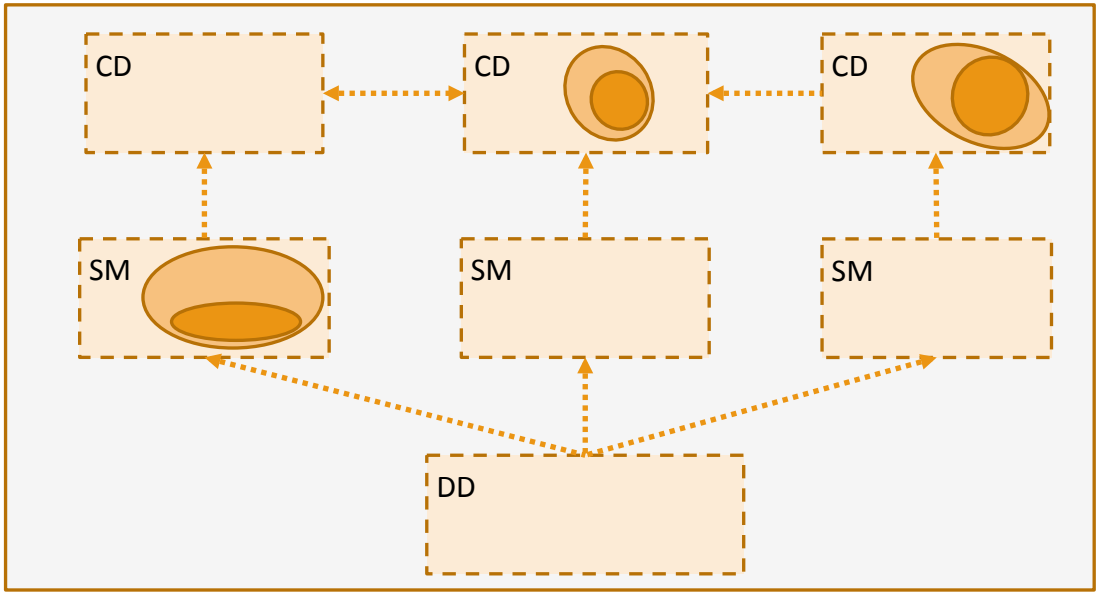
Example run: slicing criterion



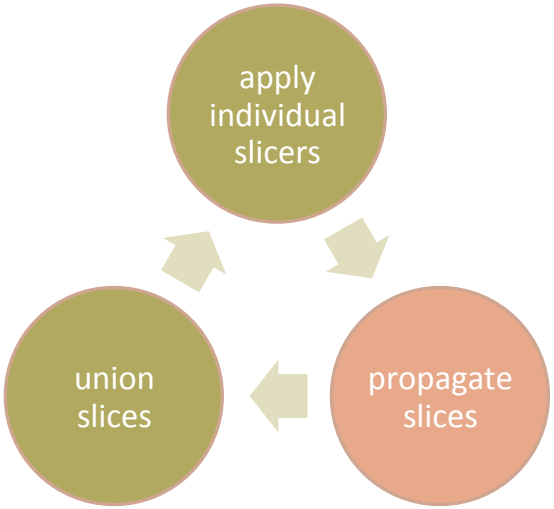


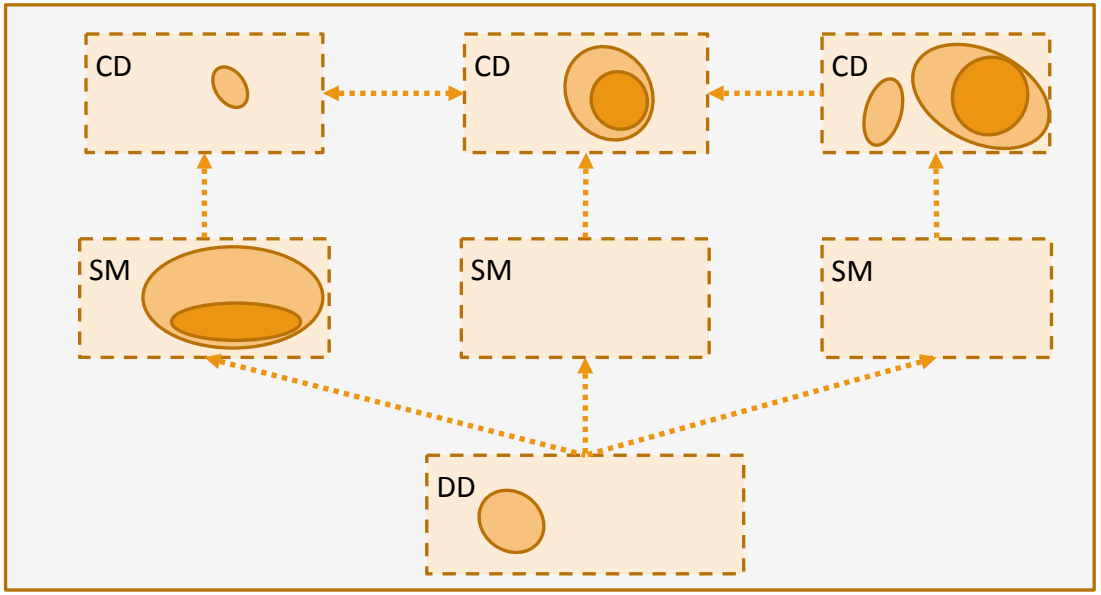
Example run: 1st iteration



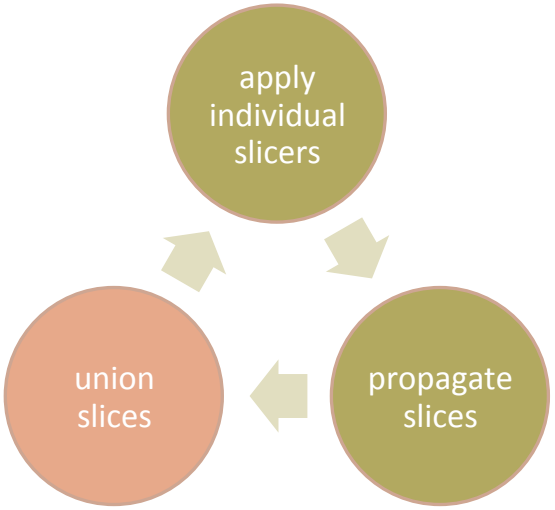


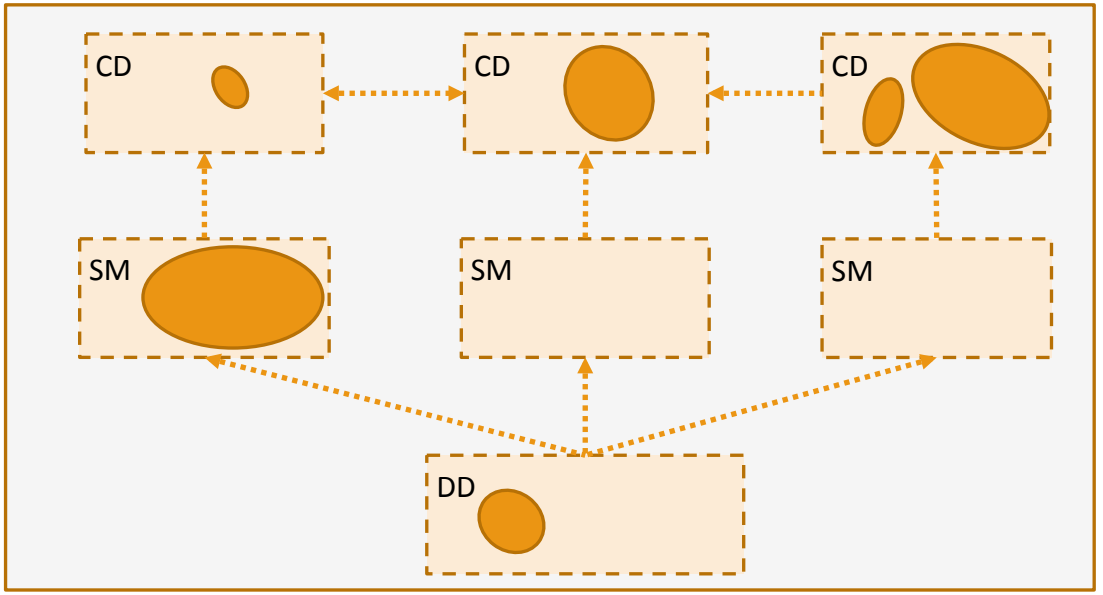
Example run: 1st iteration



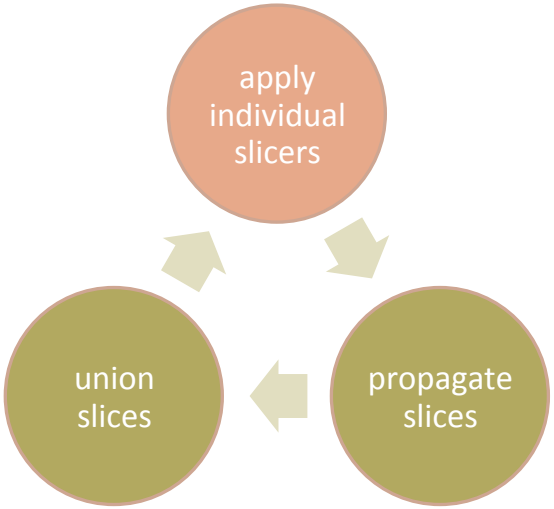


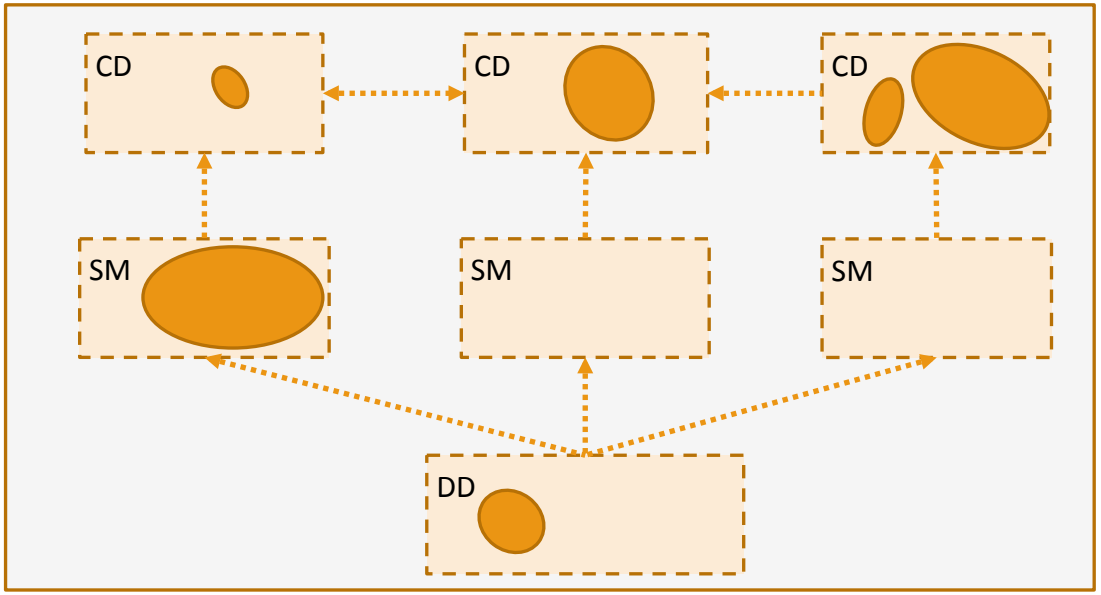
Example run: 1st iteration



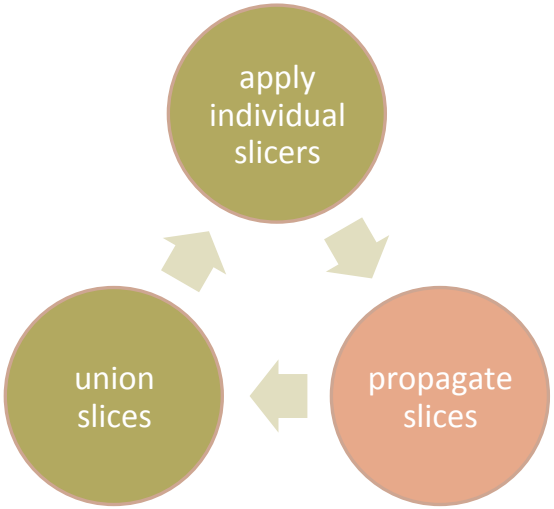


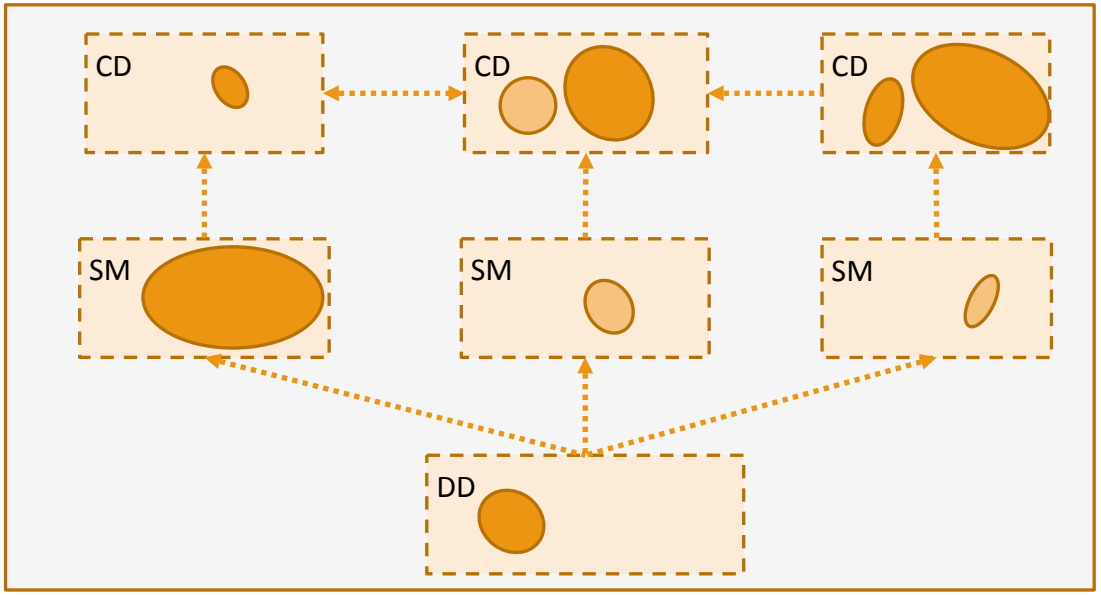
Example run: 2nd iteration



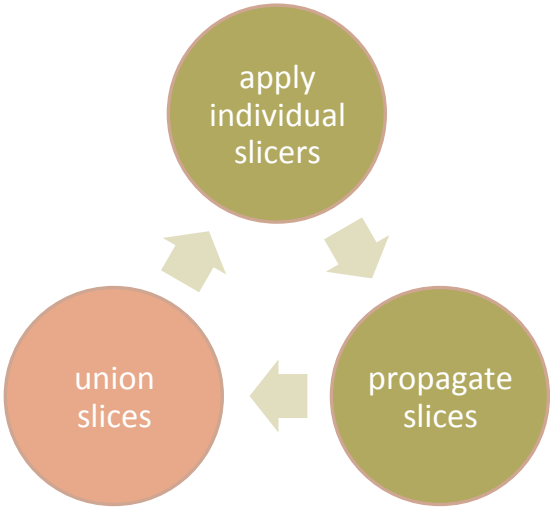


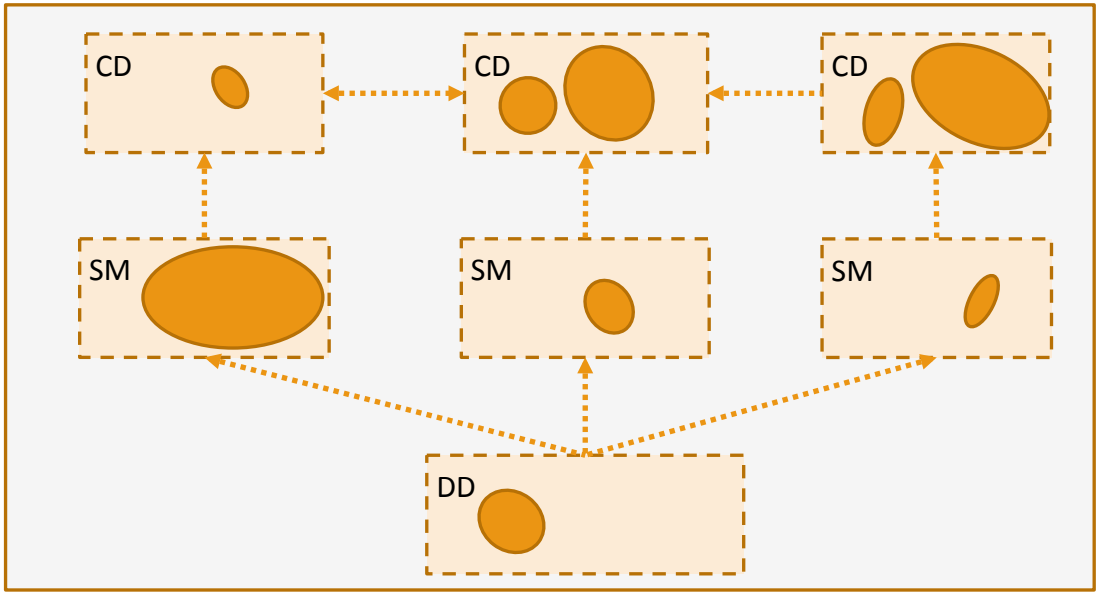
Example run: 2nd iteration



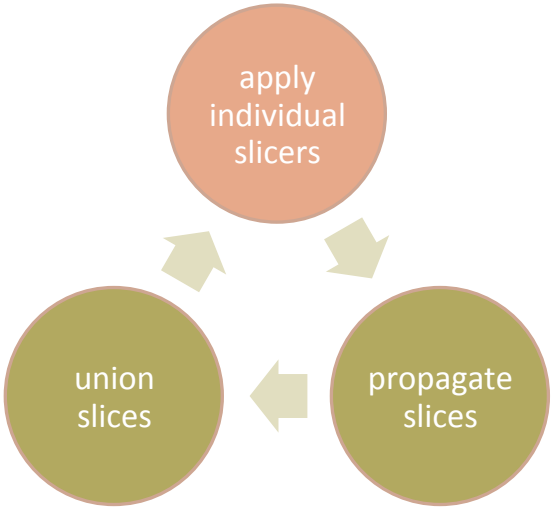


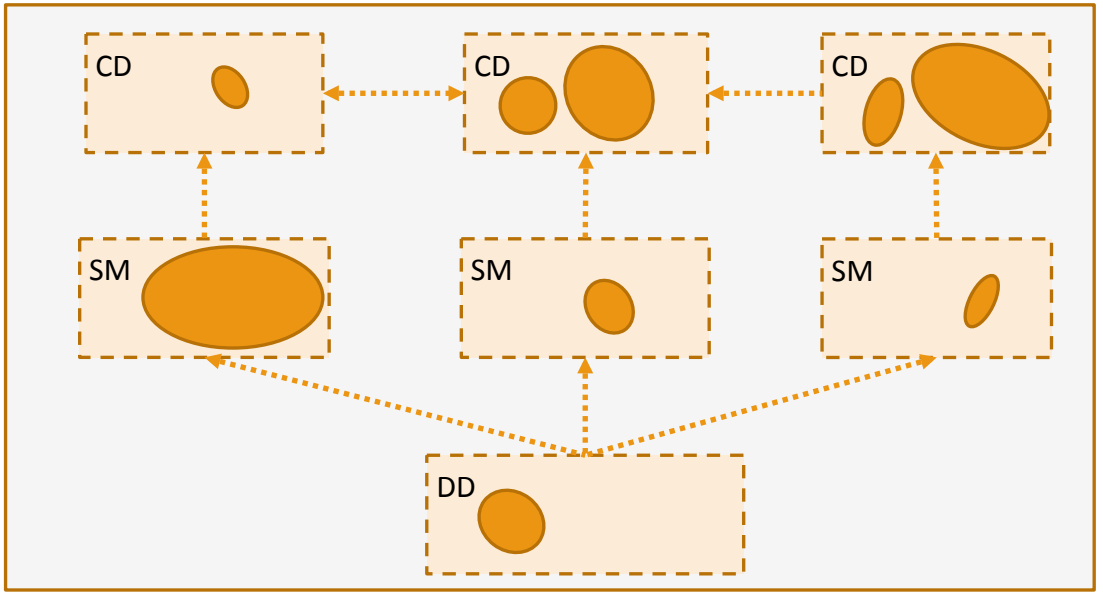
Example run: 2nd iteration



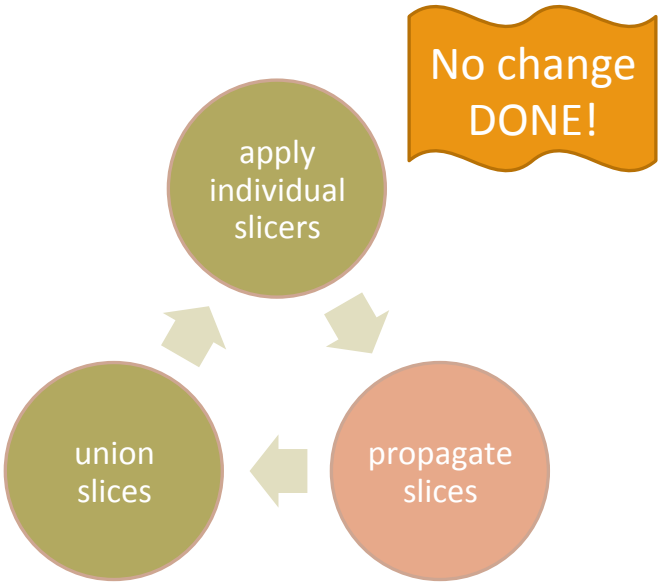


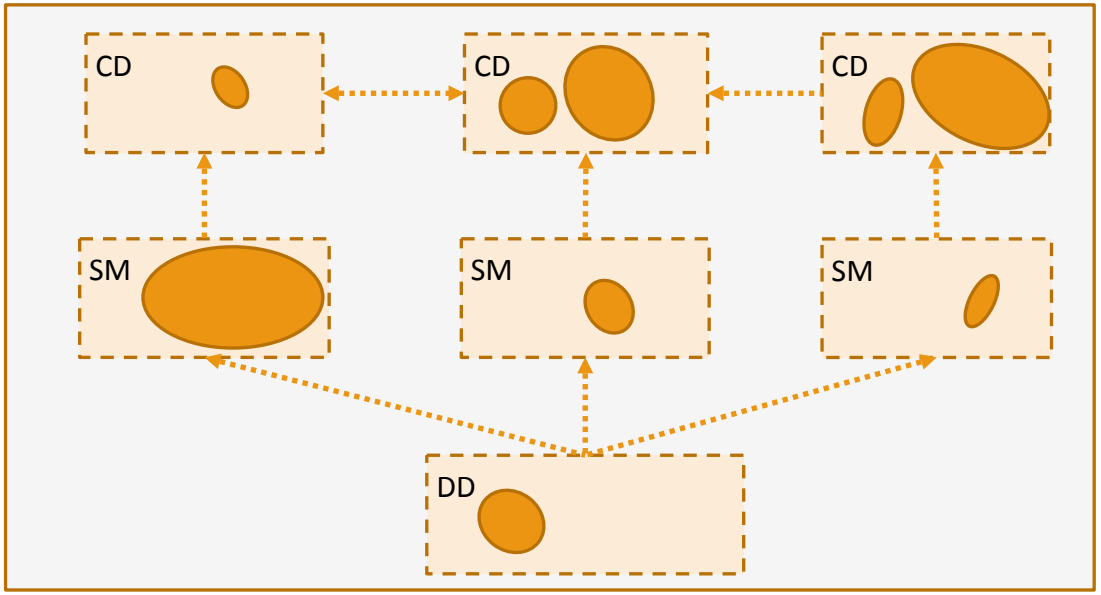
Example run: 3rd iteration





Example run: 3rd iteration





Post processing

The algorithm focuses only on propagating dependencies

- ..and requires each slicer to do the same

However, the slicing algorithm should produce an output having additional characteristics:

- (Well-formedness) Should consist of well-formed slice models
- (Referential integrity) Should contain enough information to resolve references between related models

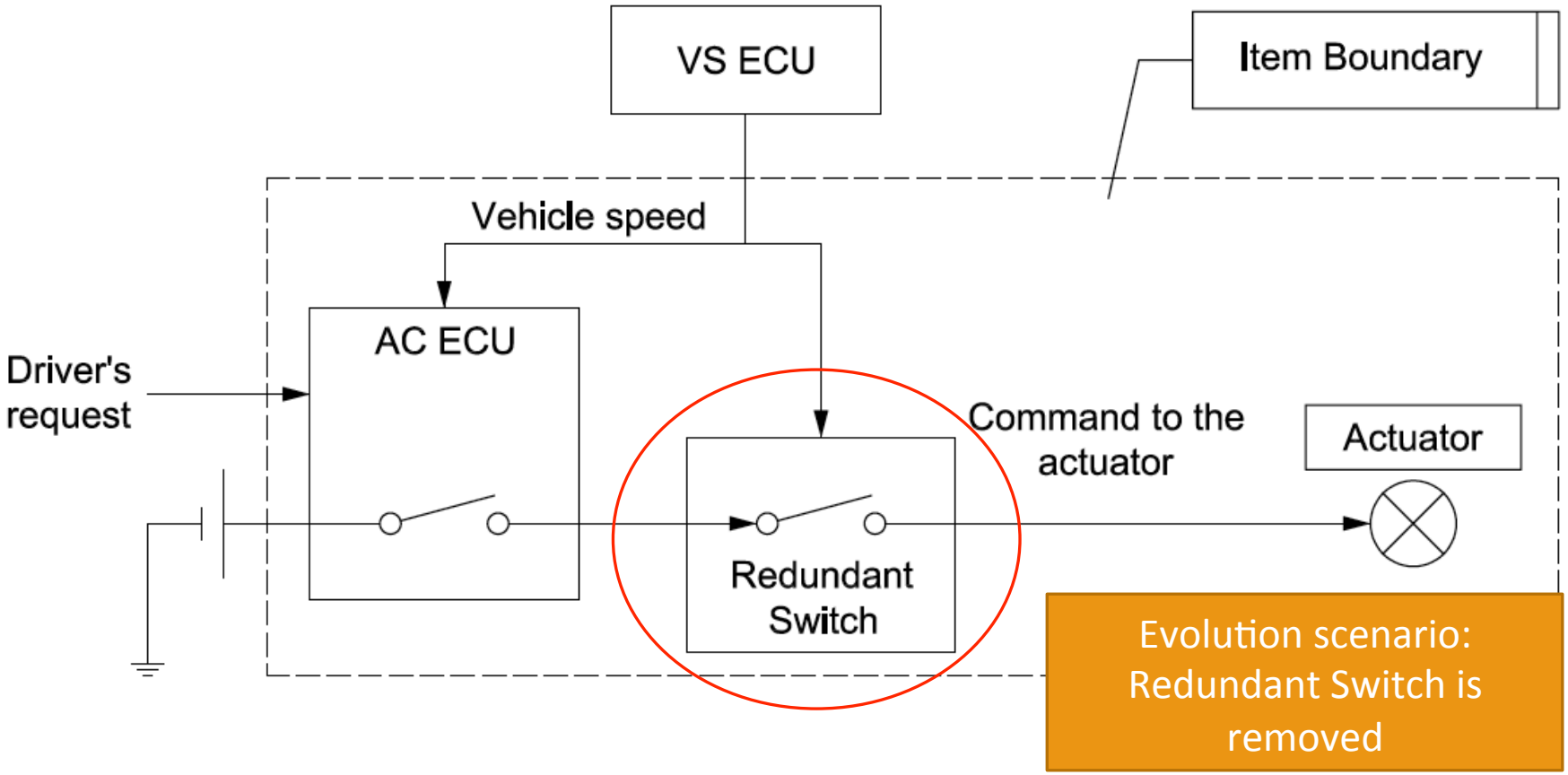
These are obtained by expanding the algorithm output in a post-processing step.

Example: Power Sliding Door

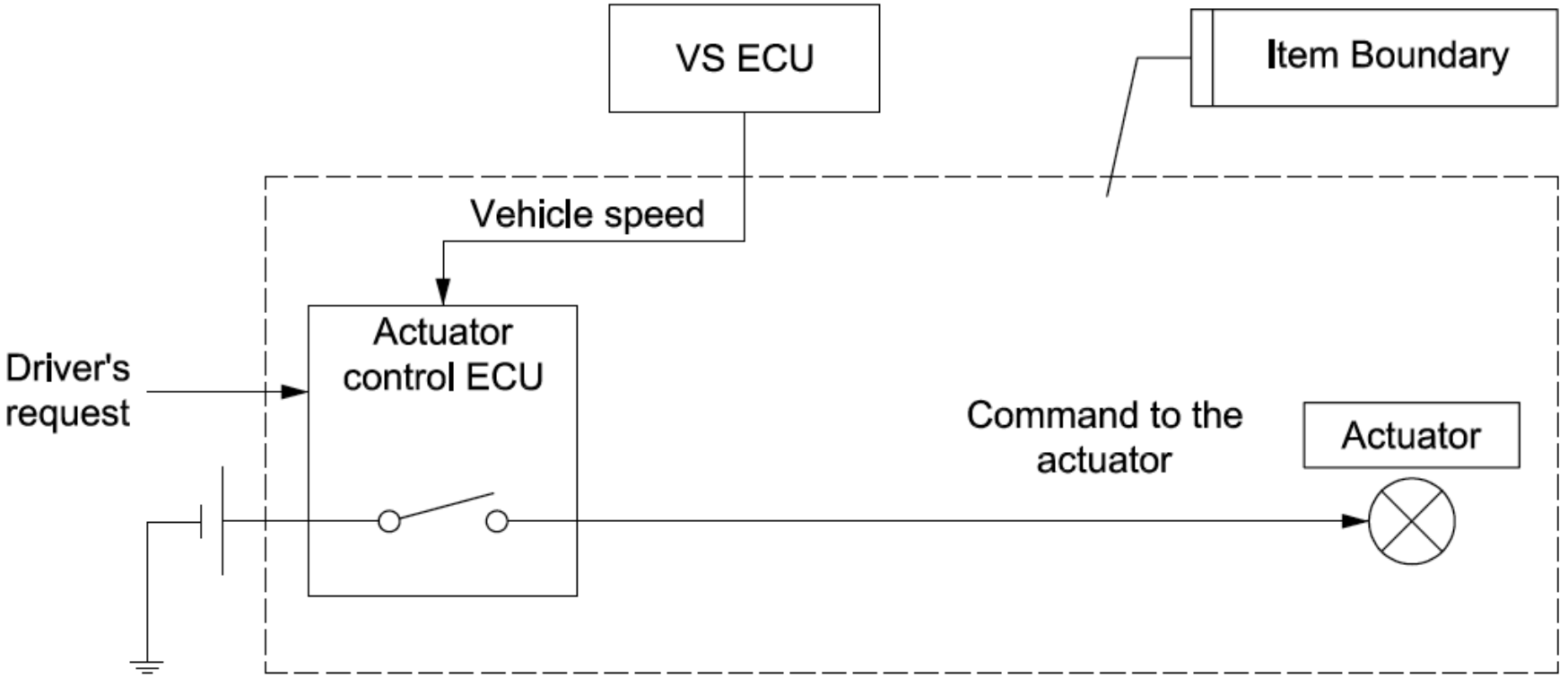
Automotive subsystem that controls the behaviour of a power sliding door in a car.

- This example is presented in Part 10 of the ISO 26262 standard

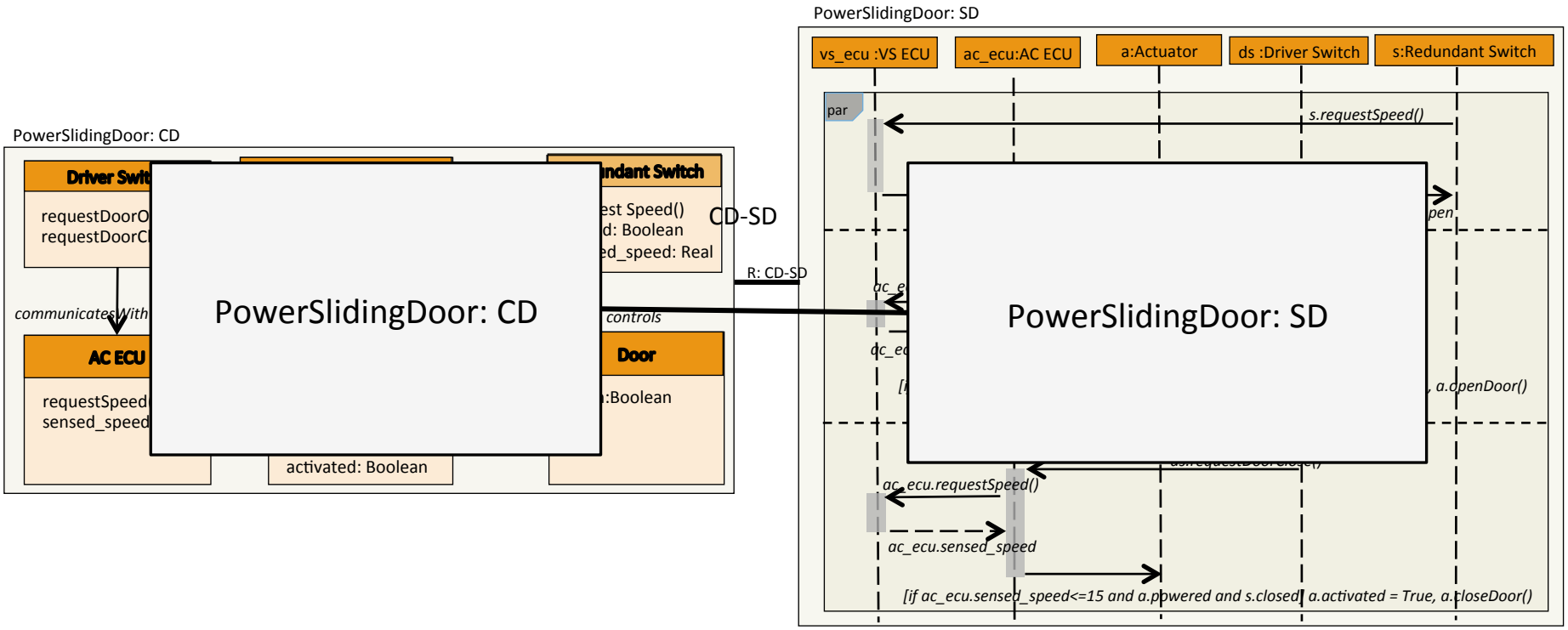
Power sliding door architecture (before evolution)



Power door slider architecture (after evolution)



Initial megamodel

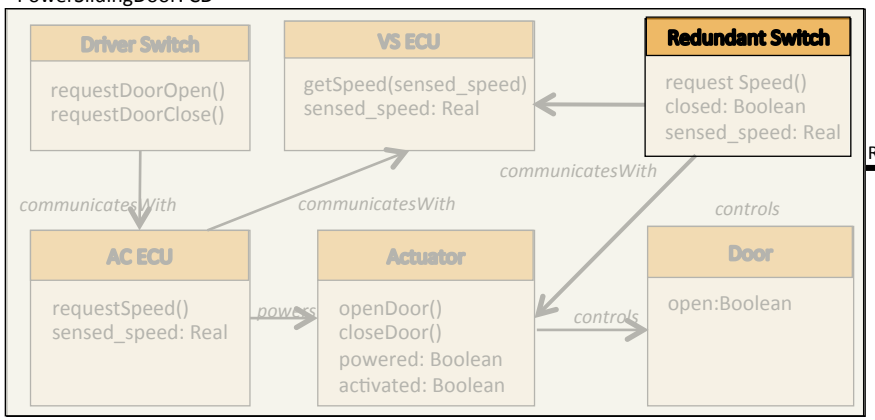


Slicing rules

Rule	Component under assessment	Dependant parts potentially impacted
CD1	Class	Owned attributes and methods. Associations connected to class. Attributes/methods in other classes using types introduced in this class. Subclasses.
SD1	Term (portion of an expression)	Associated expression.
SD2	Expression (guard/action)	Associated message.
SD3	Message	Associated arrow (from source to target lifeline).
SD4	Arrow	Arrows directly after the arrow in the sequence. Message on the arrow.
SD5	Lifeline	Arrows connected to the lifeline. Messages on arrows connected to the lifeline.

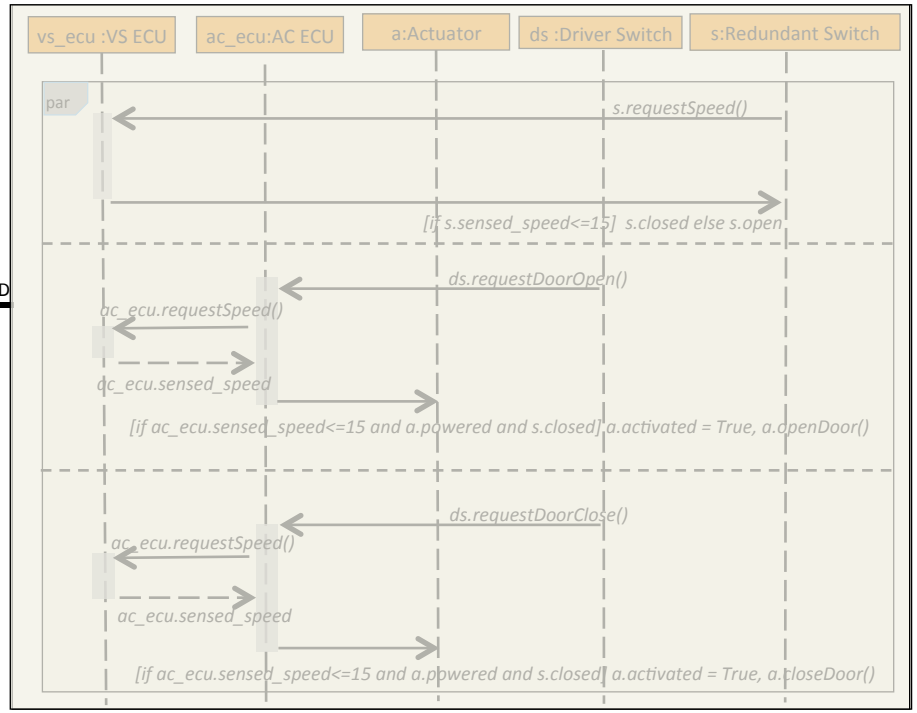
1st iteration: Slicing Criterion

PowerSlidingDoor: CD



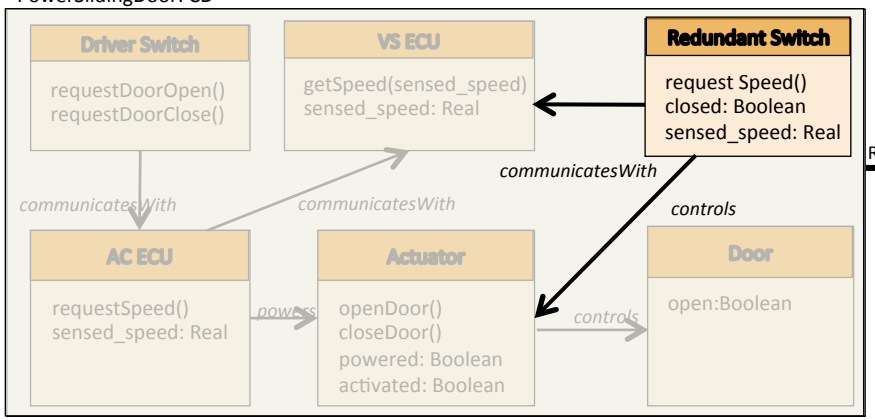
R: CD-SD

PowerSlidingDoor: SD



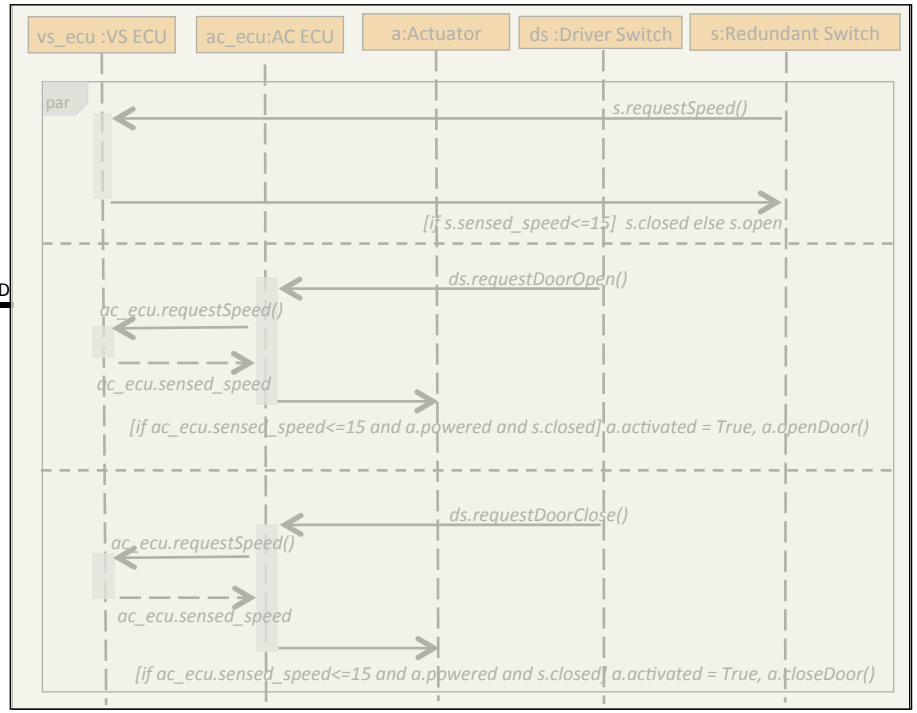
1st iteration: Level 1 slice

PowerSlidingDoor: CD



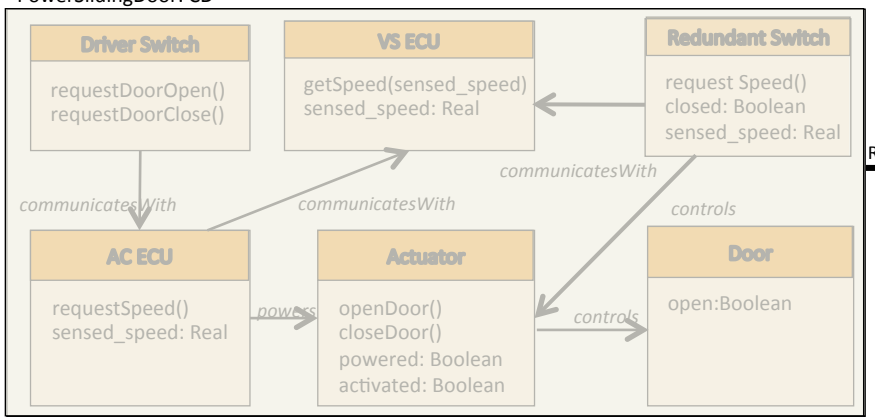
R: CD-SD

PowerSlidingDoor: SD



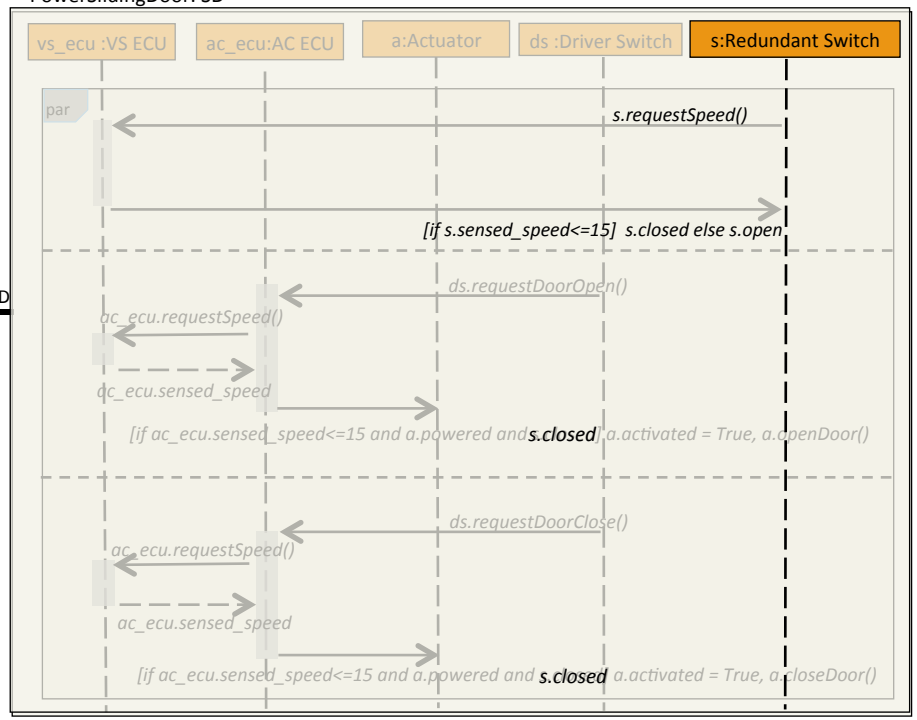
1st iteration: Level 2 slice

PowerSlidingDoor: CD

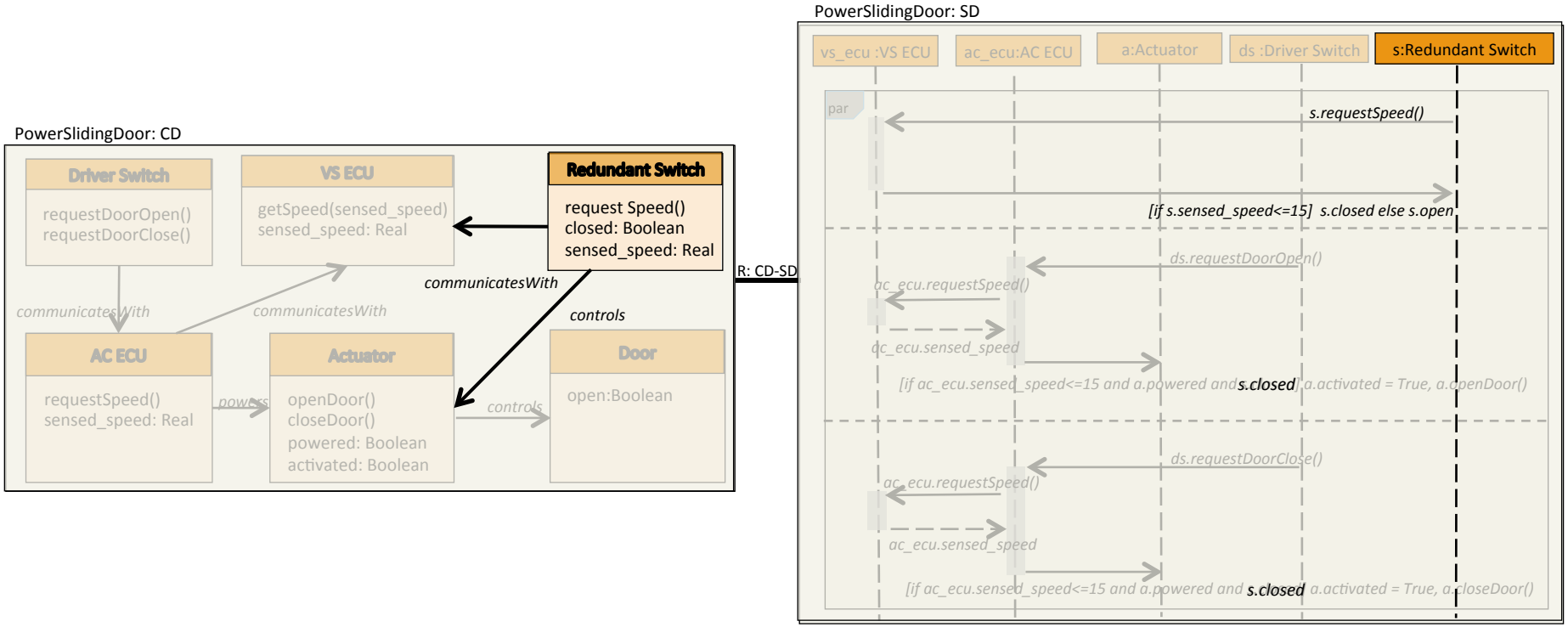


R: CD-SD

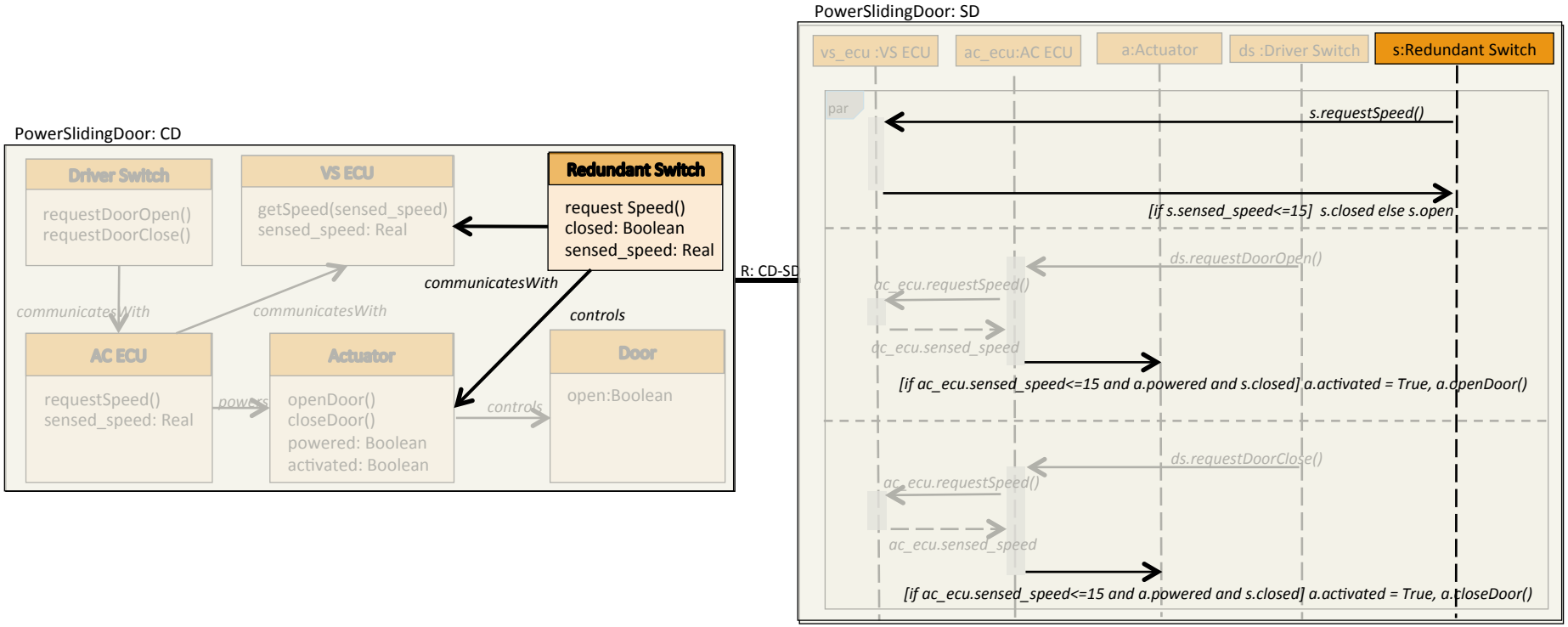
PowerSlidingDoor: SD



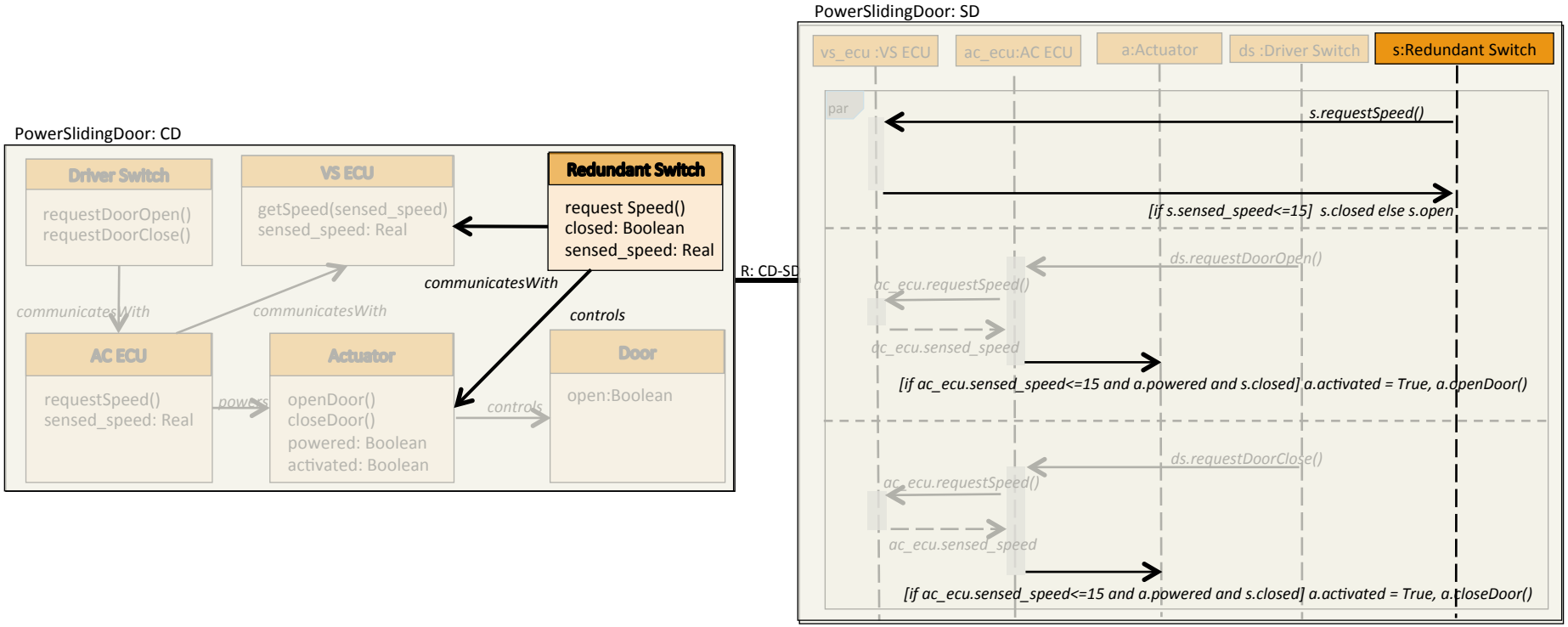
1st iteration: Result



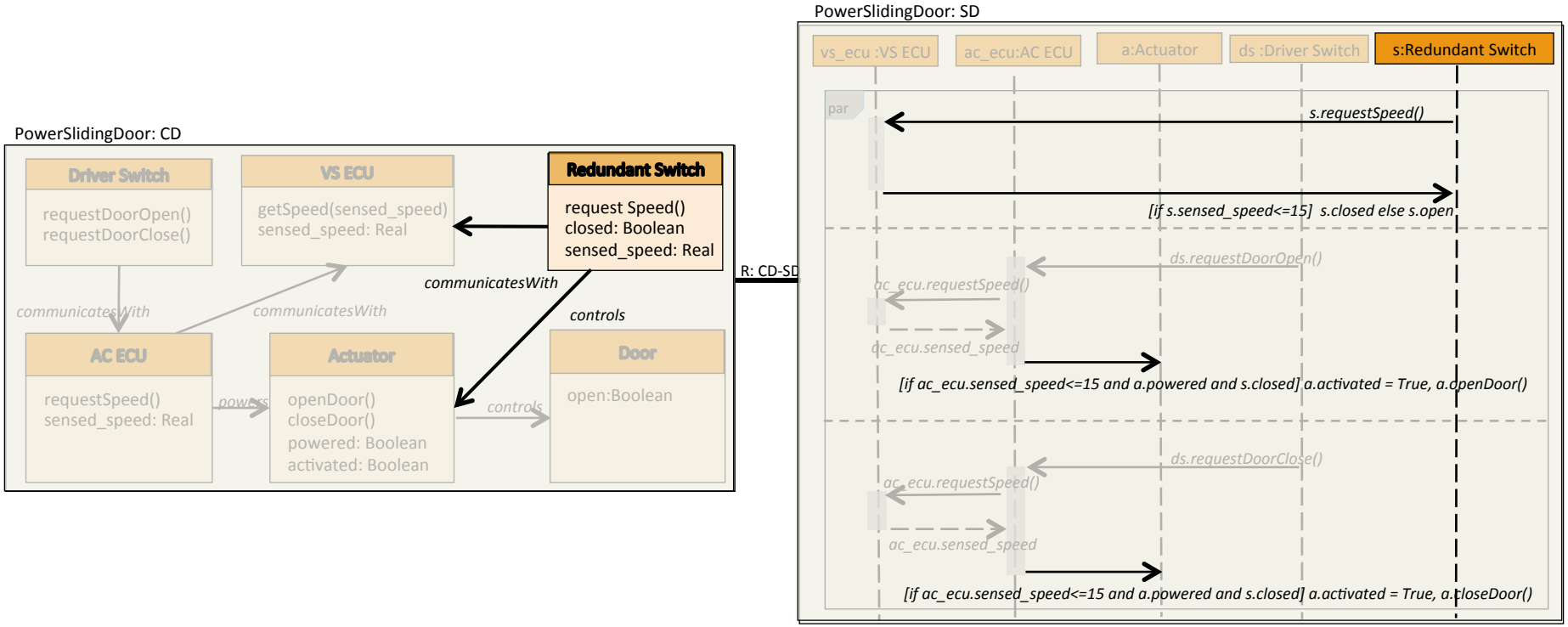
2nd iteration: Level 1 slice



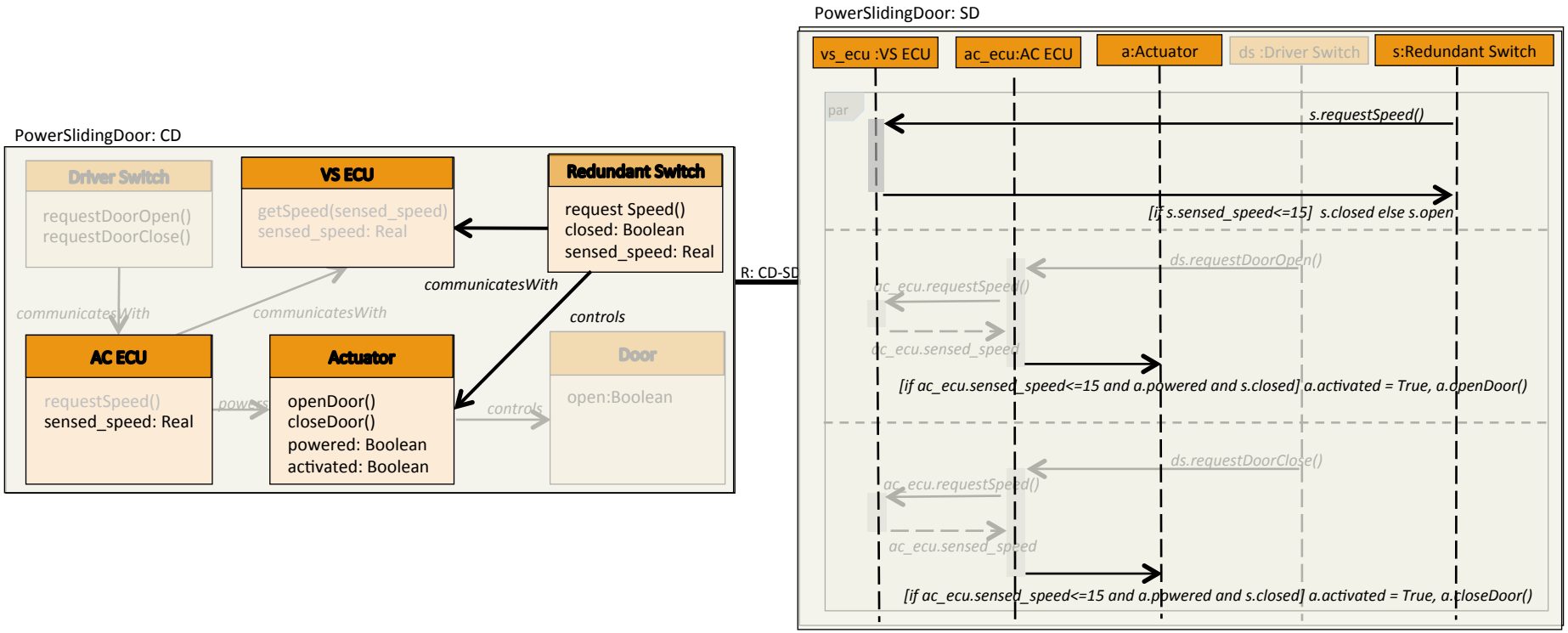
2nd iteration: Result



3rd iteration: No change - done



Post processing: well-formedness and referential integrity



Analysis of the algorithm

Correctness and Termination

- Given the assumptions, the algorithm is shown to be correct and guaranteed to terminate

Worst Case Time complexity

- $O(N_a \, x \, N_M \, x \, SL(N_a) + N^2{}_M \, x \, N^2{}_a)$
- Where N_a= # of atoms, N_M= # of models, $SL(n)$ is upper bound of slicer time complexity when input has n atoms

Minimality

- Algorithm is shown to produce the minimal slice containing all atoms dependent on the slicing criterion

Summary of Contributions

Proposed: a slicing algorithm for heterogeneous megamodels that:

- works with collections of arbitrary model types by using existing type-specific model slicers
- uses standard model relationships (e.g. traceability relationships) for inter-model dependencies

Proved: algorithm is minimal, correct, terminates

Showed: algorithm has quadratic time complexity relative to slicer time complexity

Demonstrated: algorithm on a simple case study from an existing standard (ISO 26262)

Current and Future Work

We plan to generalize the algorithm to address the following:

- N-ary relationships
  - currently on binary relationships are supported
- Nested megamodels
  - currently the input megamodel cannot contain other megamodels
- Arbitrary relationship types
  - currently only dependency relationships are allowed – others: e.g, refinement, overlap, etc.

We are developing an implementation using the Model Management INTeractive (MMINT)* framework.

**https://github.com/adisandro/MMINT/*

Questions?

Heterogeneous Megamodel Slicing for Model Evolution

RICK SALAY, SAHAR KOKALY, MARSHA CHECHIK AND TOM MAIBAUM

ME 2016 @ MODELS'16

OCT. 2, 2016, SAINT MALO, FRANCE