Simulation Preorder Semantics for Traceability Relations in Enterprise Architecture

Mika Cohen

Simulation preorder semantics for traceability relations in Enterprise Architecture

Mika Cohen

Swedish Defence Research Agency (FOI)
Information and Aeronautical Systems
Decision Support Systems

Royal Institute of Technology (KTH)
School of Computer Science and Communication
Theoretical Computer Science

Abstract. The paper proposes a formal semantics for traceability relations in enterprise architecture. The proposed semantics requires that traceability relations should be simulation preorders, a requirement on abstraction relations widely used in program verification. The effectiveness of the proposed semantics is illustrated on a well-known enterprise architecture model from the military domain.

Key words: Enterprise Architecture, Traceability, Verification

1 Introduction

An enterprise architecture describes an enterprise through a sequence of abstraction layers, with each successive layer refining (implementing) the layer above. Model elements can be traced across abstraction layers via traceability relations that relate an element to its implementations (realisations) at lower, more concrete layers. Typically, the enterprise architecture modelling language provides only a weak formal semantics for its traceability relations; model elements at an abstraction layer may trace to model elements at higher abstraction layers in more or less arbitrary ways without violating the formal semantics. However, not every mapping between abstraction layers is intuitively reasonable given the internal connections within each abstraction layer. For example, it would be unreasonable to map two *interacting* technical system at a lower layer to two *independent* business functions at a higher abstraction layer. Since the semantics for traceability relations is (mostly) informal, verifying the traceability links in a model is a manual process, and as such can be both time consuming and error-prone, especially so for large models.

When looking for a formal semantics for traceability relations it is reasonable to consider *abstraction relations* [1] from program verification. Abstraction relations in program verification have the same intuitive semantics as traceability relations in enterprise architecture but come with a well-founded and much applied mathematical theory.

In this paper we propose a formal semantics for traceability relations based on *simulation preorder*, the most widely used abstraction relation in program verification [2]. The formal semantics translates directly to an executable modeling guideline - in OCL, SQL, SPARQL, or other rule- and query language used in enterprise architecture modeling tools – that warns a user about problematic traceability links. We show its effectiveness (in identifying modeling errors) on a well-known architecture model from the military domain.

The proposed semantics might be too restrictive to be a mandatory part of a general-purpose enterprise architecture modeling language intended also for relaxed, imprecise modeling. However, a particular organisation or modeling project might adopt the proposed semantics as an executable part of their organization- or project specific modeling guidelines. In fact, the proposed semantics is already a part of a 'rule book' that FMV Swedish Defence Material Administration uses to verify enterprise architecture models.

The rest of the paper is organized as follows. Section 2 defines enterprise architecture models. Section 3 presents the formal semantics in the form of integrity constraints on enterprise architecture models. Section 4 shows how to implement the semantics in rule- and query languages. Section 5 presents a case study. Section 6 discusses related work. Finally, section 7 concludes.

2 Enterprise Architecture

In this section we define enterprise architecture models. The definition is simplified to avoid distracting detail in the following section. Informally, an enterprise architecture describes an enterprise through a sequence of abstraction layers, with each successive layer refining (implementing) the layer above; traceability relations link elements to their implementations (realisations) at lower, more concrete layers.

Example 1. [MODAF] As our running example we consider MODAF, an enterprise architecture modeling language developed by the UK Ministry of Defence. MODAF has three abstraction layers: green, blue and orange. The green 'strategic' layer specifies the intended business outcome and the capabilities these require, the blue 'operational' layer describes the processes and information flows needed to fulfill the capabilities specified at the green layer, and, finally, the orange 'system' layer detail the physical implementation of the processes and information flows from the blue layer. In this paper, we consider only a representative fragment of MODAF, shown in figure 1. Models (over this fragment) contain interdependent capabilities in the green abstraction layer; nodes that performs activities and exchange information in the blue abstraction layer; and, finally, resources that perform functions and exchange data in the orange abstraction layer. Traceability links (represented by dotted arrows in figure 1) connect nodes to capabilities, resources to nodes, functions to activities, resource interactions to information exchanges, and, finally, data elements to information elements. As a (toy) example model, figure 2 depicts a MODAF-model describing baby rearing.

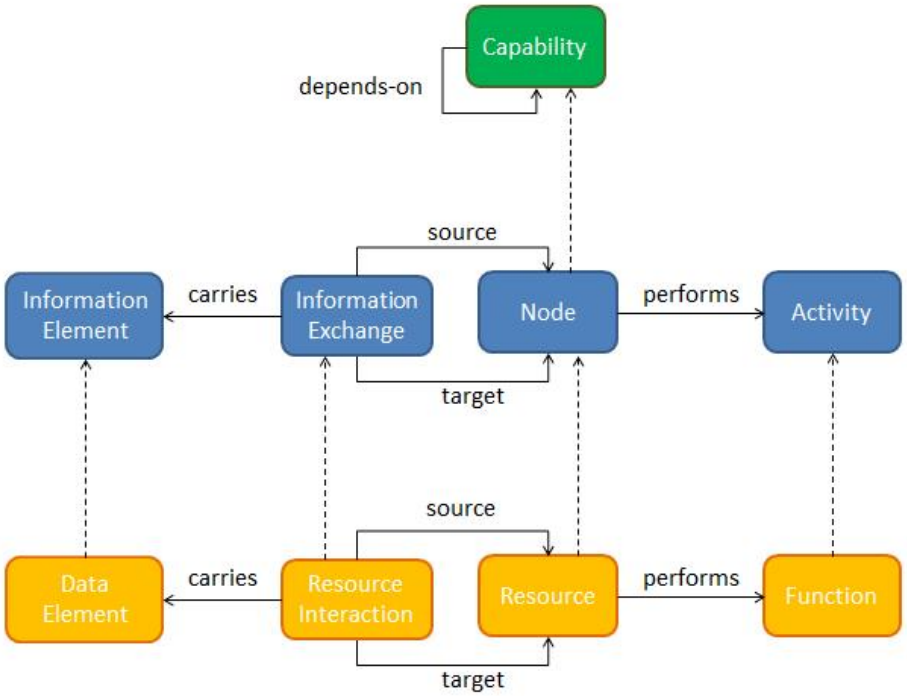


Fig. 1: MODAF fragment

Formally, an enterprise architecture modeling language, *language* for short, is a non-empty set O of unary predicates ('classes') and binary predicates ('relations'), including a special traceability relation, hereafter denoted *implements*.[1]

Example 2. Continuing the above example, the MODAF fragment O considered in this paper contains classes:

- *Capability* (from the green abstraction layer)
- *Node, Activity, InformationExchange, InformationElement* (from the blue abstraction layer)
- *Resource, Function, ResourceInteraction, DataElement* (from the orange abstraction layer)

and relations:

- *depends-on, performs, source, target, carries, implements*

Note that the MODAF abstraction layers are not explicitly captured in O.

Informally, a model M is a set of facts expressed with the given vocabulary in O. Formally, *facts* over a domain D (i.e., a non-empty set of elements) and

[1] For ease of presentation, we assume a single (un-typed) traceability relation.

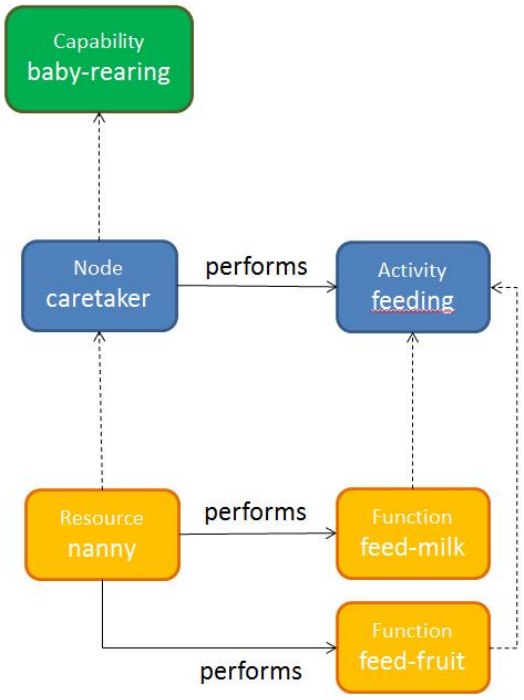


Fig. 2: MODAF model

a language O have either the form a: C or have the form a R a', where a and a' are elements from the domain D, and C and R are, respectively, classes and relations in O. Intuitively, a: C asserts that element a belongs to class C, while a R a' asserts that relation R relates element a to element a'. In particular, the fact a *implements* a' asserts that element a realises (implements, refines, supports) element a'.

Definition 1 (Model). *An enterprise architecture model, model for short, over a language O is a non-empty set M of facts over O and some domain D.*

Example 3. The model M from Example 1 and figure 2 contains the facts:

- *baby-rearing: Capability*
- *caretaker: Node*, *feeding: Activity*, *care-taker performs feeding*
- *nanny: Resource*, *feed-milk: Function*, *feed-fruit: Function*, *nanny performs feed-milk*, *nanny performs feed-fruit*
- *caretaker implements baby-rearing*, *nanny implements caretaker*, *feed-milk implements feeding*, *feed-fruit implements feeding*

The above definition of an enterprise architecture model is, of course, simplified. In particular, the definition does not explicitly capture abstraction layers – explicit abstraction layers would unnecessarily complicate the presentation of

the semantics in the next section. Moreover, the definition ignores the customary typing constraints inherited from UML class diagramming − these are both standard and straight forward .

3 Semantics for traceability relations

In this section we propose a formal semantics for traceability relations in enterprise architecture. Informally, a traceability relation links an element to its implementations (realisations) at lower, more concrete layers. To capture this intuitive semantics, we require that the traceability relation is a simulation preorder, a requirement on abstraction relations widely used in program verification.

Roughly, we require that an association between two elements at the lower abstraction layer is permitted only if there is a corresponding association between their abstractions at the higher abstraction layer; the structure at the higher abstraction layer thus constrains the possible solutions (realisations) at the lower abstraction layer.

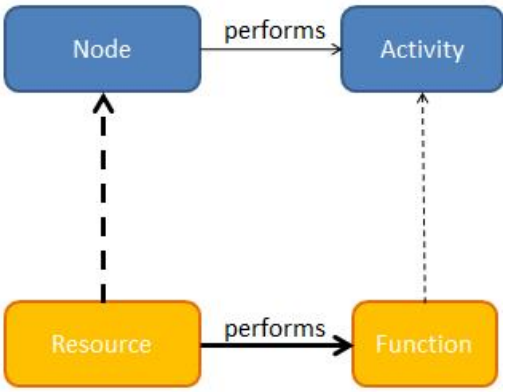
Example 4. Continuing the earlier examples, the semantics proposed will enforce the integrity constraints on MODAF-models shown in figures 3a, 3b, 3c, and 4a, where *abstracts* is the inverse relation to the traceability relation (*implements*). The constraints are expressed in SBVR Structured English and should be self-explanatory. The diagrams should be interpreted as saying that the two thicker arrows jointly imply the existence of two thinner arrows.

In the above example, a relation R (*performs, target, source, carries*) at the higher abstraction layer corresponds to the same (identically named) relation R at the lower abstraction layer; for each move along R at the lower layer there must exist a corresponding move along R at the higher abstraction layer. However, a relation R at the higher abstraction layer may sometimes correspond to a differently named relation R' at the lower abstraction layer.

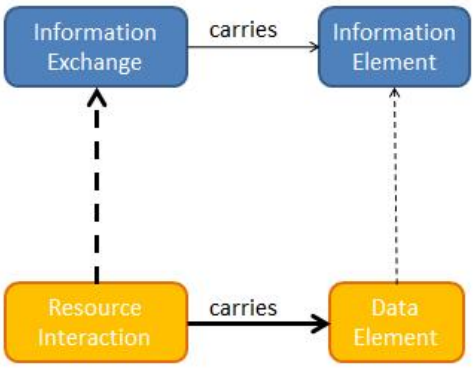
Example 5. Continuing the above examples, the semantics proposed will enforce the integrity constraint on MODAF-models shown in figure 4b. Here, the relation R = *depends-on*, at the green strategic abstraction layer, corresponds to the relation R' = *receives-from*, at the blue operational abstraction layer. The latter relation is a derived relation: *a node1 receives-from a node2 if there exists some information exchange that targets the node1 and that is sourced from the node2.*

From now on we assume that an enterprise architecture modelling language O comes with a counterpart function, i.e., a partial function $f\colon O \longrightarrow O$ that maps relations to their more abstract counterparts (if any).[2]

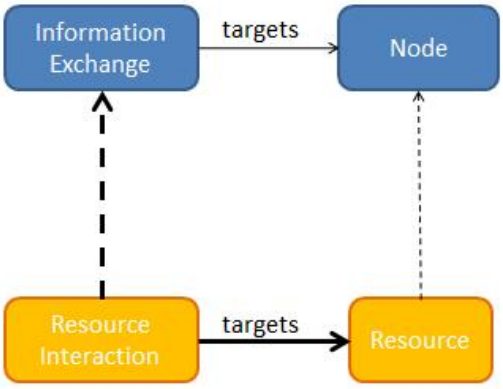
[2] For ease of presentation, we assume (somewhat sloppily) that O contains also derived relations, such as *receives-from* in the case of MODAF. For ease of presentation, we assume moreover that the traceability relation connects only between directly neighbouring abstraction layers



(a) *A node that abstracts a resource that performs a function, must perform an activity that abstracts the function*

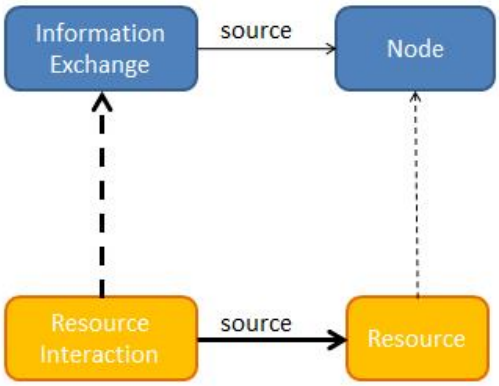


(b) *An information exchange that abstracts a resource interaction that carries a data element, must carry an information element that abstracts the data element*

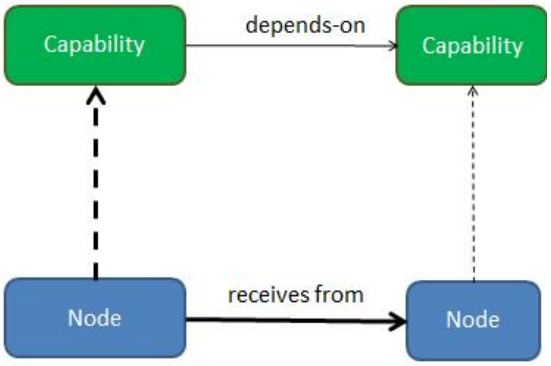


(c) *An information exchange that abstracts a resource interaction that targets a resource, must target a node that abstracts the resource*

Fig. 3: Integrity constraints enforced by the simulation preorder semantics



(a) *An information exchange that abstracts a resource interaction that is sourced from a resource, must be sourced from a node that abstracts the resource*



(b) *A capability1 that abstracts a node1 that receives from a node2, must depend on a capability2 that abstracts the node2*

Fig. 4: Integrity constraints enforced by the simulation preorder semantics

Example 6. Continuing Example 2, we assume the MODAF-fragment O comes with the following counterpart function f: *f(receives-from) = depends-on*, *f(performs) = performs*, *f(carries)=carries*, *f(source)=source*, *f(target)=target*

Of course, counterpart functions are not an explicit part of enterprise architecture languages, as found in the 'real world'. We believe, however, that they are there implicitly. Relations that correspond to each other will typically either be identically named or identically (stereo-)typed in the language meta-model. When this is not the case, informal modelling directives may indicate correspon-

dences. As an example, the official MODAF handbook at the Swedish Armed Forces states: *'Dependencies between capabilities ought to lead to interaction between the instantiating nodes'* (*Handbok för försvarsmaktens tillämpning av MODAF*, Section 6.3.1.4.1, authors translation from Swedish). In other words, the relation R = *depends-on* at the green strategic abstraction layer corresponds to the relation R' = *receives-from* at the blue operational abstraction layer.

We are now in a position to formulate our proposed semantics for traceability relations. Assume a modeling language O with a counterpart function f, and let M be a model over O. Roughly, we require that for every arc in M at the lower abstraction layer there is a corresponding arc at the higher abstraction layer (see figure 5).

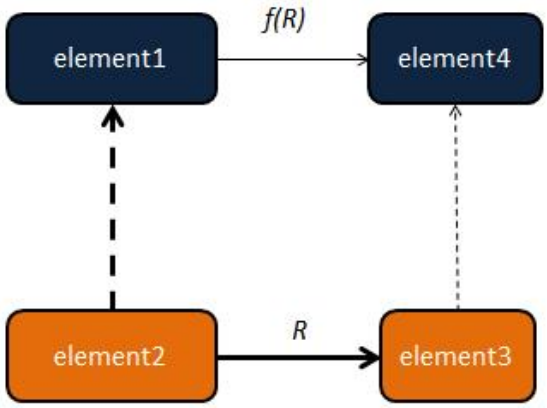


Fig. 5: Simulation preorder semantics. *An element1 that abstract an element2 that is R-related to an element3, must be f(R)-related to an element4 that abstracts the element3*

Definition 2 (Semantics for traceability). *We say that traceability relations are correct in M if the following condition holds for all elements a, b and a' in the domain and for all relations R in O such that f(R) exists: if M contains facts*

- *a R b*
- *a implements a'*

then there exists an element b' in the domain such that M contains facts

- *a' f(R) b'*
- *b implements b'*

Example 7. Continuing the above examples, traceability relations are correct in a model M (over our MODAF-fragment O) if the integrity constraints in figures 3a, 3b, 3c, 4a, and 4b hold. In particular, traceability relations are correct in the

model over baby rearing (Example 3 and figure 2) since the functions the nanny performs are reflected in activities performed by the caretaker.

4 Implementation

The formal semantics proposed in the previous section translates directly to executable modeling guidelines in OCL, SQL, SPARQL, and other rule- and query languages used in modeling tools. In fact, the integrity constraints in SBVR-SE that instantiate the semantics in the previous section compile automatically to SQL with SBVR-compilers (c.f. [3, 4]).

Example 8. The constraint in figure 3a translates to the following SPARQL-query that identifies traceability links (between resources and nodes) that violate the constraint:

```
SELECT ?resource ?node {
  ?resource a Resource.
  ?node a Node.
  ?function a Function.
  ?resource implements ?node.
  ?resource performs ?function.
  NOT EXISTS {
    ?activity a Activity.
    ?node performs ?activity.  ?function implements ?activity
  }
}
```

Of course, executable modeling guidelines should preferably produce appropriate warning messages, not merely list data.

Example 9. With SPARQL Inference Notation (SPIN), the SPARQL-query that identifies the constraint-violations can be associated with a custom error-message. E.g., the query-logic from the previous example can be associated with the error message:

```
CONCAT(
  ?resource, 'implements ', ?node,
  ' but ', ?resource, ' performs a function ', ?function,
  ' that does not implement some activity performed by ', ?node
)
```

Of course, the error-message that the executable modeling guideline produces need not necessarily point the finger at traceability links as the source of error. For some applications it might be more reasonable to assume that when a traceability link fails the simulation preorder semantics, the most likely cause of error is a mismatch between the higher- and lower abstraction layers, i.e., either a R-relation in the lower abstraction layer is unwanted or a correspondning $f(R)$-relation in the higher abstraction layer is missing.

Example 10. Continuing the above example, the implementation could accept the traceability links (between resources and nodes) as given and instead warn about illegitimate functions, i.e., functions not sanctioned by the higher abstraction layer:

```
CONCAT(
  ?resource, ' performs an illegitimate function ', ?function,
  ' that does not implement any activity performed by ', ?node
)
```

The proposed semantics has been implemented in MooD (as SQL-queries) and in MagicDraw (as OCL-constraints). The latter was used in the case study discussed in the next section.

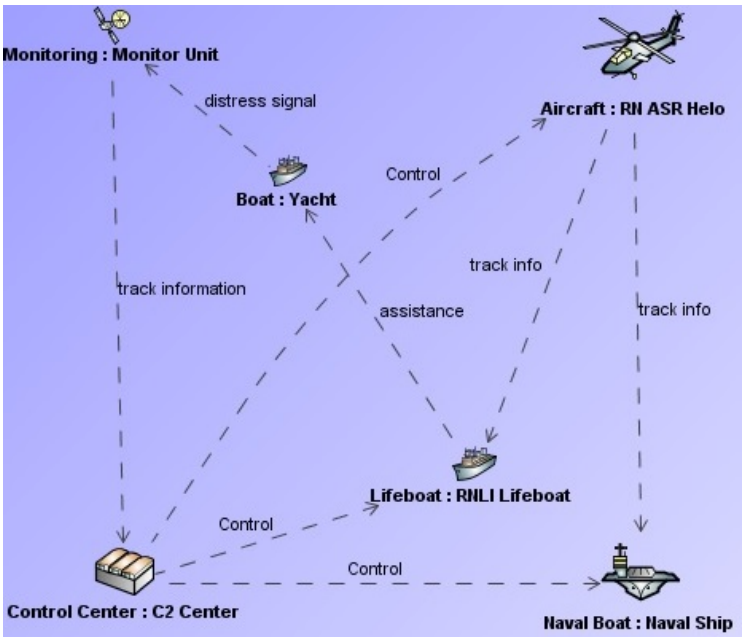


Fig. 6: SAR scenario (snippet)

5 Case study

The semantics for traceability relations proposed above is part of a 'Rule book for MODAF' developed at FOI (the Swedish Defence Research Agency) and used at FMV (the Swedish Defence Material Administration) to verify MODAF models. Unfortunately, the models that have been verified at FMV are secret, and cannot

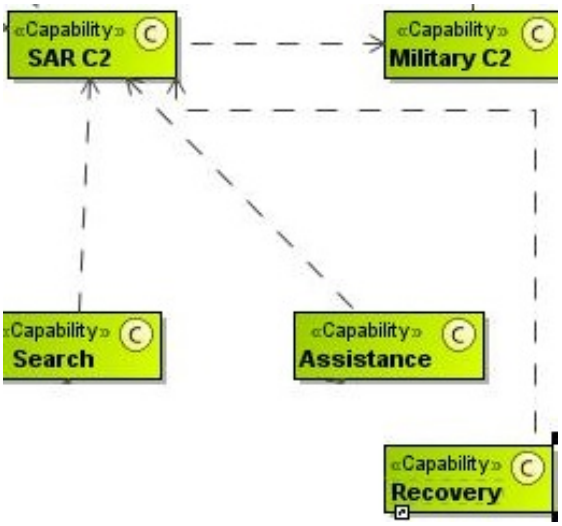


Fig. 7: Capability dependencies (snippet)

be discussed in this paper. Instead, we illustrate the proposed semantics and its implementation in MagicDraw on the well-known SAR (Search And Rescue) model from the UK Ministry of Defence, a publically available MODAF model. The implementation identified numerous modeling errors. In this section, we consider two of the identified errors. To the best of our knowledge, neither of these errors has been identified in the MODAF-literature.

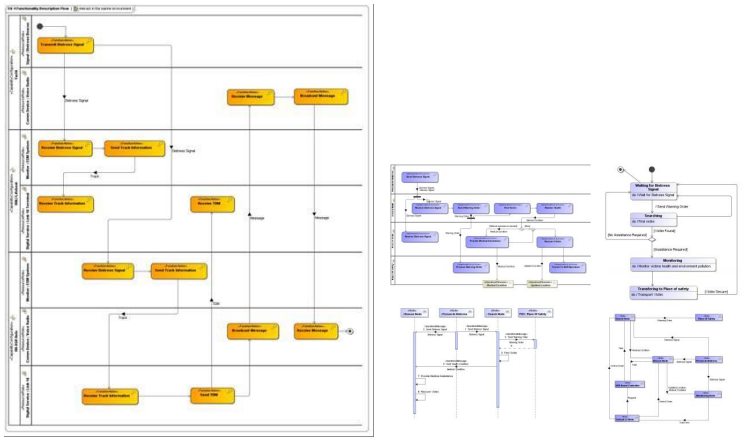
Figure 6 shows a scenario snippet from the SAR model in which a distressed yacht signals for help. The distress signals are caught by a rescue team consisting of a life boat (*RNLI Lifeboat*), a helicopter (*RN ASR Helo*) and other resources. The arcs between resources represent resource interactions. E.g., the lifeboat receives data (*track info*) from the helicopter .

Figure 7 shows another view from the same SAR model, this time from the more abstract green strategic layer. Here, the search and rescue capability is defined at a higher level of abstraction; the capability is decomposed into a number of simpler capabilities and dependencies (represented by dotted lines) between these. For example, the capability *Recovery* depends on the capability *SAR C2*.

The scenario in figure 6 is intended to realise the more abstract capability definition in figure 7; each resource in figure 6 implements some capability. E.g., the helicopter *RN ASR Helo* implements the capability *Search* while the lifeboat *RNLI Lifeboat* implements the capability *Recovery*. The traceability links between resources and capabilities are scattered at various places in the SAR model.

Are the traceability links between resources and capabilities correct? With the semantics for traceability relations implemented in our modeling tool (Mag-

icDraw), we simply press a button to find out. After a few seconds the modelling tool produces a number warnings, among others: 'Resource interaction between *RN ASR Helo* and *RNLI Lifeboat* is not reflected in any capability dependency'. According to the warning, the helicopter *RN ASR Helo* exchanges data with the life boat *RNLI Lifeboat* (see figure 6) but there is no dependency between the capabilities these resources realise, *Search* and *Recovery* respectively, in the more abstract view, i.e., there is no dotted line between *Search* and *Recovery* in figure 7; Note that the error message does not warn about an incorrect traceability links per se; rather it warns about a mismatch between the scenario view in figure 6 and the capability view in figure 7.



(a) Scenario realisation (miniature snapshot)

(b) Abstract scenario (miniature snap shots)

Fig. 8: Abstract scenario and realisation

Continuing, we consider next a more detailed scenario realisation at the orange system layer in the SAR model (figure 8a). The scenario starts when a distressed yacht transmits distress signals (top swim lane in figure 8a) that are eventually picked up (*Receive Distress Signal*, third swim lane from bottom) by a monitoring system on the helicopter *RN ASR Helo*. The helicopter eventually sends a message back to the distressed yacht (*Broadcast Message*, second swim lane from bottom), the yacht receives the message (*Receive Message*, second swim lane from top) and sends a reply (*Broadcast Message*, second swim lane from top), and, finally, the helicopter receives the reply (*Receive Message*, second swim lane from bottom).

The scenario in figure 8a is intended to realise a more abstract scenario definition from the blue, operational abstraction layer (figure 8b); resources and functions from figure figure 8a implement, respectively, nodes and activities from the blue abstraction layer (figure 8b). E.g., the resource *Yacht* maps to the node

Person in Distress, while the functions *Send Message* and *Broadcast Message* both map to the activity *Send Distress Signal*. Again, the implementation-links are scattered in the SAR model.

Are the traceability links correct? Again, we simply press a button to find out. As before, the modelling tool warns us about a number of identified modelling errors, among others the error: '*Yacht* performs illegitimate function *Receive Message*'. According to this warning, the function *Receive Message* is not sanctioned by the more abstract scenario definition at the blue operational layer; the integrity constraint from figure 3a is violated. In more detail, the yacht receives messages while its abstraction at the blue layer, *Person In Distress*, merely sends distress signals (see figure 9). This might be a rather serious modeling error. The blue operational layer specifies a capability of rescuing a person in distress who sends distress signals. But the proposed physical realisation (figure 8a) assumes that the person in distress is reachable (can be contacted), an assumption which cannot be traced back to the scenario specification at the blue abstraction layer (figure 8b).

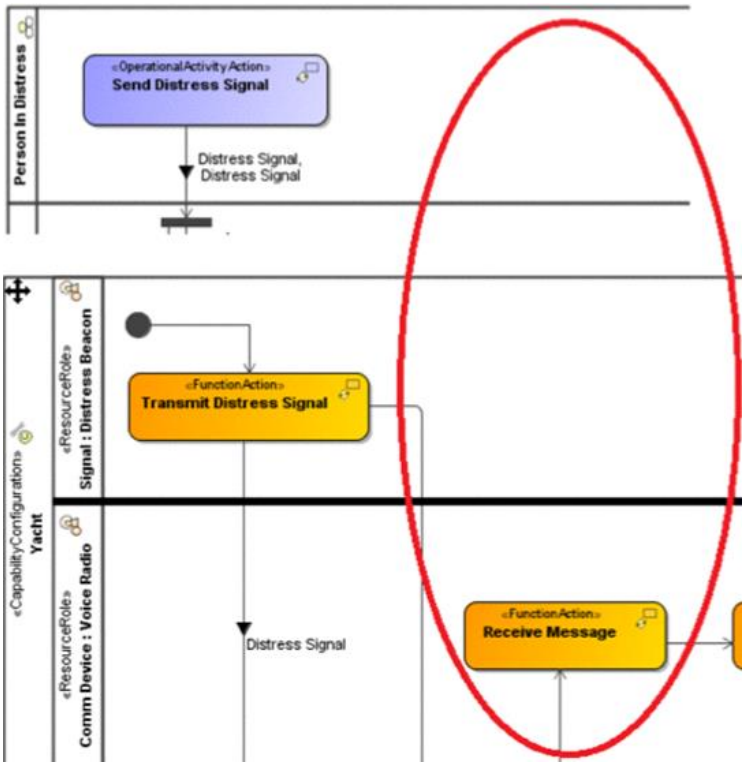


Fig. 9: Identified Error. The *Yacht* both transmits and receives but its abstraction, *Person In Distress*, only sends

6 Related work

Some enterprise architecture modeling tools enforce cardinality constraints on traceability relations. E.g., MagicDraw warns if e.g. a resource at the orange system layer in MODAF does not implement any node from the blue operational layer. Clearly, cardinality constraints alone constitute a weak semantics. In particular, none of the example modeling errors in the case study above fail such cardinality constraints.

[5] extends ArchiMate, a particular enterprise architecture modeling language, with inference rules that derive (numerical) data attributes in an element from other attributes in the same or related elements. The inference rules reflect empirically established correlations ('laws of causation') rather than an informal intuitive semantics, as do the integrity constraints in the present paper. [6, 7] extend DoDAF and MODAF with inference rules capturing empirical correlations between high-level capabilities and attributes of the implementing technical systems.

Conditions similar to simulation preorder have been used as tools for debugging ontology mappings (cf. [8, 9, 10]). The approach in the present paper is similar: an informal, intuitive semantics for 'correspondences' is captured using mathematical constructions from theoretical computer science. However, the application in the present paper − traceability relations in enterprise architecture − is, to the best of our knowledge, novel.

7 Conclusion

Traceability relations trace model elements across abstraction layers in an enterprise architecture. Verifying traceability links is a manual, time consuming and error-prone process − existing formal semantics for traceability relations is weak (merely the cardinality constraints familiar from UML class diagrams).

The paper proposed a formal semantics for traceability relations in enterprise architecture. The proposed semantics required that traceability relations should be simulation preorders, a requirement on abstraction relations widely used in program verification. The effectiveness of the proposed semantics was illustrated on a well-known enterprise architecture model from the military domain.

Traceability relations play an important role not only in enterprise architecture but in model-based engineering more broadly. In the future, it would therefore be interesting to extend the semantics proposed to the model transformations in model-based engineering.

References

1. P. Cousot and R. Cousot: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixed points. In POPL, 1977

2. E.M. Clarke, O. Grumberg, and D.E. Long: Model checking and abstraction. ACM Trans. Program. Lang. Syst., 1994
3. M. Minnock: C-Phrase: A system for buidling robust natural kanguage interfaces to databases. DKE, 2010
4. S. Moschoyiannis, A. Marinos, P. Krause: Generating SQL queries from SBVR rules. RuleML, 2010
5. P. Johnson and M. Ekstedt: Enterprise Architecture: Models an analysis for information systems decision making. 2007
6. U. Franke, W.R. Flores, P. Johnson: Enterprise Architecture dependency analysis using fault trees and Bayesian networks. Spring Simulation Multiconference, 2009
7. U. Franke, P. Johnson, E. Ericsson, W. R. Flores, K. Zhu: Enterprise Architecture analysis using Fault Trees and MODAF. CAiSE Forum, 2009
8. S. Ghilardi, C. Lutz, F. Wolter: Did I damage my Ontology? A case for conservatve extensions in description logics. KR, 2006
9. J. Trevor, M. Bench-Caopn, G. Malcolm: Formalising ontologies and their relations. DEXA, 1999.
10. M. Cohen: Semantics for mapping relations in SKOS. RR, 2013