Enterprise architecture with executable modelling rules: A case study at the Swedish Defence Materiel Administration

Mika Cohen[1], Michael Minock[2], Daniel Oskarsson[1], and Björn Pelzer[1]

[1] FOI, Stockholm, Sweden,
{mika.cohen,daniel.oskarsson,bjorn.pelzer}@foi.se
[2] KTH Royal Institute of Technology, Stockholm, Sweden,
minock@kth.se

Abstract. Formal modeling rules can be used to ensure that an enterprise architecture is correct. Despite their apparent utility and despite mature tool support, formal modelling rules are rarely, if ever, used in practice in enterprise architecture in industry. In this paper we propose a rule authoring method that we believe aligns with actual modelling practice, at least as witnessed in enterprise architecture projects at the Swedish Defence Materiel Administration. The proposed method follows the business rules approach: the rules are specified in a (controlled) natural language which makes them accessible to all stakeholders and easy to modify as the meta-model matures and evolves over time. The method was put to test during 2014 in two large scale enterprise architecture projects, and we report on the experiences from that. To the best of our knowledge, this is the first time extensive formal modelling rules for enterprise architecture has been tested in industry and reported in the literature.

Key words: enterprise architecture, data quality, meta-model, semantics, business rules, case study

1 Introduction

An enterprise architecture (EA) model is composed of symbols (boxes, arrows, words, etc.) that combine to make claims about the business being modelled. How the symbols combine to express meaningful statements is given by the model's *semantics*. Part of the semantics is typically governed by an explicit *meta-model*; ideally, the rest is governed by informal or tacit agreement across modellers and model users.

As long as the model semantics is thus fully determined, explicitly or informally, it can, at least in part, be captured by formal (and thus executable) modelling rules[1], e.g. in OCL[2] [8, 13, 14], forming an *extended meta-model* which

[1] Sometimes referred to as "compliance rules"
[2] Object Constraint Language

can in turn be used to automatically verify that the model complies with the semantics.

But what if part of the semantics are not even implicit but thoroughly undetermined? In a typical modelling scenario, with multiple modellers and stakeholders modifying and interacting with the model over a span of time, the risk is then that semantic underdetermination leads to model inconsistencies that go undiscovered by automatic compliance checking, since that which is undetermined cannot be formalized into compliance rules.

Fortunately, the very process of formulating executable rules lends itself to weeding out vagueness: Since a rule, to be executable, must be expressed in precise concepts, formulating the rule entails specifying those concepts that are yet not precise.

To secure the participation and involvement of a broad range of stakeholders in the rule formulation process, and thus ultimately to ensure the quality of the extended meta-model thus produced, it helps if the rules, in addition to being formal, are expressed in a language that resembles natural language.

So, resting upon the assumptions that (1) it is useful to continually submit an EA model to validation against semantic rules, (2) the process of expressing formal rules has the effect of forcing semantic specification, bringing value by precluding model vagueness, and (3) natural language rules improve quality by involving a wider range of stakeholders in the formulation of the semantics, we propose a method for automatic model validation and continual semantic specification based on semantic rules expressed in a controlled natural language.

We report the results from applying this method within two EA projects at the Swedish Defence Materiel Administration (FMV) in the main section of this paper (5). But first, in section 2, we describe the background that lead us to formulate a set of hypotheses about the role of modelling rules in EA projects—listed in section 3—which in turn lead to the formulation of the method—described in section 4. The case study section (5) evaluates the method in terms of the hypotheses, and the verdict is summarized in Conclusions (6).

Related work The need for enterprise architecture to be correct has been pointed out repeatedly and formal modelling rules have been proposed as a suitable means to enforce correctness [1, 4, 6, 8, 10, 13, 14]. Indeed, several commercial EA-modelling tools allow the user to specify formal modelling rules in OCL; the modelling environment will then warn the user whenever data is entered that violates a rule. To the best of our knowledge, however, no extensive use of formal modelling rules in a large scale industrial EA project has been reported previously in the literature.

The meta-modelling method described in the present paper can be seen as *domain specific modelling* [9], a modelling methodology used in software engineering. In domain specific modelling, the meta-model is tailored to suit a narrow problem domain, e.g. a particular product line or a particular software project. The narrow problem domain allows stronger, more effective modelling rules compared to the rather weak modelling rules that come with general purpose software modelling languages such as UML. Typically, the modelling rules

are expressed in OCL or other similarly low-level constraint language inaccessible to stakeholders outside IT.

The rule authoring method described in the present paper, by contrast, follows the *business rules approach* [5]. In particular, the rules are captured in a (controlled) natural language which is subsequently transformed into executable code. In typical business rules applications, the rules capture operational guidelines such as regulatory compliance rules rather than, as in our case, modelling rules for a domain specific modeling language. Moreover, the executable code (that the rules compile to) is typically decision logic (e.g. in a workflow system) rather than database integrity constraints.

Richer, extended EA meta-models are considered in [7], which extends ArchiMate, an enterprise architecture modelling 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.

The natural language compiler used in the case study is described in [3].

2 Background

During 2013, FOI[3] was called in to lend support to the EA modelling project *SK TS*[4] at the Swedish Defence Materiel Administration (FMV). The SK TS architecture describes, at a high level of abstraction, the dependency relationships between technical systems[5], and how the development, production, use and retirement of the systems is planned over time.

Our task was to design a set of consistency rules and implement them as queries into the EA tool[6] used to host the model. The queries were to be run on a regular basis to uncover inconsistencies in the models, and thus to eschew manual "proof reading" that was becoming intractable as the model was growing in size and complexity.

We discovered at an early stage that having access to the model, attendant meta-model and other documentation was insufficient as specification for the rules to be designed; trying to design rules raised a multitude of questions of interpretation, which we directed at the architecture modelling team. Our second discovery was that these questions often did not have ready answers, but gave rise to discussions that fed into the modelling process itself. We also found that it took a few rounds of execution and redesign of the rule queries for them to mature. Finally, we found that implementing the rules directly as queries into the EA modelling tool offered poor overview over the rule set, and that keeping an

[3] Swedish Defence Research Agency

[4] *Systemkarta tekniska system*, Swedish for "system map (over) technical systems"

[5] The term "technical system" can be loosely defined as a category of equipment of non-trivial complexity, encompassing aircraft as well as munitions, but not e.g. clothing.

[6] MooD Business Architect 2010

informal catalogue of rules as a companion to the queries posed its own problems of synchronization.

These realizations lead to a redefinition of our task, from delivering a bundle of rules, implemented as queries, to handing over a framework for continuous rule development, management and execution. Having expressed the rules in the technical syntax most expedient to the task of implementation, we now turned to a controlled natural language to make the rule design process accessible to all stakeholders. The new approach—detailed in the next section—was applied to the continuation of the SK TS project, as well as to a different EA modelling project, called FM UFS, at FMV.

3 Hypotheses

Based on the SK TS experience during 2013, we formulated a number of hypotheses about the possible role of executable semantic rules in EA modelling:

H1 EA models contain many errors that are missed despite extensive manual auditing, but which can be captured automatically through the execution of formal rules.
H2 EA meta-models contain many poorly defined concepts; the activity of designing and executing formal semantic rules uncovers vagueness, imprecision and ambiguity.
H3 EA modelling tends to require project-specific semantics, even when based on well-established architecture frameworks.
H4 EA model semantics need to evolve with the modelling process.
H5 A natural language format for the rules boosts the semantic specification process by stimulating broader participation in rule development.

The rationale behind that last hypothesis (H5) is that if the model semantics need to be developed continually and specifically for the project, and if—as H2 suggests—this semantic development is to be catalysed by the development and execution of semantic rules, then the design and execution of rules is of concern to a broad range of stakeholders, including non-technical ones. Such broad participation should be facilitated by being able to directly execute natural language formulations of rules.

4 Proposed method: Rule authoring as continual modelling support

The method we propose can be described as a business rules approach to EA modelling, where one works simultaneously and iteratively with the architecture modelling process to produce a *rule book* that encodes project specific semantics in SBVR[7] [12], a controlled natural language. The rule book is continually put

[7] Semantics of Business Vocabulary and Rules

to use in validating the architecture model as new rules are formulated, catching semantic errors early in the modelling process. Meanwhile, the very process of creating the rule book extends the range of testable semantics, further shoring up the modelling process.

4.1 Executable natural language rules in SBVR

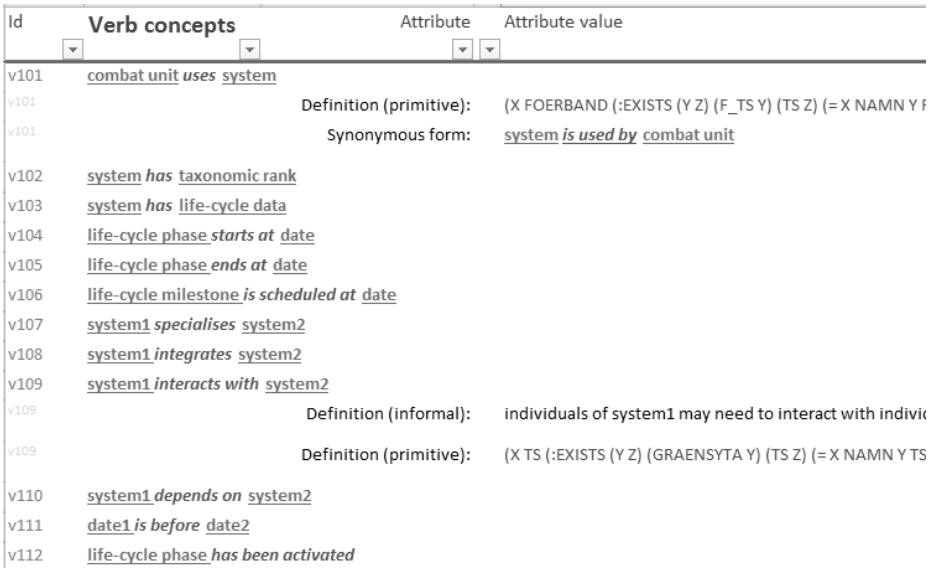
An SBVR model consists of a vocabulary and a set of rules expressed in terms of the vocabulary. The vocabulary is made up of *nouns* (see figure 1), naming entities in the architecture model, and *verbs* (see figure 2), naming relationships between the entities.

Id	Noun concepts	Attribute	Attribute value	New noun concep
n100	system			
n100		Definition (informal):	a category of equipment of non-trivial technical comple›	
n100		Definition (primitive):	(X TS (= X NAME $C1))	
n101	combat unit			
n101		Definition (informal):	a formal organizational unit able able to perform combat	
n101		Definition (primitive):	(X FOERBAND (= X NAME $C1))	
n102	taxonomic rank			
n103	version level system			
n104	life-cycle data			
n105	life-cycle phase			
n106	date			
n107	concept phase			
n108	development phase			
n109	production phase			
n110	use phase			
n110		Definition (informal):	a life-cycle phase during which a system is used	
n110		Category of:	life-cycle phase	
n110		Definition (primitive):	(X LCS (= X LCS "Användning"))	

Fig. 1. Some noun concepts in the SK TS SBVR model, including (optional) informal definitions as well as formal definitions ("Definition (primitive)") in tuple calculus, mapping the nouns to corresponding elements in the database.

The naming of entities and relationships serves the semantic function of appealing to human intuition about what states of affairs in the real world the terms refer to. To provide semantic information to a *machine*, however, we need to specify in what patterns the entities and relationships can appear in the architecture model. This is done by the rule part of the SBVR model.

The rules are statements in the controlled natural language of SBVR, composed of the nouns and verbs of the vocabulary, bound together by generic operators, such as *and*, *or*, *not*, *it is necessary that*, to specify allowable patterns



Id	Verb concepts	Attribute	Attribute value
v101	combat unit *uses* system		
v101		Definition (primitive):	(X FOERBAND (:EXISTS (Y Z) (F_TS Y) (TS Z) (= X NAMN Y F
v101		Synonymous form:	system *is used by* combat unit
v102	system *has* taxonomic rank		
v103	system *has* life-cycle data		
v104	life-cycle phase *starts at* date		
v105	life-cycle phase *ends at* date		
v106	life-cycle milestone *is scheduled at* date		
v107	system1 *specialises* system2		
v108	system1 *integrates* system2		
v109	system1 *interacts with* system2		
v109		Definition (informal):	individuals of system1 may need to interact with indivi
v109		Definition (primitive):	(X TS (:EXISTS (Y Z) (GRAENSYTA Y) (TS Z) (= X NAMN Y TS
v110	system1 *depends on* system2		
v111	date1 *is before* date2		
v112	life-cycle phase *has been activated*		

Fig. 2. Some verb concepts in the SK TS SBVR model, including (optional) informal definitions as well as formal definitions ("Definition (primitive)") in tuple calculus, mapping the verbs to corresponding elements in the database. Attributes shown here only for a subset of the verbs.

in the EA model. As an example, take the last rule in figure 3 (r115), *"It is necessary that a system1 that is part of a system2 that has a use phase2, have a use phase1"*. It uses the nouns *system* and *use phase* (with indices to identify distinct variables of the same class) and the verbs *system1 is part of system2* and *system has life-cycle phase*, connected into a meaningful statement by the modal operator *It is necessary that*, the existential quantifier *a* and the specifying operator *that*. The rule disallows the pattern where a sub-system lacks a use phase while its super-system has one.

Since the rules follow the controlled grammar of SBVR, they are machine-readable and can be "executed", in the sense of generating reports of rule violations committed by the architecture model. Figure 4 shows an example of a violations report produced by executing a rule.

The execution of rules over the architecture model is made possible by compiling them into SQL code that queries the architecture model—that is, the database representation of it in the particular EA modelling tool used—for violations against the rules. To enable such compilation, the nouns and verbs in the SBVR vocabulary must be formally mapped onto entities and relationships of the model, as represented in the database. In our current implementation, the mappings are encoded in Codd's tuple calculus [2] (as can be seen in figures 1 and 2), and the compilation is performed by a modified version [3] of the natural language question-answering engine C-Phrase [11].

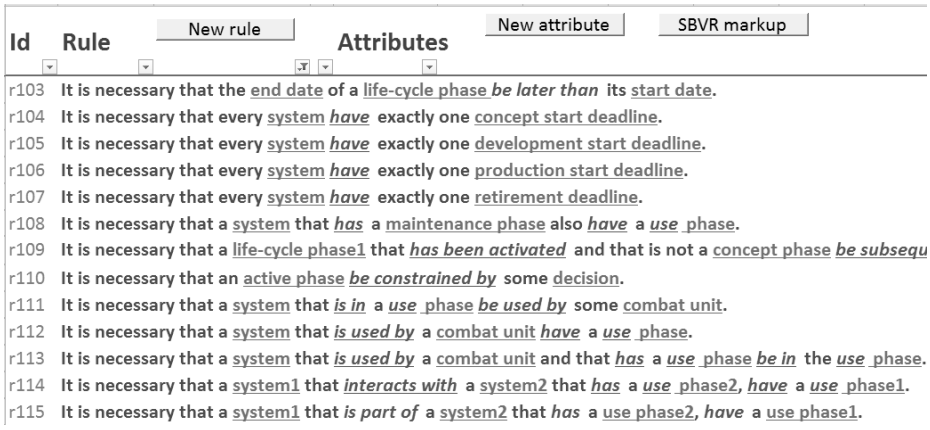


Fig. 3. Some rules in the SK TS SBVR model.

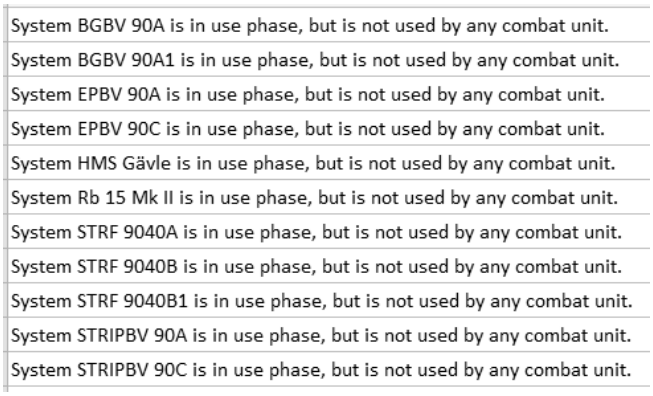


Fig. 4. Violations report produced when executing the rule *"It is necessary that a system that is in a use phase be used by some combat unit."*. Results shown here are from a public demo release of SK TS, not the actual SK TS model, which is confidential.

4.2 Process

The semantic rules should be developed in parallel with the meta-model, as an integral part of the meta-model development itself, with the participation of as broad a range of stakeholders as possible: enterprise architects, problem owners, domain experts, database technicians, etc. As long as the meta-model has not been set in stone, the rule book should equally be considered a living document. The process we propose can be roughly summarized in the following steps, to be repeated indefinitely:

1. Express intended model semantics as rules (either by adding new rules or modifying existing ones). If questions are raised as to what is meant by some of the terms, engage in a discussion, agree upon a meaning, and let the rules reflect the agreement.

2. Execute rules to generate violation reports.
3. Examine the violation reports and figure out to what extent they indicate errors in the model, errors in the meta-model (the rules), or temporary and tolerable incongruence between them.
4. Act accordingly, that is, correct the model, modify the rules, or tolerate.

5 Case study: Two EA projects at FMV

In this section, we evaluate the method just proposed, in terms of the hypotheses that underpin it (as listed in section 3), in the context of two EA modelling projects at FMV.

Throughout 2014 the method was applied to a continuation of the SK TS project and to FM UFS[8], another EA modelling project at FMV. FM UFS describes the capabilities, current and targeted, of military units at different levels of aggregation, what types of tasks the units are expected to perform, and the relationships between capabilities and tasks. Both cases differ somewhat from the ideal application scenario in that the rule development process was initiated well into the EA modelling process.

5.1 Executing rules to find errors (H1)

Executing the rule book for SK TS produced a list of several thousand violations—despite the extensive manual validation and verification that had already been performed on the model. Most of the rule violations trace back to errors in the various data sources that feed the SK TS model, and to inconsistencies between the data sources. As an example, the rule *"It is forbidden that a system that specialises an abstract system has an object-group1 that generalises the object-group2 of the abstract system"* identifies cases where the *object group* hierarchy (imported from one particular data source) and the system hierarchy (imported from another data source) run in different directions of abstraction. More than 10% of all systems in SK TS violate this particular rule; one instance is the system *Grävmaskin hjul* ("Wheeled excavator"), of object group *L151* and with the specialisation *GM HB 19T* ("Wheeled excavator 19T"), of object group *L15*, which is a more general group than *L151*.

Executing rules also uncovered more trivial inconsistencies in SK TS. For instance, the rule *"The start date of a life cycle phase must precede its end date"* identifies several life cycle phases with inconsistent start- and end dates.

In addition to uncovering inconsistencies, executing the SK TS rule book also uncovered incompleteness in the model. The rule *"Every system in use phase must have a decision of use"*, for instance, identified more than a hundred incompletely specified systems, that is, systems that were required to have been

[8] *Försvarsmaktens uppgifts-, förmåge- och systemkartor*, Swedish for "the Armed Forces' maps over tasks, capabilities and systems", with "systems" in this case referring roughly to military units

cleared for use according to FMV policy, but where this use decision had not been entered into the model.

We do not report error rates in the FM UFS model, since at the time of writing, the version of model for which we designed rules has not yet been released, except for a small preview sample. However, FMV plans to use the rule book as part of the validation step prior to the release of the model. Presently, the UFS rule book contains approximately 100 rules.

5.2 Specifying the semantics through rules (H2)

The rule authoring process quickly uncovered ambiguities in both SK TS and FM UFS meta-models—some local, others affecting the entire architectures. Throughout the efforts of formulating rules for both SK TS and FM UFS, questions of interpretation kept coming up for the modelling teams to address. Some were answered promptly because the interpretation, while not being explicitly encoded in any meta-model, was a matter of implicit consensus among the modellers. Some questions triggered discussions about the meaning of terms and how they should relate to each other. Both modelling teams stated that these discussions brought clear value to their respective modelling efforts.

The rule authoring process for FM UFS uncovered ambiguity in each and every relation (both properties and associations) in the FM UFS meta-model. For instance, when discussing the rule *"A combat unit that performs a task that supports a capability, must have the capability."* we found that the relation *task supports capability* was interpreted differently by different members of the modelling team. Some understood it to mean that the task single-handedly realises the capability; others, that the task is one of a possible multitude of tasks that together realise the capability. The ambiguity had not been spotted before, and there was nothing in the meta-model to resolve it.

The process of authoring rules for the SK TS model also uncovered many ambiguities. As an example, when evaluating the rule *"Every system used by a combat unit must be in its use phase."* it was discovered that the relation *system is used by combat unit* is used in two different senses, namely: (1) "the system has been allocated to the combat unit in the defence planning" and (2) "the combat unit has requested the system". The rule in question is valid only under the first interpretation.

We also found that being able to execute the rules under design provided input to that design, and hence helped specify the semantics thus expressed. As an example, executing the rule *"Each system that is used by a combat unit must be in use phase."*, intended to capture cases of erroneously planned use phases, returned instances of systems that, for acceptable reasons (according to the modellers), did not have a use phase registered. The rule was then changed to *"Each system that is used by a combat unit that has a use phase must be in the use phase."*. In this and many more cases then, an iterative rule design-execution process was key to uncovering semantic subtleties that needed to be sorted out.

The previous example highlights an all-encompassing case of semantic underdetermination in the SK TS meta-model that kept coming up during rule

authoring, namely: What is the meaning of absent data? For example, what is the meaning of an absent system life-cycle phase? That no such phase has in fact (yet) been planned? That data about a possible plan has not been entered into the model? Or, that life-cycle phases of systems at this level of abstraction are to be inferred from those of higher or lower level systems?

5.3 Project-specific semantics (H3)

Both SK TS and FM UFS modelling efforts were based on the MODAF[9] architecture framework and its attendant meta-model M3[10], and the modellers were experienced MODAF practitioners. Yet the two projects took quite different approaches to the application of M3.

For reasons of practical expediency, SK TS redefined how concepts such as the life-cycle phases of systems and resource interactions between systems where to be represented in M3 terms. While these redefinitions did not introduce any concepts that couldn't have been represented in an orthodox application of M3, they did shuffle the relationships between terms and referents, invalidating any inheritance of meaning from M3.

FM UFS also departed quite radically from orthodox M3 usage, but in a different way. MODAF is designed to support *capability based planning*, whereby plans are initially expressed at a high level of abstraction ("what"-questions), deferring specifics ("how"-questions) to a later time and lower-level decision-making. M3 thus has *capabilities* at a "strategic" level, that are realised by *nodes* that perform *operational activities* at an "operational" level; *nodes* are then further realised by *resource configurations* at the lowest, "systems" level. In FM UFS, though, *capabilites* and *operational activities*—rebranded as *tasks*—do not express different levels in a realisation hierarchy; instead, the *task* concept is fused into the *capability* concept by being seen as its qualitative component.

5.4 Semantic evolution (H4)

Both SK TS and FM UFS projects kept developing their meta-models—and more generally their semantics—in parallel with architecture modelling. While the SK TS meta-model only underwent minor modifications to its original version, the FM UFS meta-model changed frequently, and at one point, radically.

A notable change to the SK TS meta-model was in the handling of system life-cycle phases. Originally, a system use phase could include, within its time span, a number of maintenance phases. This was changed such that a maintenance phase would end its preceding use phase and then engender a new use phase after its completion. In addition to this modification, and as noted in section 5.2, several semantic decisions where made along the way, triggered by rule development.

The redefinition of the relationship between *capabilities* and *tasks* in FM UFS (described in the previuos section), which was done way into the architecture modelling process, was the most radical semantic shift of the the FM UFS

[9] Ministry of Defence Architecture Framework
[10] MODAF Meta-Model

project. In addition, a number of more specific semantic modifications where made along the way. An example is the relation *combat unit has capability*, whose interpretation was changed from "the combat unit is required by its specification to have the capability" to "the combat unit will realise the capabaility according to the plan". Another example is the rule-of-thumb *"Each combat unit type is realised by at most one combat unit"*, which was enacted a few months into modelling, when it was discovered that this was the pattern of the data being used to populate the model (thus deviations from the pattern could be flagged as possible errors).

5.5 Natural language rules (H5)

Because of the evolving and project-specific nature of the semantics of the architecture modelling projects we observed (as described in sections 5.3 and 5.4), and because, as described in section 5.2, the activity itself of developing and executing rules contributed in a significant and positive way to the semantic evolution, we found it fruitful to have regular discussions, centered around rules, with the modelling teams at FMV. During the 2013 SK TS project, before natural language rules were introduced, resolving ambiguities uncovered through rule authoring was an active initiative on our part—alternating between designing rules, executing them and directing questions at the SK TS modellers. With rules in natural language, however, we found that simply handing over the rule book to the modellers triggered discussions that propelled semantic specification forward with much less active effort required from our part—the modellers became a part of the rule formulation process, rather than just being passive receivers of its output.

Natural language rules also facilitated the participation of business stakeholders outside the modelling teams in the rule authoring process.

6 Conclusions

Formal modelling rules are rarely used in industrial EA-projects, despite their apparent utility and despite mature tool support. In this paper we have reported on their use in two large scale EA-projects at the Swedish Defence Materiel Administration. Both case studies confirmed the utility, by showing that formal modelling rules effectively capture errors missed by manual auditing, and that the rule authoring process uncovers vagueness, imprecision and ambiguity in the meta-model. Moreover, the case studies confirmed a claim often made in the business rules community: that it is easy to engage business architects and other stakeholders in the rule authoring process if rules are formulated in a (controlled) natural language.

It might be argued that engaging business architects and other stakeholders in the rule authoring is unnecessary—why not simply let the formal modelling rules be built in as part of a generic architecture framework or EA-tool that

the business architects can use out of the box? However, and perhaps somewhat surprisingly, the case studies showed that meta-models are project specific—even when the organisation has agreed upon a common architecture framework—and the project specific meta-model evolves along with the architecture model itself. Consequently, modelling rules need to be formulated by the EA-project itself and the rules need to be continually updated during the project, which suggests a need for a natural and accessible rule format.

References

1. Aier, S., Buckl, S., Franke, U., Gleichauf, B., Johnson, P., Närman, P., Schweda, C.M., Ullberg, J.: A survival analysis of application life spans based on enterprise architecture models. In: EMISA. pp. 141–154 (2009)
2. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM 13(6), 377–387 (Jun 1970)
3. Cohen, M., Minock, M.J., Oskarsson, D., Pelzer, B.: Natural language specification and violation reporting of business rules over er-modeled databases. Accepted and to appear in Proceedings of the 18th International Conference on Extending Database Technology (2015)
4. Dam, H.K., Lê, L.S., Ghose, A.: Managing changes in the enterprise architecture modelling context. Enterprise Information Systems (ahead-of-print), 1–31 (2015)
5. Date, C.J.: What not how: the business rules approach to application development. Addison-Wesley Professional (2000)
6. Fischer, R., Aier, S., Winter, R.: A federated approach to enterprise architecture model maintenance. Enterprise Modelling and Information Systems Architectures 2(2), 14–22 (2007)
7. Johnson, P., Ekstedt, M.: Enterprise architecture: models and analyses for information systems decision making (2007)
8. Johnson, P., Ullberg, J., Buschle, M., Franke, U., Shahzad, K.: P2amf: Predictive, probabilistic architecture modeling framework. In: van Sinderen, M., Oude Luttighuis, P., Folmer, E., Bosems, S. (eds.) Enterprise Interoperability, Lecture Notes in Business Information Processing, vol. 144, pp. 104–117. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-36796-0_10
9. Kelly, S., Tolvanen, J.P.: Domain-specific modeling: enabling full code generation. John Wiley & Sons (2008)
10. Lankhorst, M.M.: Enterprise architecture modelling—the issue of integration. Advanced Engineering Informatics 18(4), 205–216 (2004)
11. Minock, M.J.: A step towards realizing codd's vision of rendezvous with the casual user. In: Proceedings of the 33rd international conference on Very large data bases. pp. 1358–1361. VLDB Endowment (2007)
12. Semantics of business vocabulary and rules (sbvr), version 1.2. OMG (2013)
13. Sousa, P., Caetano, A., Vasconcelos, A., Pereira, C., Tribolet, J.: Enterprise architecture modeling with the unified modeling language. Enterprise Modeling and Computing with UML. IGI Global pp. 69–97 (2006)
14. Steen, M.W., Akehurst, D.H., ter Doest, H.W., Lankhorst, M.M.: Supporting viewpoint-oriented enterprise architecture. In: Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International. pp. 201–211. IEEE (2004)