On Domain Modelling and Requisite Variety

Current state of an ongoing journey

Hend*erik* A. Proper[1,2] and Giancarlo Guizzardi[3]

[1] Luxembourg Institute of Science and Technology (LIST), Belval, Luxembourg
[2] University of Luxembourg, Luxembourg
[3] Free University of Bolzano-Bozen, Italy
e.proper@acm.org, giancarlo.guizzardi@unibz.it

Abstract. In the 1950's, W. Ross Ashby introduced the *Law of Requisite Variety* in the context of General Systems Theory. A key concept underlying this theory is the notion of variety, defined as the total number of distinct states of a system (in the most general sense). We argue that domain modelling (including enterprise modelling) needs to confront different forms of variety, also resulting in a need to "reflect" / "manage" this variety. The aim of this paper is to, inspired by Ashby's *Law of Requisite Variety*, explore some of the forms of variety that confront domain modelling, as well as the potential consequences for models, modelling languages, and the act of modelling. To this end, we start with a review of our current understanding of domain modelling (including enterprise modelling), and the role of modelling languages. We then briefly discuss then notion of Requisite Variety as introduced by Ashby, which we then explore in the context of domain modelling.

Keywords: domain modelling · conceptual modelling · requisite variety

1 Introduction

In the context of software engineering, information systems engineering, business process management, and enterprise engineering & architecting in general, many different kinds of models are used. In this paper, we consider each of these kinds of models as being valued members of a larger family of *domain models*.

Domain models have come to play an important role during all stages of the life-cycle of enterprises and their information and software systems, which also fuels the need for more fundamental reflection on domain modelling itself. This includes *the act of modelling*, *the essence of what a model is*, and *the role of (modelling) languages*. Such fundamental topics have certainly been studied by different scholars (e.g. [**?**, **?**, **?**, **?**, **?**, **?**]), as well as by ourselves (see e.g. [**?**, **?**, 2, 3, **?**, 4, **?**, **?**, **?**, 1]). At the same time many challenges remain (see e.g. [4, **?**]).

In this paper, we focus on the challenge of how domain modelling (including enterprise modelling) needs to deal with different forms of variety, also resulting in a need to "reflect" / "manage" this variety. In line with this, we will explore some of the forms of variety that confront domain modelling, as well as the potential consequences for models, modelling languages, and the act of modelling. We will do so from the perspective of the *Requisite Variety*. This concept has been introduced by W. Ross Ashby [**?**], in the context of General Systems Theory, as part of the Law of Requisite Variety, where *variety* refers to the number of states of a system (system in the most general sense).

We see this short paper as part of an ongoing "journey" we undertake, with the aim of deepening our insights into the foundations of domain modelling, mixing our theoretical work and practical experiences in developing (foundational and core) ontologies and domain models, associated modelling languages, frameworks, and methods.

The remainder of this paper is structured as follows. Section 2 starts with a review of our current understanding of domain modelling, while section 3 then complements this with our view on the modelling languages. We then continue, in section 4, by reviewing the Law of Requisite Variety as introduced by Ashby [**?**], and the notion of Requisite Variety in particular. The notion of Requisite Variety is then, in section 5, explored further in the context of domain modelling.

2 Domain Models

Based on general foundational work by e.g. Apostel [**?**], and Stachowiak [**?**], more recent work on the same by different authors [**?**, **?**, **?**, **?**], as well as our own work [**?**, **?**, 1, 2, **?**], we consider a *domain model* to be: An *artefact* that is *acknowledged* by an *observer* to *represent* an *abstraction* of some *domain* for a particular *purpose*. Each of the stressed words in this definition requires a further explanation, and as we will see in section 5, results in different kinds of *variety*.

A model is seen as an *artefact*; i.e. it is something that exists outside of our minds. In our fields of application, this artefact typically takes the *form* of some "boxes-and-lines" diagram. More generally, however, it can, depending on the *purpose* at hand, take different forms including text, mathematical specifications, physical objects, etc.

With *domain*, we refer to "anything" that one can speak / reflect about explicitly. It could be "something" that already exists in the "real world", something desired towards the future, or something imagined. The *observer* observes the domain by way of their senses and / or by way of (self) reflection. What results in the mind of the observer is, what is termed a *conceptualisation* in [2], but also what is termed *conception* in [**?**].

When the modelled *domain* pertains to a part / perspective / aspect of an enterprise, then one can indeed refer to the resulting domain model as an *enterprise model*. This may, include enterprise (wide) data models, enterprise architecture models, etc.

As, it is ultimately the observer who needs to *acknowledge* the fact that the *artefact* is indeed a model of the domain, it actually makes sense to treat *their* conceptualisation / conception of the domain as the de-facto "proxy" for the domain. As such, we should also realise that the *observer* observes the model (as artefact) as well, which therefore also creates a conceptualisation (in their mind) of the modeler. The observer, therefore, needs to validate the alignment between their model-conceptualisation and their domain-conceptualisation, using the *purpose* as alignment criterion.

Models are produced for a *purpose*. In the context of enterprise modelling, [**?**] suggest (at least) seven high-level purposes for the creation of enterprise models: *understand*, *assess*, *diagnose*, *design*, *realise*, *operate* and *regulate*. In specific situations, these high-level purposes will need to be made more specific in terms of, e.g., the need for different stakeholders to *understand*, *agree*, or *commit* to the content of the model [**?**], or for a computer to be able to interpret the model in order to e.g. automatically analyse it, use it as the base of a simulation / animation, or even execute it.

A model is the representation of an *abstraction* of the domain. This implies that, in line with the *purpose* of the model, some (if not most) "details" of the domain are consciously filtered out. For domain modelling, important abstraction flavours are [**?**]: (1) *selection*, where we decide to only consider certain elements and / or aspects of the domain; (2) *classification* (including typing); (3) *generalisation*; and (4) *aggregation*.

As a result, an observer actually needs to harbour (at least) four conceptualisations: (1) a "full" conceptualisation of the domain (as they "see" it), (2) a conceptualisation of the purpose for the model, (3) an abstracted conceptualisation of the domain, (4) a conceptualisation of the artefact that is (to be) the model representing the latter abstraction.

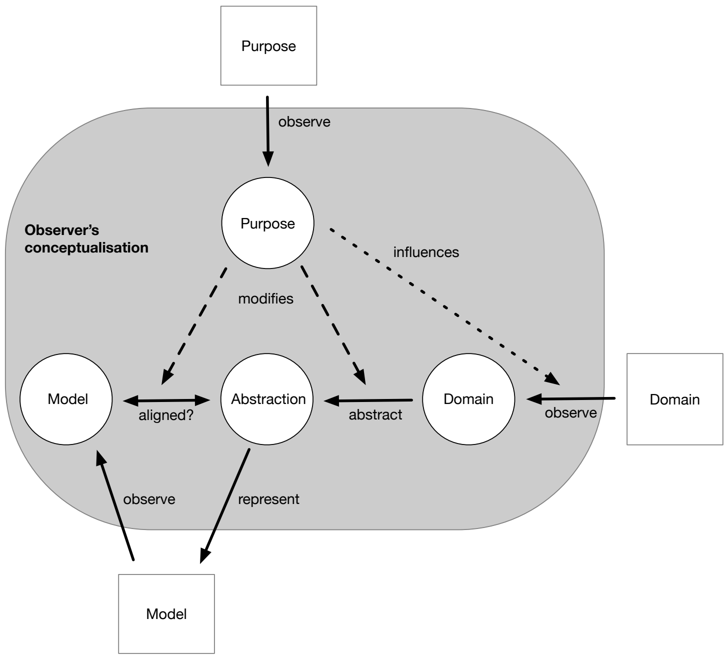

Fig. 1. Conceptualisations involved in domain modelling

The latter has been illustrated in figure 1, where we see how the conceptualisation of the purpose modifies the abstraction, and the alignment between the conceptualisations of the model and the abstraction. The purpose may actually already influence the original observation of the domain. When the model-conceptualisation corresponds to the abstraction-conceptualisation, then the observer would agree that the artefact is a model of the domain for the given purpose. As a consequence, different models may indeed result in the same model-conception, in which case (for the observer) they are equivalent models of the same domain (for the same purpose). If the observer is "the modeller", i.e. the person creating the model, they also need to "shape" the model in such a way that it best matches their *desired* model-conceptualisation.

In line with the above discussion, a domain model should be (the representation of) the abstraction of (the conceptualisation of) a domain. At the same time, for different *computational* purposes, such as the ability to use the model as a base for simulation, computer-based reasoning, execution, or database design, it may be necessary to make "compromises" to the conceptual model. These compromises result in a model that does *not* correspond to (an abstraction of) the original domain. They are essentially models of a "close enough" approximation of (the conceptualisation of) the original domain.

This is where we suggest to make a distinction between *conceptual domain models* and *computational design model* in the sense that a *conceptual domain model* is: "*A model of a domain, where the purpose of the model is dominated by the ambition to remain as-true-as-possible to the original domain conceptualisation*", while a *computational design model* includes compromises needed to support computational design considerations that e.g. support simulation, animation, or even execution of the model.

Note the use of the word *ambition* in the definition of conceptual domain model. We are not suggesting there to be a crisp border between conceptual models and computational design models. However, the word *ambition* also suggest that a modeller / observer, as their insight in a domain increases, should be driven to reflect on the conceptual purity of their conceptualisation and of the resulting model.

Computational design models certainly have an important role to play. However, it is important to be aware of the compromises one has made to the original domain conceptualisation. As such, it is also possible that one conceptual model has different associated computational design models, each meeting different purposes.

3 The Role of Modelling Languages

In its most general form a language identifies the conventions to which the expressions in the language should conform to. In a domain modelling context, these conventions are often equated to a definition in terms of a concrete visual syntax, and a grammar in terms of a set of rules, while the semantics are defined in terms of some mathematical structure (*formal semantics* [**?**]) or an ontological theory (*ontological or real-world semantics* [**?**]). However, style guides, reference models, patterns, etc, can also be seen as part of the set of conventions that define a (modelling) language.

Sometimes, a modelling language comes in the form of a number of connected sub-languages. Typical examples are ARIS [**?**], UML [**?**] and ArchiMate [**?**, **?**], each featuring more specific languages focused on one aspect or layer, as well as (some more, some less) the integration / coherence between these aspects or layers.

We argue that, if a model is represented in some (pre-)defined language, then the (relevant part of the) definition of that language should actually be seen as being a *part* of the model. This also allows us to illustrate the role of the modelling language in the sense of providing a trade off. If, given some purpose, there is a need to represent an abstraction A, and one has a choice between using a language L1 with an elaborate set of conventions (the light grey part), or using a language L2 with only a more limited set of conventions (the dark grey part), then this will lead to a difference in the size of the (situation specific parts of the) model one would still need have to specify. This has been illustrated in figure 2, where we show that the "size" of the two models as a whole remains the same. Of course, the notion of "size" is to be clarified. We will return to this in section 5 when discussing the consequences of the Requisite Variety, as the "size" indeed connects directly to the variety of the model.

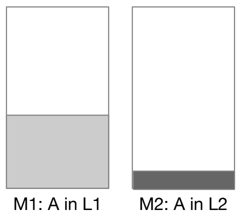

Fig. 2. Trade off between languages with different sizes of their definition

A final consideration is the fact that the conventions which define a modelling language need to reflect the intended set of valid models. Which also means that these conventions need to accommodate all the possible purposes of these intended models. For standardised general purpose languages, such as UML [**?**], BPMN [**?**] or ArchiMate [**?**, **?**], this does lead to tensions, as the set of purposes keeps growing, resulting in a continuous growth of the set of allowable models, thus also triggering a growth in the "size" of the body of conventions defining these languages [**?**, **?**].

As such, the we should also realise that the grey parts in figure 2 amount only to that part of the respective languages that are relevant for the interpretation of the white parts. However, the other parts of the language will also need to be learned by the "users" of the language, or at least be visible as part of the standard.

4 Requisite Variety

The *Law of Requisite Variety* by W. Ross Ashby [**?**] is one of the defining contributions to the field of General Systems Theory and Cybernetics in particular. This law postulates that when a system *C* aims to control / regulate parts of the behaviour of a system *R*, then the variety of *C* should (at least) match the variety of that part of *R*'s behaviour it aims to control; "*only variety can destroy variety*" [**?**, page 2020]. The notion of *Requisite Variety*, which also builds on Shannon's Information Theory [**?**], refers to the variety that is *required* from the controlling system, where *variety* refers to the number of states of a system (system taken in the most general sense).

In the context of the *Law of Requisite Variety* it is also important to clearly understand the scope of the system that would need to be controlled. For example, controlling a car in the sense of getting it into motion and steering it in a certain direction on an empty car park, is quite different from driving a car through busy traffic. The latter *system* clearly needs to deal with a larger variety.

It is also important to realise that the earlier remark "*the variety of C should (at least) match the variety of that part of R's behaviour it aims to control*", is actually directly related to the notion of abstraction as discussed in figure 1. The controlling system *C* does need, in line with its steering goal / purpose, to have access to a relevant abstraction / model of the aspects of *R* it aims to control. More recently, [**?**] formulated this as (stress added by us): "*The analogy of structure between the controller and the controlled was central to the cybernetic perspective. Just as digital coding collapses the* *space of possible messages into a simplified version that represents only the difference that makes the difference, so the control system collapses the state space of a controlled system into a simplified model that reflects only the* ***goals*** *of the controller. Ashby's law does* ***not*** *imply that every controller must model every state of the system but* ***only those states that matter for advancing the controller's goals****. Thus, in cybernetics, the goal of the controller becomes* ***the perspective from which the world is viewed****.*"

In our understanding, domain modelling involves different flavours of variety. In the next section, we will explore these in more detail. The existence of these varieties (as well as complexities) has inspired us to try and follow the logic behind Ashby's *Law of Requisite Variety*, and apply this in the context of domain modelling. In doing so, however, we should indeed not forget that the *Law of Requisite Variety*, is defined in the context of one system controlling (aspects of) another system. As such, it would have been better to refer to it as the *Law of Requisite Variety of systems controlling systems*.

The latter raises three main questions: (1) how to define variety in the context of domain modelling, (2) what is "that" what may need to have requisite variety, and (3) would this indeed result in a law akin to the *Law of Requisite Variety*? In line with the explorative nature of this paper, this paper will certainly not provide the full answer to these questions. In the next section, while discussing (some of the) flavours of variety as we (currently) understand to exist in relation to domain modelling, we will also reflect on these three questions.

5 Requisite Variety in Domain Modelling

In this section, we explore (some of) the flavours of requisite variety we may encounter in the context of domain modelling.

Requisite variety needed to conceptualise a domain – When conceptualising a domain (see figure 1), an observer needs to deal with *variety in* the domain, *uncertainties about* properties of the domain, as well as *complexity of* the domain. The combination of these yields a requisite variety that needs to be met by the observer(s) of the domain that is to be modelled, in the sense that:

- the observer should harbour a conceptualisation in their mind, catering for the variety-in, uncertainties-about, and complexity-of the domain,
- understanding this conceptualisation (e.g. to be able to make an abstraction in line with a modelling purpose) requires a certain "mental state space",
- the later corresponds to the need for variety from the observer; i.e. requisite variety.

Towards future research, it would be beneficial to better qualify, or even quantify, how the variety-in, uncertainties-about, and complexity-of the domain results in a requisite variety that needs to be (potentially) matched by the cognitive ability of the observer.

Residual requisite variety in relation to the model purpose – Driven by the model purpose, there is a need to capture a relevant (but not trivialised) part of the domain. In other words, the observer(s) will need to make an abstraction (see figure 1) of the domain. As the abstraction involves filtering out "details", one would expect that the residual requisite variety needed from the observer, to harbour the abstraction in their mind, is less than the requisite variety needed for the "full" domain conceptualisation.

In line with figure 1, the resulting model will also need to meet the residual requisite variety, in the sense that the latter needs to be captured in terms of the "informational payload" of the model (indeed, also linking back to the information-theoretical [**?**] roots of Ashby's *Law of Requisite Variety*.

The relation between the original requisite variety of a domain, and the residual requisite variety of an abstraction, is also related to the point made in [**?**] regarding the need of a controlling system to have a model of the controlled system, tuned to the steering goals.

Requisite variety trade-offs related to modelling languages – In section 3, we already pointed (see figure 2) at the need to essentially include the (relevant parts of the) definition of the modelling language in a model. The whole of the specified model (the white parts in figure 2) and the parts provided by the language (the gray parts in figure 2) need to match the variety of the abstraction. The border between these, however, depends on the language used.

Of course, when using a "thin" language with a small set of conventions, it may be easy to learn the language, and also easy to create modelling tools that support the language. At the same time, specifying the actual models (the white parts in figure 2) will require more effort than it would cost when using a language with more predefined concepts and conventions. Provided of course, that these (pre-defined concepts and conventions) indeed meet the modelling purpose, and domain, at hand.

Here it is also interesting to refer back to older discussions (in the field of information systems engineering) regarding the complexity of modelling notations. As also mentioned by Moody [**?**], models need to provide some informational payload. A simpler notation might be easier to learn, and easier to look at when used in models, but when it cannot "carry" the needed informational payload, then the more "simpler" notation may actually turn out to be a liability (given that it transfers the modeling complexity to the modeler).

Towards future research, it would also be beneficial to better qualify, or even quantify, how abstraction actually results in a reduction of requisite variety facing observers of a domain, as well as the "informational payload" needed from the model. In addition, in terms of such insights, it would also be possible to more specifically reason about that part of the variety that should preferably by covered by a (domain specific language), and which part should be left to the actual model.

Requisite variety originating from social-complexity – The context in which the model is to be used may involve different stakeholders, uncertainty about their interests, backgrounds, etc. This is where, based on experiences from the IBIS project [**?**], Conklin coins the term social complexity [**?**, **?**].

Social complexity poses a risk on the successful outcome of development projects. As such, it is an aspect (with its variety) of a system (the development project), which would need to be managed. In this case, the original *Law of Requisite Variety* in the sense of *Law of Requisite Variety of systems controlling systems*, seems to apply.

Part of the variety that is due to the social complexity will need to be met / managed by the overall development / engineering process. However, the more finer grained processes / tasks in which domain models are actually created / used, in particular when multiple stakeholders are involved, will also need to deal with this variety. Especially since, in terms of figure 1, they will need to agree on the purpose / goals of the model, the abstraction to be made, and its representation in terms of the model.

A future research challenge would be to, on the one side, once again further qualify or even quantify the involved variety, as well as show how different collaborative modelling strategies [**?**, **?**] may deal with this variety.

6 Conclusion

In this paper, inspired by Ashby's *Law of Requisite Variety*, we explored some of the forms of variety that confront domain modelling, as well as the potential consequences for models, modelling languages, and the act of modelling.

We first provided a review of our current understanding of domain modelling (including enterprise and conceptual modelling), and the role of modelling languages.

Using this as a base, we then explored some of the possible consequences / challenges of the notion of requisite variety in a domain modelling context.

As mentioned in the introduction, we see this paper as part of an ongoing "journey", with the aim of deepening our insights into the foundations of domain modelling. In line with this, we certainly do not claim this paper to be a fully finished work. These are reflections providing a snapshot of our current understanding of these topics. We hope they will trigger debates in the modeling community, which we expect to provide fruitful insights to the next steps of this journey.

Acknowledgements

We would like to thank the anonymous reviewers for their comments on earlier versions of this short paper.

References

1. Guarino, B., Guizzardi, G., Mylopoulos, J.: On the philosophical foundations of conceptual models. Information Modelling and Knowledge Bases XXXI **321**, 1 (2020)
2. Guizzardi, G.: On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. Frontiers in artificial intelligence and applications **155**, 18 (2007)
3. Guizzardi, G.: Theoretical foundations and engineering tools for building ontologies as reference conceptual models. Semantic Web **1**(1, 2), 3–10 (2010)
4. Guizzardi, G.: Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. In: International Conference on Conceptual Modeling. pp. 13–27. Springer (2014)