Do Conceptual Modeling Languages Accommodate Enough Explicit Conceptual Distinctions?

Dirk van der Linden[1,2,3] and Hend*erik* A. Proper[1,2,3]

[1] Public Research Centre Henri Tudor, Luxembourg, Luxembourg
dirk.vanderlinden@tudor.lu
[2] Radboud University Nijmegen, Nijmegen, the Netherlands
[3] EE-Team, Luxembourg, Luxembourg*
e.proper@acm.org

Abstract. In this paper we are concerned with the degree to which modeling languages explicitly accommodate conceptual distinctions. Such distinctions refer to the precision and nuance with which a given modeling concept in a language can be interpreted (e.g., can an actor be a human, an abstraction, or a collection of things). We start by elaborating on the notion of conceptual distinctions, while also providing a list of common modeling concepts and related distinctions that are relevant to enterprise modeling. Based on this, we will then analyze a number of conceptual modeling languages to see whether they accommodate the explicit modeling of (potentially important) conceptual distinctions – that is, whether they have specific language elements to model conceptually distinct entities with. We conclude by discussing what impact our findings may have on the use (and creation) of modeling languages.

Keywords: enterprise modeling, modeling languages, conceptual distinction, conceptual understanding

1 Introduction

Most concepts common to conceptual modeling languages and methods (e.g., goal, process, resource, actor, etc.) can be interpreted in a number of conceptually distinct, yet equally valid, ways. For example, in the context of business processes, one may choose to interpret actors as being human beings who take decisions and execute actions. At the same time, however, interpreting them as being abstract agents or dedicated pieces of hardware might be equally valid in another context. One could also choose to interpret actors as being a collection of things that, together, execute some actions (e.g., an organizational department composed of many employees, a cluster of computers) instead of being a single thing executing an act. Depending on the context of the domain to be

* The Enterprise Engineering Team (EE-Team) is a collaboration between Public Research Centre Henri Tudor, Radboud University, the University of Luxembourg and HAN University of Applied Sciences (www.ee-team.eu)

modeled, the stakeholders and other modelers we interact with, and the goal of the model itself, we often choose among the different possible interpretations. These different interpretations of the same concept can lead to a host of semantic considerations. For example, if an actor is a human being, one can never be as sure that s/he will behave as expected compared to, say, a computer.

It is important that such different interpretations can be modeled distinctly. It would not do well for the overall clarity and semantic quality of a model if we conflate semantically different interpretations (e.g., human beings, abstract entities and material objects) under the same banner (e.g., 'actor') and pretend that they are one and the same thing. Yet, this is often the case with modeling languages. Frequently, the designers of a modeling language define a type (e.g., actor) and allow it to be instantiated with a wide diversity of entities (humans, hardware, abstract and mathematical entities) which have no common ontological basis. Sometimes modeling languages do accommodate (some) of these conceptual distinctions, but then do so only implicitly. That is, in their specification or meta-model they assume a particular interpretation. As such, all instantiations of a model are then implicitly assumed to abide by that interpretation (e.g., all actors in the given model are assumed to be human things, all goals are assumed to be hard goals). An example of a language doing so is the i* specification as found in the Aachen wiki [10], which defines agents (the acting entities) as having "*a concrete physical manifestation*". This implicitly makes it semantically incorrect to use abstractions (e.g., agents as they are commonly understood) and furthermore, perhaps ontologically incorrect to use composite agents – market segments – as the composition itself is not physically manifested.

It is more useful if a modeling language accommodates such conceptual distinctions *explicitly*, to the extent needed in relation to its expected and planned use. That is, instead of relying on the underlying semantics to define every concept they allow (or perhaps require), to use a notation that explicitly encodes information about our interpretation – and do so by providing distinct notational elements for all the important different conceptual distinctions. This can mean for instance, having exclusive (visual) elements to represent such distinct concepts by (e.g., the amount of 'stick puppets' in in ArchiMate actor type denoting whether it is a single actor or a collection of them). This is important from a cognitive point of view as it improves the quality of the notation by ensuring there is no notational homonymy. These points (and more) were argued for by e.g., Moody in his work on a general "physics of notation" [14]. Several modeling languages have been analyzed to estimate their cognitive quality in terms of this framework (e.g. i* [15], BPMN [7], UCM [6], and UML [13]). However, most of these analyses are aimed at the semantics of the (visual) syntax, and forego a more detailed analyses of the semantics of the individual elements of meaning themselves. By this we mean that they analyzed the semantic quality of the formalization of the syntax (i.e., which elements interoperate in what way), but spent less attention to the question what the elements arranged by this syntax actually means to the users of the language (e.g., what *is* this element called 'agent', what thing does it really represent). From a quality perspective, important related issues are *semiotic clarity* (one-to-one correspondence between se-

mantic constructs and graphical symbols) and *perceptual discriminability* (symbols should be clearly distinguishable) [14].

Thus, in this paper we will specifically look at the cognitive quality of a number of modeling languages and methods in terms of the semiotic clarity of their semantic constructs. These constructs can be both visual (for visual notations) and textual (for textual notations), but both require a proper correspondence between semantic constructs and symbols used for them. To do so we will provide an initial (likely non-exhaustive) overview of different aspects of enterprises that are explicitly modeled today, and show to what degree relevant conceptual distinctions can be explicitly modeled in the languages and methods used for them. The goal of this work is not to provide detailed individual analyses of all the languages involved, but to explore whether there is a trend in modeling languages to support enough distinctions or not, and provide advice on basis of that for modeling language use and design in general.

2 Aspects of Enterprises and Associated Languages

Enterprises are large socio-technical systems encompassing many aspects (e.g., business processes, value exchanges, capabilities, IT artifacts, motivations, goals), which themselves are often the domain of specialized (groups of) people. As these models are produced by different people, often using different languages, integration is a vital step in order to have a coherent picture of the enterprise [11]. Ensuring that different conceptual distinctions are modeled explicitly is thus especially important in this context, as much information can be lost in this integration step, leading to enterprise models that are no longer correct or complete in regards to the semantics intended to be expressed (and possibly only done so implicitly) in the models made of each of the distinct aspects. Traditionally processes and goals received a lot of attention in terms of explicit models and dedicated modeling languages and frameworks, while recently more and more aspects are being considered equally as important to deal with. Other aspects such as motivations and goals, value exchanges, deployment and decision making now have dedicated, often formally specified, modeling languages available. This increases the amount of languages (ideally) capable of explicitly supporting conceptual distinctions important to the individual aspects that are in use, but perhaps at the cost of fragmenting the modeling landscape itself. Table 1 gives a brief overview of some current languages and the aspects they are, or can be used for.

3 Conceptual Distinctions for Aspects & Languages

The different aspects that are focused on in enterprise modeling, typically have a number of (not necessarily overlapping) specific conceptual distinctions, which are important to be aware of. For example, a motivational model describing the things to be achieved by an enterprise and the reasons for wanting to achieve

Table 1: A cross-section of aspects of modern enterprises and some modeling languages used, or usable to represent them.

Aspect of an Enterprise	Related languages
Architecture	ArchiMate [24] (1.0, 2.0), ISO/DIS 19440, ARIS
(Business) Processes	BPMN [17], (colored) Petri nets, IDEF3, EPC [1]
Design decision-making	EA Anamnesis [19], NID [5], OMG DMN (proposed, seemingly unfinished)
Deployment of IT artifacts	ADeL [18]
Goals & Motivations	i*, GRL, KAOS [2], TROPOS [8], AMORE [21], ArchiMate [24] 2.0's motivational extension, OMG BMM [16]
Management of IT artifacts	ITML [4]
Strategy & Capability Maps	TBIM [3], OMG BMM [16], Capability Maps [22]
Value exchanges	e3Value [9], REA-DSL [23], VDML (under development)

them is likely to require more detail (and thus fine-grained conceptual distinctions) for what goals are than, say, a model describing the related process structure. Such distinctions can be for instance whether goals absolutely have to be achieved, whether the 'victory' conditions for achieving it are known, whether the goal itself is a physical thing to be attained or not, and so on. On the other hand, a model describing the process undertaken to achieve a certain goal (e.g., bake a pizza) might require conceptual distinctions like whether the actors involved are human entities or not, whether it is one or more actors responsible for ensuring the goal's satisfaction, and so on. Thus, not all conceptual distinctions that are relevant to one aspect (and the modeling language used for them) will be as relevant (and necessary to model explicitly) for other aspects. In order to systematically talk about whether the selected modeling languages accommodate different conceptual distinctions we need both a set of common modeling concepts and a set of distinctions to analyze. We base ourselves on an analysis of modeling languages and methods commonly used in (enterprise) modeling as reported on in [12].

Table 2: This table gives an overview of a number of relevant conceptual distinctions for common modeling. For each of the concepts, we list relevant conceptual distinctions, show what they are useful for, and what languages support modeling them explicitly, might support it, or (where relevant) make a specific implicit interpretation.

Dimension	Useful to ...	Supported by ...
	ACTOR	

human	Distinguish between actors that can be more fickle than pure rational agents.	BPMN through the explicit use of a 'Human Performer' resource type, VDML does contain a 'Person' subtype of Actor which is specified to be human, but does not distinguish in the visual notation between types of Actors.
composed	Distinguish whether an actual entity acts or whether a group of them does, which impacts responsibility judgments for actions	ArchiMate, TROPOS via 'composite Actor', somewhat as well with differentiation between 'role' and 'position', e3Value somewhat through differentiation between actor and market segments, VDML distinguishes between an 'actor' being a singular participant, and modeling 'collaboration' or 'participant' as potentially multiples.
material	Know whether an actor physically interacts with the world (and can thus be affected by it directly – think hardware vs. software)	i* assumes that an agent is an actor "*with a concrete physical manifestation*" (iStar Wiki)
intentional	Know whether an actor is considered an explicit part of a system, i.e., is expected to act or not on certain things, in contrast to actors from outside the systems scope which may act but were not regarded or thought of to do so	Implicit in most languages, mentioned as such in TBIM, depending on interpretation could be argued to be explicit in OMG BMM with differentiation between internal and external influencer.
specific	Knowing whether an actor is a specific thing (i.e., an instantiation) or a general thing (i.e., a role)	Supported by some (e.g., ArchiMate), through type/instantiation dichotomy, explicit in TBIM by the claim that an agent "*represents a concrete organization or person*" ArchiMate, implicit in e.g., e3Value and RBAC by automatic use of roles (types).
	EVENT	
intentional	Distinguish between events that should, or will happen given a set of circumstances, and events that happen (seemingly) unprovoked.	Arguably explicitly supported by BPMN through the use of 'None' type triggers for Start Events.

	GOAL	
composed	Distinguish between complexity level of goals, i.e., whether they are an overarching strategy or directly needed goals.	TBIM explicitly models composite goals as 'business plan' types, implicit in some other languages focused on strategy/tactics (e.g., OMG BMM).
material	Distinguish between objects and their representations, i.e., is the goal to achieve an increment in the integer on a bank account, or to hold an n amount of currency.	
necessary	Distinguish between goals that have to be attained and those that should.	
specific	Distinguish between goals for which the victory conditions are known and not, i.e., hard vs. soft goals.	Most goal modeling languages/methods/frameworks (e.g., i*, GRL, KAOS, AMORE) support this explicitly.
	PROCESS	
composed	Distinguish between black (closed, singular) and white (open, composed) boxes.	Arguable either way for BPMN with the use of pools, which can function as black boxes, however, those do not allow for linking sequence flow to it, and are thus self-contained.
intentional	Know whether they are part of an intended strategy or something that has to be dealt with (i.e., negative environmental processes)	
specific	Know whether the structure is (supposed to be) clear (deterministic) or not (fuzzy).	
	RESOURCE	
natural	Know whether a resource requires a 'fabrication' process.	Somewhat related, TBIM explicitly models resource types as being either animate or not.
human	Know whether resources can act on their own and produce issues, e.g.., be unreliable, not always generate the same outcomes	
material	Distinguish between objects and their representations, i.e., whether a given resource a collection of paper and ink blobs or the information contained within them.	Explicit in ITML through the use of hardware/software dichotomy.

	RESTRICTION	
natural	Distinguish between restrictions we cannot do anything about and those we can.	
intentional	Distinguish between restrictions we stipulate from those that arise holistically (whether good or bad).	Some languages implicit, e.g., EA Anamnesis, and BPMN through use of 'Potential Owner'.
necessary	Distinguish restrictions that can be broken from those that cannot.	(supported by some GPML, e.g., ORM 2.0).
specific	Distinguish restrictions for which we know when they are broken and not.	
	RESULT	
natural	Know whether a result requires some kind of 'fabrication' process	
material	Distinguish between an object and its representation, i.e. whether the physical pizza or the status update in the IS saying a pizza was baked is the result of a given step in the pizza making process.	
specific	Know whether a result is (supposed to be) clear (deterministic) or not (fuzzy).	Arguably supported in BPMN through the use of 'None' type End Events.

4 Discussion

Around half of the conceptual distinctions we analyzed were explicitly supported by at least one modeling language, with some cases being arguable either way. Languages used for specific aspects do seem to explicitly accommodate some basic (and often widely accepted) necessary conceptual distinctions. For example, the de facto used language for process modeling, BPMN, has explicit support for differentiating between human and non-human actors, which can be important to know for critical steps in a process. Most modeling languages used for motivations and goals also accommodate the distinction between goals with well-specified victory conditions and those with vague or unknown conditions by means of separate hard and soft-goal elements. These explicit distinctions in the notation are likely correlated with the conceptual distinctions being widely accepted as important and having become part of the basic way of thinking. However, taken overall, there does not seem to be a consistent or systematic

pattern behind what language explicitly accommodates (or lacks) which conceptual distinctions.

As such, there are a number of conceptual distinctions for which we found no explicit support by any modeling languages. For example, we found no support for explicitly modeling goals and results as being material things. It also did not seem possible to explicitly model goals as being a logical necessity in the investigated languages. The distinction whether results were things that naturally occurred or fabricated was also not supported. When it comes to processes we found no support to model them explicitly as being intentional, and distinguishing between specific (i.e., well-defined) processes and processes more fuzzy in their structure. Modeling resources as being humans was also not supported, while this is likely not an unthinkable interpretation – effective management of 'human resources' being important for large enterprises. Finally, we found no explicit support for modeling restrictions as naturally occurring and specific things. We will discuss some of these distinctions in more detail.

4.1 Some unaccommodated conceptual distinctions

Surprisingly, we found no explicit support for differentiating between goals with varying levels of necessity and obligation. While many common methodologies (e.g., the MoSCoW technique of dividing requirements into must, should could, and would haves) call for such distinctions, many modeling languages conflate them all into a single kind of goal. Arguably in certain aspects it would make sense to make an implicit choice, as in e.g., process modeling it is necessary for certain steps in a flow to be reached before the flow continues, which can be seen as an analog to logically necessary goals. However, goal models in dedicated languages seem to not make this distinction, even though there is a strong focus on differentiating between hard and soft-goals, which seem correlated with different levels of necessity (e.g., one cannot as certainly rely on a soft-goal to be achieved compared to a hard-goal, especially for mission critical goals).

Another seemingly unaccommodated distinction is the necessity of restrictions, that is, whether some restriction (e.g., a rule, principle, guideline) is an alethic condition that cannot be broken or whether it is not and thus can be broken. While in the context of enterprise modeling there is a strong differentiation of terminology used for different kinds of normative restrictions that can be considered breakable, or at least not strictly enforceable (e.g., principles, guidelines, best practice), these often seem to be used outside of modeling languages in their own approaches – e.g., architecture principles [20]. It seems problematic that many languages used for aspects of enterprises, and languages used to describe the actual enterprise architecture like ArchiMate do not have explicit notational support for these different kinds of restrictions. Many models that are analyzed a posteriori (e.g., when they are integrated in other models, and the original modelers are no longer involved or available) then become difficult to interpret, as the notation of different kinds of restrictions can be ambiguous and lead to situations where it is not clear whether a restriction can be relaxed or not. Surprisingly the only language that seems to support this conceptual

distinction is ORM (in particular version 2), which supports the explicit modeling of restrictions as being either alethic or deontic conditions through its visual notation.

Thus, it seems necessary to stimulate a move towards more explicit focus on (formalization of) the semantics of the elements of meaning of modeling languages. The lack of coverage for some of the distinctions shown in Table 2 makes it clear that more work on extending the specification of relevant languages with the ability to explicitly distinguish between these different conceptual understandings. Given the existence of a large number of different dialects of modeling languages sometimes only differing slightly (e.g., i*, GRL, TROPOS for goal modeling), it seems that supporting many different conceptual distinctions in a single notation would be welcomed by many.

5 Conclusion and Future Work

We have discussed the importance of explicitly modeling conceptual distinctions and analyzed a number of modeling languages to investigate what kind of distinctions they support. We showed that, while some conceptual distinctions are explicitly supported by relevant modeling languages, there are still a large amount of potentially relevant distinctions that are not accommodated, or implicitly interpreted in a specific way by modeling languages. We proposed that research should be done regularly to keep up to date with conceptual distinctions deemed relevant and important by modelers and stakeholders alike. Our future work will involve investigations into which distinctions are deemed important.

Acknowledgements. This work has been partially sponsored by the *Fonds National de la Recherche Luxembourg* (`www.fnr.lu`), via the PEARL programme.

References

1. van der Aalst, W.M.: Formalization and verification of event-driven process chains. Information and Software technology 41(10), 639–650 (1999)
2. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. 20, 3–50 (April 1993)
3. Francesconi, F., Dalpiaz, F., Mylopoulos, J.: Tbim: A language for modeling and reasoning about business plans. Tech. Rep. DISI-13-020, University of Trento. Department of Information Engineering and Computer Science (May 2013)
4. Frank, U., Heise, D., Kattenstroth, H., Ferguson, D., Hadar, E., Waschke, M.: ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management. In: Proc. of the 9th OOPSLA workshop on DSM (2009)
5. Gal, Y., Pfeffer, A.: A language for modeling agents' decision making processes in games. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. pp. 265–272. ACM (2003)
6. Genon, N., Amyot, D., Heymans, P.: Analysing the cognitive effectiveness of the ucm visual notation. In: Kraemer, F.A., Herrmann, P. (eds.) System Analysis and Modeling: About Models, Lecture Notes in Computer Science, vol. 6598, pp. 221–240. Springer Berlin Heidelberg (2011)

7. Genon, N., Heymans, P., Amyot, D.: Analysing the cognitive effectiveness of the bpmn 2.0 visual notation. In: Malloy, B., Staab, S., Brand, M. (eds.) Software Language Engineering, Lecture Notes in Computer Science, vol. 6563, pp. 377–396. Springer Berlin Heidelberg (2011)
8. Giunchiglia, F., Mylopoulos, J., Perini, A.: The tropos software development methodology: processes, models and diagrams. In: Agent-Oriented Software Engineering III, pp. 162–173. Springer (2003)
9. Gordijn, J., Akkermans, J.: Value-based requirements engineering: Exploring innovative e-commerce ideas. Requirements engineering 8(2), 114–134 (2003)
10. Grau, G., Horkoff, J., Yu, E., Abdulhadi, S.: i* guide 3.0. Internet (August 2007), http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=67
11. Lankhorst, M.M.: Enterprise architecture modelling–the issue of integration. Advanced Engineering Informatics 18(4), 205 – 216 (2004)
12. van der Linden, D.J.T., Hoppenbrouwers, S.J.B.A., Lartseva, A., Proper, H.A.: Towards an investigation of the conceptual landscape of enterprise architecture. In: T. Halpin et al. (ed.) Enterprise, Business-Process and Information Systems Modeling, LNCS, vol. 81, pp. 526–535. Springer Berlin Heidelberg (2011)
13. Moody, D., Hillegersberg, J.: Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the uml family of diagrams. Lecture Notes in Computer Science 5452, 16–34 (2009), cited By (since 1996) 0
14. Moody, D.L.: The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering 35, 756–779 (2009)
15. Moody, D., Heymans, P., Matuleviaius, R.: Visual syntax does matter: Improving the cognitive effectiveness of the i* visual notation. Requirements Engineering 15(2), 141–175 (2010), cited By (since 1996) 0
16. Object Management Group: Business motivation model (bmm), version 1.1. Internet (2010), http://www.omg.org/spec/BMM/1.1/
17. Object Management Group: Business Process Model and Notation (BPMN) FTF Beta 1 for Version 2.0. Internet (2010), http://www.omg.org/spec/UML/2.0/
18. Patig, S.: Modeling deployment of enterprise applications. In: Proc. CAISE Forum, LNBIP 72. pp. 253–256 (2010)
19. Plataniotis, G., de Kinderen, S., Proper, H.A.: EA Anamnesis: Towards an approach for Enterprise Architecture rationalization. In: Printing, S. (ed.) Proceedings of the The 12th Workshop on Domain-Specific Modeling (DSM12). ACM DL (2012)
20. Proper, H.A., Greefhorst, D.: The Roles of Principles in Enterprise Architecture. In: Proper, H.A., Lankhorst, M.M., Schoenherr, M., Barjis, J., Overbeek, S. (eds.) Trends in Enterprise Architecture Research. Lecture Notes in Business Information Processing, vol. 70, pp. 57–70. Springer Berlin Heidelberg (2010)
21. Quartel, D., Engelsman, W., Jonkers, H., Van Sinderen, M.: A goal-oriented requirements modelling language for enterprise architecture. In: Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International. pp. 3–13. IEEE (2009)
22. Scott, J.: Business Capability Maps – The missing link between business strategy and IT action. Architecture & Governance 5(9), 1–4 (2009)
23. Sonnenberg, C., Huemer, C., Hofreiter, B., Mayrhofer, D., Braccini, A.: The rea-dsl: a domain specific modeling language for business models. In: Advanced Information Systems Engineering. pp. 252–266. Springer (2011)
24. The Open Group: ArchiMate 2.0 Specification. Van Haren Publishing (2012)