On the accommodation of conceptual distinctions in conceptual modeling languages*

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, e.proper@acm.org
[2] Radboud University Nijmegen, the Netherlands
[3] EE-Team, Luxembourg, Luxembourg†

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. On basis of these findings we then further discuss how to ensure such different distinctions are captured in created models, how to know which of them to support in modeling languages, and where existing methods fall short. We conclude by discussing what impact our findings may have on the use (and validation) of modeling languages.

1 Introduction

The creation of conceptual models, in particular when they represent a part of an enterprise, involves a myriad of stakeholders and informants, each of which has its own background and views on the domain that is modeled. As a result, conceptual modeling is considered to be an inter-subjective activity [PS01, Moo05], where modeling "ideally" boils down to the representation of a shared social reality. 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. This is partially reflected in the already large, and diverse, amount of terminology used by modeling

*An initial version of this paper appeared as "Dirk van der Linden, Henderik A. Proper. Do conceptual modeling languages accommodate enough explicit conceptual distinctions? Short Paper Proceedings of the *6th IFIP WG 8.1 working conference on the Practice of Enterprise Modeling (PoEM 2013)*, CEUR-WS, 2013"

†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)

languages and their users. 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 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. If an actor is seen as a composite entity (i.e., an organizational department) the issue of the responsibility of the actions the department takes comes into play as well, since in the end, a concrete, specific person needs to be held (legally and/or socially) responsible. These considerations hold in the case of many of the common concepts. For example, interpreting a resource as an immaterial thing (e.g., using a piece of information as a resource) will require one to carefully distinguish between the actual resource and its physical representation (e.g., the collection of paper and ink blobs).

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 [GHYA07], 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. Many researchers have proposed methods and frameworks to analyze the degree to which languages are complete in this sense [GW04, BJWW09], often ontological in nature (e.g., UFO [GW10], Bunge-Wand-Weber [WW90] and their applications [FL03, GHW03]), although some have been criticized as being poorly suited when applied to the information systems domain [WK05]. A major effort on this topic was undertaken by e.g., Moody in his work on a general "physics of notation" [Moo09]. Several modeling languages have been analyzed to estimate their cognitive quality in terms of this framework (e.g. i* [MHM10], BPMN [GHA11], UCM [GAH11], and UML [MH09]). 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 grammar or 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 semantic constructs and graphical symbols) and *perceptual discriminability* (symbols should be clearly distinguishable) [Moo09]. This issue comes into play more clearly with domain-specific modeling languages than it does with general-purpose languages like UML, ER or ORM (even though these languages were originally designed for specific purposes like software and database engineering) because they have more native specialized semantic elements (i.e., types) to represent the important aspects from their domain by. It is thus important that these domain-specific languages have the ability to explicitly express important semantic distinctions that might arise in needed specific situations.

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 on basis of that argue what kinds of research and engineering efforts are needed to deal with optimizing the conceptual completeness of modeling languages. Hence the initial purpose of our work is to gain a deeper (empirical) understanding of the issues and challenges involved, rather than 'jumping' to the creation/suggestion of mechanisms to possibly deal with them. Therefore, the work reported on in this paper specifically looks 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. We do so in the context of Enterprise Modeling, as there are many conceptually different aspects of enterprises that need to be modeled (e.g., goals, processes, rules). These are often captured in specialized (domain-specific) languages, which reflect the different conceptual landscapes of each aspect, and should thus be a good source of finding different kinds of accommodated conceptual distinctions. 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 rest of this paper is structured as follows. In Section 2 we introduce the different as-

pects and modeling languages we selected for our investigation. In Section 3 we introduce the conceptual distinctions and analyze to what degree they are supported by the selected languages. We discuss our findings and the consequences for modeling (languages) in Section 4, and conclude with needed future work in Section 5.

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 [Lan04, KBJK03, DDB05]. 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.

This increased amount of focus on specific aspects has thus, amongst other factors, led to a plethora of modeling languages, methods and frameworks. Some were proposed or designed solely by academia, some invented in industry, most of them having different focus and purpose. Some aspects have a large amount of dedicated languages differing only slightly in their actual notation or specification (e.g., as evidenced by the large amount of overlap between the notations used in goal modeling such as i*, GRL, KAOS, TROPOS, etc.). In order to have an overview of modeling languages from a wide array of subjects, we selected a number of languages, both languages proposed in academia, and languages widely used in industry for the different aspects listed in Table 1. We chose these specific languages in order to have a diverse amount of of languages and notations, while not necessarily ensuring an exhaustive list of all aspects or methods and languages. Instead, our focus was on ensuring we included languages covering as many aspects as possible, so as to be able to investigate potential issues with the accommodation of conceptual distinctions as broadly as possible. Modeling languages that are typically used for general purpose modeling such as UML, ER and ORM were not included as these languages themselves do not (and by design perhaps should not) contain specialized semantic constructs for domain concepts (e.g., goal, process). A part of the selection for Table 1 was based on the languages integrated into the Unified Enterprise Modeling Language

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 [The12] (1.0, 2.0), ISO/DIS 19440, ARIS
(Business) Processes	BPMN [Obj10b], (colored) Petri nets, IDEF3, EPC [vdA99]
Design decision-making	EA Anamnesis [PdKP12], NID [GP03], OMG DMN (proposed, seemingly unfinished)
Deployment of IT artifacts	ADeL [Pat10]
Goals & Motivations	i*, GRL, KAOS [DvLF93], TROPOS [GMP03], AMORE [QEJVS09], ArchiMate [The12] 2.0's motivational extension, OMG BMM [Obj10a]
Management of IT artifacts	ITML [FHK$^+$09]
Strategy & Capability Maps	TBIM [FDM13], OMG BMM [Obj10a], Capability Maps [Sco09]
Value exchanges	e3Value [GA03], REA-DSL [SHH$^+$11], VDML (under development)

(UEML) [ABH$^+$10], which incorporates ARIS, BMM, BPMN, colored petri nets, GRL, IDEF3, ISO/DIS 19440, KAOS, and some diagram types from UML.

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 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 mod-

eling languages and methods commonly used in (enterprise) modeling as reported on in [vdLHLP11], which resulted in a set of concepts common to most modeling languages, and a set of conceptual dimensions which were often found to distinguish between specific interpretations. From this we take a set of common concepts shared between most languages: ACTORS, EVENTS, GOALS, PROCESSES, RESOURCES, RESTRICTIONS and RESULTS. For each of these concepts we look at whether one of the following conceptual distinctions is relevant for that concept, namely whether something is considered to: naturally occur (*natural*), be human (*human*), be a single thing or composed of multiple (*composed*), be intentional or unintentional (*intentional*), be a logical necessity (*necessary*), be physically existing (*material*), and whether something is well specified (*specific*). The result of this step was the basis for Table 2. We started with a full list including each possible conceptual distinction for each concept, resulting in many different possible points of analysis. We then went through all the concepts and removed the distinctions that we deemed less relevant or interesting (e.g., whether a process is human, whether a result is intentional, and so on), ending up with a list table in which each concept has a number of potentially relevant distinctions. We then show for each concept-distinction combination why it can be useful to be aware of this distinction, and what modeling language supports doing so.

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).

Table 2: (cont.)

	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. Surprisingly[1] ArchiMate's motivational extension does not.
	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).

[1] Given that it was derived from AMORE, which does explicitly support soft/hard goal distinctions

Table 2: (cont.)

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 [CB94] 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 not to 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 [PG10]. 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.

Another conceptual distinction that is typically not accommodated by most languages is whether something is material or not. In particular, the material status of resources is often defined in a conceptually ambiguous way. For example, in TROPOS, resources are stated to be "*physical or informational entities*", which makes it difficult to know whether a modeled resource is the actual 'object' (e.g., some information) or its representation (e.g., a collection of paper and ink blobs that represents that information). It is important to be aware of this distinction as this has consequences for the way in which the resource can be interacted with, and in what way it can be manipulated, and possibly consumed. For example, if we have a process in which a human actor performs a certain task for which they need clear instructions, we can see those instructions as being a vital resource. Modeling them as the physical representation – a paper printout of the instructions – means that this specific resource is only available to one actor. On the other hand, if we model it as the actual informational object, it is available to more than just one actor at a time. Furthermore, when a resource is material, it also has the possibility of being consumed.

For example, when we model the process of baking a pizza, some of the resources involved (i.e., the ingredients) are consumed. It is important to be aware of this, as that means it is necessary to keep track of stock levels, and perhaps optimization thereof via e.g., system dynamics models.

Finally, a conceptual distinction that is not explicitly accommodated by many languages is whether an actor is a human being or not. BPMN was the only language we analyzed which explicitly supports it by having a notional element 'Human Performer' which is only used for human actors (albeit called a resource in the BPMN jargon). It is important to be aware of this, especially when responsibility comes into play, which is for instance done in KAOS models, by making some agent responsible for some goals. However, the *actual* responsibility for any given thing cannot, from a legal and social perspective be placed on a non-human entity. At the end of the day (or chain of responsibility), there is always a person held responsible (and accountable) for some given action. In the case of software engineering, for example a programmer is held responsible for the downtime caused by bugs, in the case of a building collapsing after a summer breeze the architect is held responsible for not properly analyzing the environment and soil conditions, and so on. When responsibility is modeled, it thus seems prudent to know whether an actor is the actual responsible party or whether it defers its responsibilities to a different, human entity. Another important aspect of human beings is that they are not necessarily rational and reliable. Thus, when a given task or process depends on a specific human actor, it is quite possible that the process is not performed as well as needed, or at all. As such, knowing that a process involves human actors, a certain level of fault tolerance and redundancy would be needed. Conflating human actors with non-human actors makes it far more difficult to know where this is necessary, and could thus lead to models (and predictions made with them, e.g., efficiency or execution time of a process) not holding true to the real world situation.

4.2 Consequences

The overall lack of explicitly accommodated conceptual distinctions (of which there might be more than just those we discussed) in many modeling languages are especially relevant for enterprise modeling. It makes it much more difficult to ensure that integrated models are valid (or complete) representations of the semantics intended by the original modelers, as sometimes these modelers simply lack the notational elements to express important semantics. While it is possible to 'simply' denote this information by annotating the models with extra text, it would be a more ideal solution if modeling languages supported these distinctions. Furthermore, while some languages do offer explicit notational support, their specification or meta-model does not necessarily enforce correct use of these elements (e.g., ArchiMate does not enforce correct distinction between composite and singular actors). There are many languages we analyzed which have an implicit interpretation of some of the conceptual distinctions, sometimes specific (e.g., i*'s handling of agents as having a concrete physical manifestation) sometimes vague (e.g., TROPOS' handling of resources), which further complicates matters, as this interpretation of the language might not be the

interpretation a modeler wishes to take for a given context. The fact that some languages have semantics which are considered to remain vague (e.g., GRL [HSD06], i* [LFM11]) only adds to this. Most languages seem to have a well-defined and formalized semantics of the syntax, while lacking much, if any, formalization of the semantics of the elements of meaning themselves (e.g., EPC [Kin04]).

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 is needed on extending the specification of relevant languages with the ability to explicitly distinguish between these different conceptual understandings. This could, and perhaps should, be done in accordance with the actual practitioners in the field, by investigating what conceptual distinctions are important for them, and what they need to be able to explicitly model, instead of solely relying on analyzing languages with pre-existing reference material like Bunge-Wand-Weber or UFO. It is important to not do this just once, but keep up to date with the changing conceptual distinctions that the practitioners and stakeholders have in order for our modeling languages to stay relevant and capable of representing to the real world. 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.

4.3 Needed next steps

One of the more important things that has to be done in order to deal with the issue of lacking conceptual distinctions in modeling languages, from a research point of view, is to ensure a detailed insight into what distinctions are conceptually relevant and important to actual users of modeling languages (i.e., *modelers*) and a created model's end-users (i.e., *stakeholders*). This might seem to be in contrast to what would intuitively be important to find out: whether particular modeling languages accommodate enough conceptual distinctions. However, until we become aware of what is actually important to accommodate, it would not make sense to analyze and criticize a modeling language's quality in this regard. Thus, we should focus on doing empirical work based on finding out what entities (and with which properties they manifest) are vital to modelers and stakeholders for the typical domains they model, and in doing so determine what the conceptual needs are for domain-specific languages for particular domains (e.g., that goal modeling languages need to at least explicitly distinguish between hard and soft goals, goals that have to be achieved versus goals that ought to be achieved, and so on).

Much existing work, both research, and methods actually used to improve the conceptual landscape of modeling languages lack this particular *personal* aspect – they tend to focus on postulating a priori or analyzing only language specifications. While the (cognitive) mapping approach that many of these frameworks (e.g., Bunge-Wand-Weber or UFO) use in their analysis of the ontological completeness of a modeling language, they do so on basis of data that is in itself not directly based on the actual people involved in the model-

ing process. The mapping approach (e.g., ensuring that each conceptually distinct element is represented by a distinct element in the language) itself is the correct way to do this, but the data for the conceptual elements needs to be tailored to the specific people from the domain. This means that if we wish to analyze whether a goal modeling language is conceptually or ontologically complete that we need to figure out which things are important for the modelers and stakeholders to represent, and only then continue to the mapping approach and determine whether the language does that correctly and completely. This is important for a number of points. Firstly, many of the ontologies used for such mapping exercises are either solely or predominantly upper ontologies (i.e., the more abstract conceptual elements and properties), which makes them less suited to speak about the conceptual completeness of a domain-specific language, as with such a language one should expect more lower ontology level conceptual elements to appear in the language. Where such lower ontologies exist for particular domains, they can of course be used if found to be an up-to-date representation of the conceptual needs of users in that domain. Furthermore, many other mapping approaches (or integration efforts such as UEML) rely on analyzing existing text corpuses or language specifications. As such specifications are not necessarily a correct or complete representation of how the language is actually used or abused, one cannot safely say that all the conceptual needs of a particular domain's users can always be found in the specification of the modeling languages they use.

Finally, it is important to be aware of the constantly changing conceptual landscapes that modelers and stakeholders operate in. While a particular ontology might practically be static (not being significantly updated in years), the concepts that become important to modelers and stakeholders can appear and change much more quickly. For this reason alone we should focus our efforts on a repeating empirical effort to elicit such conceptual needs through research efforts. It is also important to keep in mind that in doing this, we should not try to solve fundamental issues to do with the conceptualizations and representations people have (e.g., *is* information actually a non-physical entity?), but 'merely' elicit represent them as accurately as we can.

This can be done by working with, and investigating the conceptual understanding of modelers and stakeholders through experiments and observations adopting proven and well-used protocols from cognitive science and (psycho)linguistics. While there are many discussions on the degree of conceptual information that we can discover through the use of words as stimuli (see e.g., [MAG+11]), it is commonly accepted that finding the edges between concepts (i.e., where concepts would be considered distinct) can be done. Examples of such experiments that can be done are for example categorization studies and feature elicitation experiments (e.g., [BC87, Est03]). Categorization studies can reveal in detail both the structure and typicality of particular concepts and to what degree certain elements are considered members. For example, the answer to the question whether a human being is an actor gives us information whether it is considered an actor, but also whether that judgment was made discretely (it being absolutely or absolutely not a member of the category actor), or whether it was made in a graded fashion (a human being being somewhat of an actor). Especially in this later case eliciting the features people associate with such concepts becomes interesting, as we can determine what the typical interpretation for the concept is, what interpretations are also commonly used and accepted, and which

interpretations are only considered poor examples.

For example, when it comes to the concept actor we might find that there are many graded judgments in an initial categorization experiment (i.e., many terms are considered only actors to a certain degree). If we then use a large group of modelers to elicit features we might find such things like “is human”, “is autonomous”, “part of a group”, “responsible for its actions”, and so on. Investigating afterwards how typical, or common each of these features are for the concept can give us a ranking for (sets of) features, which will show us the most common (and conceptually distinct) interpretations. We might for instance find that the two most occurring sets of features are that an actor is “an autonomous human being responsible for its own actions”, and that it is “an physical machine making decisions”. Based on that empirical data we can then deduce at least that modeling languages involving actors should explicitly distinguish between human and non-human actors.

We are currently in the stage of performing a large-scale study employing these methods (some preliminary results having been published in [vdL13]) in which we aim to figure out the feature structure of the common modeling concepts used in Table 2. After this step we will investigate the typicality of all these possible features. In doing so, we will produce both an overview of all (conceptually relevant) features that modelers and stakeholders might need in order to represent their domains correctly, and more importantly: an analysis of how typical or common (and thus important) such features are to the concepts. With such data gathered, a ranking list of what kind of conceptual distinctions must, should and ideally would be supported by a modeling language can then be systematically produced. Performing such work for different specialized groups (e.g., business process modelers, goal modelers) and repeating it on regular intervals should lead to a situation where we do not only have useful methods to ensure that a modeling language is conceptually and ontologically complete, but that they can be based on tried and proven relevant personal insights as well.

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 on active practitioners 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 a broader empirical investigation into which conceptual distinctions are deemed important by practitioners. Based on these latter insights, we would then be in a position to develop/select better strategies to deal with the challenges of conceptual distinctions.

Acknowledgements.

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

References

[ABH+10] Victor Anaya, Giuseppe Berio, Mounira Harzallah, Patrick Heymans, Raimundas Matulevicius, Andreas L. Opdahl, Hervï Panetto, and Maria Jose Verdecho. The Unified Enterprise Modelling Language–Overview and further work. *Computers in Industry*, 61(2):99 – 111, 2010. Integration and Information in Networked Enterprises.

[BC87] RobinA. Barr and LeslieJ. Caplan. Category representations and their implications for category structure. *Memory & Cognition*, 15(5):397–418, 1987.

[BJWW09] Andrew Burton-Jones, Yair Wand, and Ron Weber. Guidelines for empirical evaluations of conceptual modeling grammars. *Journal of the Association for Information Systems*, 10(6), 2009.

[CB94] Dai Clegg and Richard Barker. *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[DDB05] Dursun Delen, Nikunj P. Dalal, and Perakath C. Benjamin. Integrated modeling: the key to holistic understanding of the enterprise. *Communications of the ACM*, 48(4):107–112, 2005.

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20:3–50, April 1993.

[Est03] Z. Estes. Domain differences in the structure of artifactual and natural categories. *Memory & cognition*, 31(2):199–214, 2003.

[FDM13] Fabiano Francesconi, Fabiano Dalpiaz, and John Mylopoulos. TBIM: A Language for Modeling and Reasoning about Business Plans. Technical Report DISI-13-020, University of Trento. Department of Information Engineering and Computer Science, May 2013.

[FHK+09] Ulrich Frank, David Heise, Heiko Kattenstroth, Donald Ferguson, Ethan Hadar, and Marvin Waschke. ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management. In *Proc. of the 9th OOPSLA workshop on DSM*, 2009.

[FL03] Peter Fettke and Peter Loos. Ontological Evaluation of Reference Models Using the Bunge-Wand-Weber Models. In *AMCIS*, volume 384, pages 2944–2955, 2003.

[GA03] Jaap Gordijn and JM Akkermans. Value-based requirements engineering: Exploring innovative e-commerce ideas. *Requirements engineering*, 8(2):114–134, 2003.

[GAH11] Nicolas Genon, Daniel Amyot, and Patrick Heymans. Analysing the Cognitive Effectiveness of the UCM Visual Notation. In Frank Alexander Kraemer and Peter Herrmann, editors, *System Analysis and Modeling: About Models*, volume 6598 of *Lecture Notes in Computer Science*, pages 221–240. Springer Berlin Heidelberg, 2011.

[GHA11] Nicolas Genon, Patrick Heymans, and Daniel Amyot. Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation. In Brian Malloy, Steffen Staab, and Mark Brand, editors, *Software Language Engineering*, volume 6563 of *Lecture Notes in Computer Science*, pages 377–396. Springer Berlin Heidelberg, 2011.

[GHW03] Giancarlo Guizzardi, Heinrich Herre, and Gerd Wagner. On the General Ontological Foundations of Conceptual Modeling. In Stefano Spaccapietra, SalvatoreT. March, and Yahiko Kambayashi, editors, *Conceptual Modeling (ER) 2002*, volume 2503 of *Lecture Notes in Computer Science*, pages 65–78. Springer Berlin Heidelberg, 2003.

[GHYA07] Gemma Grau, Jennifer Horkoff, Eric Yu, and Samer Abdulhadi. i* Guide 3.0. Internet, August 2007.

[GMP03] Fausto Giunchiglia, John Mylopoulos, and Anna Perini. The tropos software development methodology: processes, models and diagrams. In *Agent-Oriented Software Engineering III*, pages 162–173. Springer, 2003.

[GP03] Ya'akov Gal and Avi Pfeffer. A language for modeling agents' decision making processes in games. In *IFAAMAS'03*, pages 265–272. ACM, 2003.

[GW04] Andrew Gemino and Yair Wand. A framework for empirical evaluation of conceptual modeling techniques. *Requirements Engineering*, 9(4):248–260, 2004.

[GW10] Giancarlo Guizzardi and Gerd Wagner. Using the Unified Foundational Ontology (UFO) as a Foundation for General Conceptual Modeling Languages. In *Theory and Applications of Ontology: Computer Applications*, pages 175–196. Springer, 2010.

[HSD06] Patrick Heymans, Germain Saval, and Gautier Dallons. A template-based analysis of GRL. *Advanced Topics in Database Research, Volume V*, 5:124, 2006.

[KBJK03] Harald Kuehn, Franz Bayer, Stefan Junginger, and Dimitris Karagiannis. Enterprise Model Integration. In *E-Commerce and Web Technologies*, volume 2738 of *Lecture Notes in Computer Science*, pages 379–392. Springer Berlin / Heidelberg, 2003.

[Kin04] Ekkart Kindler. On the semantics of EPCs: A framework for resolving the vicious circle. In *Business Process Management*, pages 82–97. Springer, 2004.

[Lan04] Marc M. Lankhorst. Enterprise architecture modelling–the issue of integration. *Advanced Engineering Informatics*, 18(4):205 – 216, 2004.

[LFM11] Lidia López, Xavier Franch, and Jordi Marco. Making explicit some implicit i* language decisions. In *Conceptual Modeling–ER 2011*, pages 62–77. Springer, 2011.

[MAG+11] Barbara C. Malt, Eef Ameel, Silvia Gennari, Mutsumi Imai, and Asifa Majid. Do words reveal concepts? In *Proceedings of the 33rd Annual Conference of the Cognitive Science Society*, pages 519–524, 2011.

[MH09] D. Moody and J.V. Hillegersberg. 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.

[MHM10] D.L. Moody, P. Heymans, and R. Matuleviaius. Visual syntax does matter: Improving the cognitive effectiveness of the i* visual notation. *Requirements Engineering*, 15(2):141–175, 2010.

[Moo05] D.L. Moody. Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, 55(3):243–276, 2005.

[Moo09] Daniel L. Moody. The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, 35:756–779, 2009.

[Obj10a] Object Management Group. Business Motivation Model (BMM), Version 1.1. Internet, 2010.

[Obj10b] Object Management Group. Business Process Model and Notation (BPMN) FTF Beta 1 for Version 2.0. Internet, 2010.

[Pat10] Susanne Patig. Modeling Deployment of Enterprise Applications. In *Proc. CAISE Forum, LNBIP 72*, pages 253–256, 2010.

[PdKP12] G. Plataniotis, S. de Kinderen, and H.A. Proper. EA Anamnesis: Towards an approach for Enterprise Architecture rationalization. In Sheridan Printing, editor, *Proceedings of the The 12th Workshop on Domain-Specific Modeling (DSM12)*. ACM DL, 2012.

[PG10] Erik Proper and Danny Greefhorst. The Roles of Principles in Enterprise Architecture. In *Trends in Enterprise Architecture Research*, volume 70 of *LNBIP*, pages 57–70. Springer, 2010.

[PS01] Anne Persson and Janis Stirna. Why Enterprise Modelling? An Explorative Study into Current Practice. In Dittrich et al., editor, *Advanced Information Systems Engineering*, volume 2068 of *LNCS*, pages 465–468. Springer Berlin, 2001.

[QEJVS09] Dick Quartel, Wilco Engelsman, Henk Jonkers, and Marten Van Sinderen. A goal-oriented requirements modelling language for enterprise architecture. In *EDOC'09*, pages 3–13. IEEE, 2009.

[Sco09] J. Scott. Business Capability Maps – The missing link between business strategy and IT action. *Architecture & Governance*, 5(9):1–4, 2009.

[SHH+11] Christian Sonnenberg, Christian Huemer, Birgit Hofreiter, Dieter Mayrhofer, and Alessio Braccini. The rea-DSL: a domain specific modeling language for business models. In *CAiSE'11*, pages 252–266. Springer, 2011.

[The12] The Open Group. *ArchiMate 2.0 Specification*. Van Haren Publishing, 2012.

[vdA99] Wil MP van der Aalst. Formalization and verification of event-driven process chains. *Information and Software technology*, 41(10):639–650, 1999.

[vdL13] Dirk van der Linden. Categorization of Modeling Language Concepts: Graded or Discrete? In YanTang Demey and Herve Panetto, editors, *On the Move to Meaningful Internet Systems: OTM 2013 Workshops*, volume 8186 of *Lecture Notes in Computer Science*, pages 743–746. Springer Berlin Heidelberg, 2013.

[vdLHLP11] D. J. T. van der Linden, S. J. B. A. Hoppenbrouwers, A. Lartseva, and H. A. Proper. Towards an Investigation of the Conceptual Landscape of Enterprise Architecture. In T. Halpin et al., editor, *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *LNCS*, pages 526–535. Springer Berlin Heidelberg, 2011.

[WK05] B. Wyssusek and H. Klaus. On the foundation of the ontological foundation of conceptual modeling grammars: the construction of the Bunge-Wand-Weber ontology. In J. Castro and E. Teniente, editors, *Proceedings of the CAiSE '05 Workshops*, volume 2, pages 583–593, 2005.

[WW90] Yair Wand and Ron Weber. Mario Bunge's Ontology as a formal foundation for information systems concepts. *Studies on Mario Bunge's Treatise, Rodopi, Atlanta*, pages 123–149, 1990.