Enhancing comprehension of ontologies and conceptual models through abstractions

C. Maria Keet

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
keet@inf.unibz.it

Abstract. In addition to the Database Comprehension Problem, where diagrammatic conceptual data models are too large for a modeller or domain expert to comprehend or manage, an Ontology Comprehension Problem is emerging. Formal ontologies are, however, more amenable to automated abstractions to improve understandability. Three ways of abstraction are defined with 11 abstraction functions that use foundational ontology categories. Usability of the abstraction functions is enhanced by associating the functions with a basic framework of levels and abstraction hierarchy, thereby facilitating querying and visualizing ontologies.

1 Introduction

Information systems are rapidly increasing in size and complexity. This is caused by, among others, database integration through global schemas (integrated conceptual models) or large ontologies, such as [23–25], resulting from company mergers or the desire to link scientific databases on the Internet. Moreover, within the scope of the Semantic Web, large formal ontologies are being developed mainly by domain experts who often have not had any training in logics and therefore rely on more convenient GUIs as interface to the logical theories, such as the diagrammatic renderings with OntoViz and GrOWL. While this makes ontology development more accessible to domain experts, it has the major drawbacks that large diagrammatic logical theories do not fit onto one computer screen or printable figure and the GUIs are poor in ontology browsing and querying options, thereby making the full contents of the ontology difficult to comprehend; hence, in addition to the Database Comprehension Problem [2, 8, 9, 19], there is an analogous Ontology Comprehension Problem. Compared to most conceptual data models, ontologies have at runtime a formal underpinning integrated with it, which therefore makes it easier to develop solutions for managing user interaction with those ontologies. An obvious solution is to avail of abstractions as mechanism to go from finer to coarser-grained information to avoid detailed representations that are not of interest for the user who *chooses to ignore* undesired aspects whilst maintaining both levels of detail in the background in the system (see also [20] and references therein). Extant proposals for abstractions differ along three dimensions: language to which it is applied, methodology, and semantics of what one does when abstracting. Concerning the latter, we identify

three distinct ways of performing abstraction, which propagates to the types of abstraction functions needed to manage ontologies and conceptual data models. In addition, we use foundational ontological categories to type the functions, thereby facilitating consistent implementation and reusability. Third, we introduce more precisely the notion of abstraction level and abstraction hierarchy as an additional framework, which eases computation in particular regarding querying that can be executed as pre-processing step for visualisation of sections of ontologies. Section 2 contains a summary of extant approaches of abstraction and section 3 contains the main solutions we propose. We conclude in §4.

2 Related works

A range of approaches to abstractions are described by [2, 5, 9, 19]. The earlier works on theories of abstraction differ along three dimensions—language, axioms, and rules—and, as summarised by [5], concern topics such as abstraction for planning, reduction of search, and logical theories; we are interested in the latter. [2, 9, 19] provide overviews that focus on abstractions for conceptual models and ontologies, which are logical theories that have essential graphical 'syntactic sugar' for improved usability from the perspective of the domain expert. We illustrate main issues with extant approaches.

Manual abstraction is used for UML modularisation, EER clustering and 'abstraction hierarchies', *e.g.*, [8, 13, 19, 21, 27], that have the drawbacks that it is a laborious, intuitive, not scalable, and *ad hoc* method. Concerning methodology for abstractions, [2] introduced heuristics to simplify large ORM conceptual models, but they are tailored to ORM only and thus not directly applicable to other knowledge representation languages [9]. Limited syntax-focused formalisation of abstraction using Local Model Semantics of context reasoning is proposed by [4], which address taxonomic generalisation (subsumption), which concurs with [3, 22, 16]. This, however, ignores a crucial aspect of collapsing sub-processes into a grander process and overloads their *abs* function, thereby in itself abstracting away the finer details of the process of abstraction. Syntax abstraction augmented with semantics was investigated by [14, 16], extending [6, 7]. [14] addresses the important notion of "folding" formally: *e.g.*, for a biology domain, the catalytic reactions and proteins involved in the Second messenger system collapse into that one entity type, and Cell contains (modular) subsystems (*e.g.*, [18]). The approaches can be structured according to topic and implementation foci and the nature of the abstraction operations (see [9]). These solutions exhibit three main problems: abstraction focuses only on the contents of a level, thereby lacking a surrounding *framework*; a general abstraction function *abs* does neither reveal *what* it is abstracting nor *how*; extant proposed solutions are mainly theoretical and not developed for or assessed on its potential for reusability and scalability. In the next section, we introduce a basic framework, three types of abstraction, and abstraction functions for both basic and complex folding operations, respectively, so that abstractions become scalable, are unambiguous to implement, and amenable to automation.

3 Abstractions

The various abstractions have different purposes at the conceptual level, regardless implementation issues. What is abstracted away depends on multiple factors; we focus on the type of abstraction, the procedure of (consecutive steps of) abstraction, and on what type of representation/model the abstraction is performed. The types are depicted in Fig.1, and they are defined as follows.

Definition 1 (R-abs) *An abstraction is an* R-ABS *i the more-detailed type φ abstracts into its related parent type ψ.*

Definition 2 (F-abs) *An abstraction is a* F-ABS *i a more-detailed theory T_0 (with $\varphi_1 ... \varphi_n$ entity types, relations, and constraints among them) abstracts into a single related parent type ψ in T_1 that is distinct from any φ_i.*

Definition 3 (D-abs) *An abstraction is a* D-ABS *i a more-detailed theory T_0 with $\varphi_1 ... \varphi_n$ entity types, relations, and constraints abstracts into theory T_1 with $\varphi'_1 ... \varphi'_m$ entity types, relations, and constraints, and $m < n$ through* deletion *of elements either from T_0 to obtain T_1 or from T_0's user interface.*

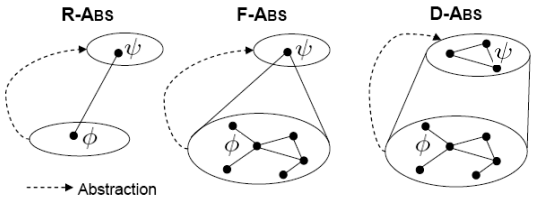

Fig. 1. Three conceptually distinct kinds of abstraction operations. R-ABS: the relation is remodelled as a function; F-ABS: folding multiple entities and relations into a different type of entity; D-ABS: hiding or deleting semantically less relevant entities and relations.

[4, 5, 7, 9, 16] mention 'levels' of abstraction, depicted in Fig.1 with the ovals, but to the best of our knowledge, the notion of level has not been specified and used specifically for abstractions. To be able to manage better both the abstraction functions and complex base- & simple theory resulting from an abstraction, we define an abstraction level as follows.

Definition 4 (Abstraction level) *Given a base theory T_0 and a simpler theory T_1 into which T_0 is abstracted, an* abstraction level*, denoted with λ_i, is the surrounding frame that contains T_i, which form a tuple $\langle \Lambda, \mathcal{T} \rangle$, where $\lambda_i \in \Lambda$ and $T_i \in \mathcal{T}$, and $\lambda_0 \preceq \lambda_1$.*

Observe that for any logical theory T_i (*i.e.*, ontology or conceptual data model), obviously the individual entity types, relations, and constraints are in T_i's level of abstraction λ_i and considered to be accessible for abstraction individually as

well. To put levels to use, we need a function to retrieve the level where a theory or its entity types reside, $\mathit{labs} : \mathcal{T} \rightarrow \Lambda$, which is analogous to such functions for granularity [11]; *e.g.*, $\mathit{labs}(T_1) = \lambda_1$ (with $\lambda_1 \in \Lambda$), which thus also holds for any type in T_1 (if $\varphi_1 \in T_1$ then $\mathit{labs}(\varphi_1) = \lambda_1$). Last, to improve abstraction functions as introduced in [4, 14] and make a significant step toward their usability for ontologies, we will avail of several ontological categories from DOLCE foundational ontology [15]. DOLCE has an OWL version—the ontology language for the Semantic Web—, is comprehensive and used across subject domains (see for an overview: http://www.loa-cnr.it/DOLCE.html). In particular, we will use DOLCE's endurant ED for entity types (OWL classes), perdurant PD, and PT for particular as top-type that subsumes any other type (`owl:Thing` in OWL). With these preliminaries, we can proceed to the abstraction functions.

3.1 Basic and compound abstraction functions

The basic abstractions for R-ABS, (1-5), are listed in Table 1. They are straightforward relation-turned-into-function along a hierarchy in the formal ontology or conceptual data model, with the two distinctions that functions (1-5) are typed with ontological categories and have additional constraints to relate the entity types to their abstraction level. The functions conform to the main relations in the OBO Relation Ontology for bio-ontologies and the latest results on types of part-whole relations [10, 17]. Note that abstraction for spatial containment (4) refers to both the type and region it occupies [10], and an addiitonal abstraction function for *proper* parthood may be useful for bio-ontologies; that is, an abs_{ppo}: $PT \rightarrow PT$ where $\mathit{abs}_{ppo}(\varphi) = \psi$, $\mathit{ppart\_of}(\varphi, \psi)$, $\mathit{labs}(\varphi) = \lambda_i$, $\mathit{labs}(\psi) = \lambda_j$, and $\lambda_i \prec \lambda_j$ hold. The basic functions are trivially extensible for other ontological categories and recurring relations in domain ontologies even if one were to use a different foundational ontology, which can be of use in an implementation. Considering DOLCE foundational ontology [15], several examples are given to illustrate some of the possibilities to refine the abs functions further.

Example. i) Refinement of abs with EDs: Abstract non-agentive physical objects ($NAPO$) or amounts of matter (M) into amounts of matter (M), using (sub-)quantities; *e.g.*, Air and its M-part Oxygen or its $NAPO$-parts the types of molecule such as O_2 and CO_2. Abstracting social agents (SAGs) like citizens into society (SC), locusts into swarm, and so forth for entities denoted with collective nouns and their members. ii) Refinement with PDs: Mapping processes (PRO) into one event (EV), *e.g.*, Running into Marathon. $\diamondsuit$

The last basic operation, abs_{d1}, covers one of the two functions for D-ABS, where some type is deleted, which is primarily applicable to conceptual data models; thus, $\varphi \in T_0$, $\varphi \notin T_1$, with $\mathit{labs}(T_0) = \lambda_0$, $\mathit{labs}(T_1) = \lambda_1$, $\lambda_0 \prec \lambda_1$, and $T_1 \subset T_0$. Note that when an attribute that is the sufficient condition of φ is removed, then the deletion implies $\varphi \subset \phi$, hence a taxonomic abstraction (abs_{isa}).

With these basic abstraction functions, we have covered the most widely used relations to construct hierarchies in 'simple' formal ontologies. Compound abstractions are required to manage comprehension and visualisation of complex formal ontologies and to enable abstractions for formal conceptual data models.

Table 1. List of basic and compound abstraction functions.

	Abstraction	Constraints; comment
(1)	$\text{abs}_{isa} : PT \to PT$	$\text{abs}_{isa}(\varphi) = \psi$, $\varphi \subseteq \psi$, $\mathit{labs}(\varphi) = \lambda_i$, $\mathit{labs}(\psi) = \lambda_j$, $\lambda_i \prec \lambda_j$; sub-supertype (class) abstraction
(2)	$\text{abs}_{po} : PT \to PT$	as (1), but $\mathit{part\_of}(\varphi, \psi)$; part into its whole
(3)	$\text{abs}_{in} : PD \to PD$	as in (1), but $\mathit{involved\_in}(\varphi, \psi)$; Abstract a part-process into the whole-process
(4)	$\text{abs}_{ci} : ED \to ED$	as in (1), but $\mathit{contained\_in}(\varphi, \psi)$; Abstract a smaller contained type into larger type
(5)	$\text{abs}_{pi} : ED \to PD$	as in (1), but $\mathit{participates\_in}(\varphi, \psi)$; Abstract an endurant into a perdurant
(6)	$\text{abs}_{d1} : PT \to \emptyset$	$\mathit{labs}(\varphi) = \lambda_i$, $\lambda_i \prec \lambda_j$; Abstract a type into 'nothing', deleting it from the theory
(7)	$\text{abs}_{p1} : ED \times ED \to ED$	$\text{abs}_{p1}(\varphi, \psi) = \phi$, where $\mathit{labs}(\varphi) = \lambda_i$, $\mathit{labs}(\psi) = \lambda_i$, $\mathit{labs}(\phi) = \lambda_j$, and $\lambda_i \prec \lambda_j$ Abstract two endurants into another endurant
(8)	$\text{abs}_{p2} : PD \times PD \to PD$	as in (7); Abstract two perdurants into a perdurant
(9)	$\text{abs}_{p3} : ED \times PD \to ED$	as in (7); Abstract an endurant and a perdurant into an endurant
(10)	$\text{abs}_{p4} : ED \times PD \to PD$	as in (7); Abstract an endurant and a perdurant into a perdurant
(11)	$\text{abs}_{d2} : ED \times Q \to ED$	as in (7), but $\lambda_i \preceq \lambda_j$, $\varphi = \phi$; Remove an attribute

The compound abstractions address F-ABS and D-ABS, folding two entities or types in λ_i into a simpler entity or type in λ_j where $\lambda_i \prec \lambda_j$. They are summarised in Table 1, (7-11). $\mathit{abs}_{p1}(\varphi, \psi) = \phi$, has, *e.g.*, Blood cell (a $NAPO$) and Plasma (an M) as direct parts of Blood (an M). One could add more specific functions that satisfy (8), such as an abs'_{p2}: $PRO \times ST \to EV$ for abstracting Running and Being thirsty into Marathon. For F-ABS, we have (9) that combines perdurants and endurants into 'systems' that are endurants, such as Second messenger system that is composed of enzymes and catalysis processes, and (10) has its analogue with EER clustering and abstractions [8, 19]; for example Orders in the fact type "Customer Orders Book", where the ordering process involves, a.o., Billing, Paying, Supplier, and Shipment. Both functions require a constraint, being that the input types have to be related to each other, ensuring that no two arbitrary types are folded, but ones that are related so that a connected subset of T_i is folded into a type in T_j, *i.e.*, upon firing abs_{p3} or $\mathit{abs}_{p4} \geq 1$ times for elements in T'_i, where $T'_i \subseteq T_i$, then T'_i is abstracted into ϕ where $\phi \in T_j$.

Constraint 1 (folding) *For each φ, ψ where $\mathit{abs}_{p3}(\varphi, \psi) = \phi$ or $\mathit{abs}_{p4}(\varphi, \psi) = \phi$, $\mathit{labs}(\phi) = \lambda_j$, there must be either i) a predicate p such that $p(\varphi, \psi) \in T_i$ that is contained in λ_i or ii) $\varphi = ED$, $\psi = p'$ and $\forall x(ED(x) \to \exists y(p'(x, y)))$ in T_i.*

Here, as with deletion (abs_{d1}), compositionality of the theory is important, which is a desirable feature from a computational viewpoint [5, 16]; from the perspective of a domain expert it is debatable, because some details in the logical theory really may be undesirable to develop tractable systems biology simulations

and making ontologies usable for ontology-guided applications [18, 24, 25]. For F-ABS this can be effectively managed with the current abstraction functions in conjunction with the levels. ϕ in T_j (in λ_j) is not a 'new' entity, but can be represented as element in the encompassing theory T_Δ for the whole system that is the union of $T_0, ..., T_n$; hence, soundness and completeness can be preserved.

With the last main abstraction function, abs_{d2}, we address the remainder of D-ABS. Suppressing details from the interface to a logical theory can already be done through toggle features, which lets the user select displaying more or less relations, attributes, and so forth, like with the OntoViz plugin [26] for the Protégé ontology development tool. This can be formally defined with $abs_{d2}(\varphi, \psi)$, where attribute ψ (a quality Q in DOLCE) folds away. But for (11), because nothing changes to the underlying theory, we have $labs(\phi) = \lambda_j$, $\phi = \varphi$ and $T_i = T_j$ and $\lambda_i \preceq \lambda_j$. More functions can be defined for the other to-be-hidden elements analogous to abs_{d2}. A software developer may want to label the abstraction not ϕ but φ' as approximation for φ, thereby communicating to the user that incomplete information is shown in the GUI and that further exploration (up to the base theory T_i) is possible. This simple hiding breaks down with theories of over about 100 entity types, and may need to be augmented with a generalisation [9] of Campbell *et al.*'s [2] rules and their weights; a.o., hiding based on prioritization where, *e.g.*, existential quantification takes precedence over all other constraints, and identification is more important than a non-key attribute. Thus, for a large diagram—with logical theory in the background—one can find out what the *important* elements are. Reformulation of the rules, which are written by [2] as "if φ then keep it" instead of "if φ then abstract it away", is feasible; *e.g.*, "*Rule 3*: *[keep] non-leaf object types*" can be rewritten for parthood relations, where $\neg(part\_of = R)$, as "*Rule 3′*: *if* $part\_of(\varphi, \psi) \wedge \neg R(\varphi, \phi)$, *then* $abs_{po}(\varphi) = \psi$."; hence, the functions proposed here are compatible with [2, 9] and are applicable also to other conceptual modelling languages and ontologies.

3.2 Abstraction hierarchy

By using abstraction functions (1-11) and Definition 1 for abstraction level, one can create abstraction hierarchies. We define an abstraction hierarchy as follows.

Definition 5 (Abstraction hierarchy) *Let $\mathcal{T}$ be set of theories, $\mathcal{F}$ denote a set of abstraction functions, and Λ the set of levels obtained from using $abs_i \in \mathcal{F}$ on a theory $T_0 \in \mathcal{T}$, then an* abstraction hierarchy *$H \in \mathcal{H}$ is the ordered sequence of levels $\lambda_0, ..., \lambda_n \in \Lambda$, with $n > 1$, obtained from firing $abs_i \geq 1$ times successively on $T_0, T_1, ..., T_{n-1}$ such that $\lambda_0 \preceq \lambda_1 .... \lambda_{n-1} \preceq \lambda_n$ and $labs(T_0) = \lambda_0 ... labs(T_n) = \lambda_n$ hold.*

For purposes of understandability on what the system does as well as ease of implementation, we have restricted the abstraction hierarchy to one that is obtained by firing only *one type of abstraction function* to create each hierarchy. This definition differs with the k-level abstraction hierarchy from Knoblock *et al.* [12], because they differ in both the scope and usage of abstraction: planning vs. improving comprehensibility of large formal ontologies and conceptual models.

3.3 Toward implementation

Foundational aspects of abstraction generally suffer from the trade-off to make a workable implementation: theory is there to guide implementation but for various reasons, such as computational complexity, usability, and relative importance, is not always strictly adhered to. The basic abstraction functions for R-ABS, however, have straightforward mappings to recursive queries with an additional clause for the ontological category or type of relation. This is supported in, *e.g.*, ontology query languages such as XQuery and nRQL and database query languages SQL and StruQL for ontologies that are stored in a database like the FMA and GO [24, 25]. The compound abstraction functions require engineering work in CASE tools in addition to the recent results by [19, 27]. Regarding F-ABS abstractions and any combination thereof for stepwise more elaborate folding, they are useful for, *e.g.*, coordinated UML-like modularisation and the 'black box' software modules in biology and ecology [27, 18]. Provided one uses a formal version of UML (*e.g.*, [1]), one can marry the functions with the UML-as-logical-theory, yet have user-friendly interfaces like provided with CASE tools such as Rational Rose. Such CASE tools also have UML packages, which can be reused as diagrammatic support for abstraction levels and the hierarchies constructed with them. With the indexing of abstraction levels, one can keep track of what is abstracted into what more easily, and let a user select (query), say, 3 levels more abstract in one go instead of step-wise clicking in the GUI. For instance, from Hepatic Macrophage (a type of cell) in λ_7 at once to the organ it is part of in λ_3 (which is the Liver), although behind the scene this involves recursively going up the abstraction hierarchy with firing abs_{po} 3 times.

4 Conclusions

Three conceptually distinct ways of abstraction were identified, consisting of remodelling a relation between finer- and coarser-grained entities as a function (R-ABS), folding multiple entities and relations into a different type of entity (F-ABS), and hiding or deleting less relevant entities and relations (D-ABS). Six basic and five complex abstraction functions were introduced, which use foundational ontological types for unambiguous specification and are easily extensible. Abstraction level and abstraction hierarchy were defined, thereby providing a means for consist use of the functions and quick cross-level navigation in applications. By having the abstraction functions at the conceptual level and their corresponding formalisation, it simplifies understanding, provides space for extensions with more abstraction functions, and makes them usable and reusable across implementations of formal ontologies and conceptual data models. Thereby abstraction is scalable and straightforward to implement as queries over the ontology or conceptual data model or methods in software applications. We are currently investigation in more detail the interplay between abstraction and granularity.

References

1. Berardi, D., Calvanese, D., De Giacomo, G. Reasoning on UML class diagrams. AI (2005) 168(1-2): 70-118.
2. Campbell, L.J., Halpin, T.A., Proper, H.A.: Conceptual Schemas with Abstractions: Making flat conceptual schemas more comprehensible. DKE (1996) 20(1): 39-85.
3. Degtyarenko, K., Contrino, S.: COMe: the ontology of bioinorganic proteins. BMC Structural Biology (2004) 4:3.
4. Ghidini, C., Giunchiglia, F.: A semantics for abstraction. TR DIT-03-082, University of Trento, Italy. 2003.
5. Giunchiglia, F., Villafiorita, A., Walsh, T.: Theories of abstraction. AI Communications (1997) 10(3-4): 167-176.
6. Giunchiglia, F., Walsh, T.: A theory of abstraction. AI (1992) 57(2-3):323-389.
7. Hobbs, J.R.: Granularity. In: Proc. of IJCAI85, 1985, 432-435.
8. Jaeschke, P., Oberweis, A., Stucky, W.: Extending ER Model Clustering by relationship clustering. In Proc. of ER'93. Arlington, Texas (1993).
9. Keet, C.M.: Using abstractions to facilitate management of large ORM models and ontologies. In Proc. of ORM'05. LNCS 3762: 603-612.
10. Keet, C.M.: Part-whole relations in Object-Role Models. In Proc. of ORM'06. LNCS 4278: 1116-1127.
11. Keet, C.M.: A taxonomy of types of granularity. IEEE Conference on Granular Computing (GrC2006), Atlanta, USA. IEEE Xplore (2006) 1: 106-111.
12. Knoblock, C.A., Tenenberg, J., Yang, Q.: Characterizing abstraction hierarchies for planning, in: Proc. of AAAI '91, AAAI Press (1991): 692-697.
13. Lind, M.: Making sense of the abstraction hierarchy. Cognitive Science Approaches to Process Control (CSAPC99), Villeneuve d'Ascq, France 21-24 September (1999).
14. Mani, I.: A theory of granularity and its application to problems of polysemy and underspecification of meaning. In: Proc. of KR'98, 245-255.
15. Masolo, C., Borgo, S., Gangemi, A., Guarino, N. and Oltramari, A.: Ontology Library. WonderWeb Deliverable D18, v1.0, 31-12-2003.
16. Pandurang Nayak, P., Levy, A.Y.: A semantic theory of abstractions. In: Proc. of IJCAI'95. San Mateo: Morgan Kaufmann (1995) 196-203.
17. Smith, B., Ceusters, W., Klagges, B., Köhler, J., Kumar, A., Lomax, J., et al.: Relations in biomedical ontologies. Genome Biology (2005) 6:R46.
18. Sontag, E.D.: Some new directions in control theory inspired by systems biology. Systems biology (2004) 1(1): 9-18.
19. Tavana, M., Joglekar, P., Redmond, M.A.: An automated entity-relationship clustering algorithm for conceptual database design. Information Systems (2007), EPub ahead of print: 4-8-2006.
20. Tzitzikas, Y., Hainaut, J-L. On the visualization of large-sized ontologies. In: Proc. of AVI2006, 99-102.
21. Yu, X, Lau, E., Vicente, K.J., Carter, M.W.: Toward theory-driven, quantitative performance measurement in ergonomics science: the abstraction hierarchy as a framework for data analysis. Th. Issues in Ergonomics Sci. (2002) 3(2): 124-142.
22. Zhang, J., Silvescu, A., Honavar, V.: Ontology-Driven Induction of Decision Trees at Multiple Levels of Abstraction. TR ISU-CS-TR 02-13, Iowa State University. 2002.
23. Cell Cycle Ontology. http://www.cellcycleontology.org.
24. Foundational Model of Anatomy. http://fme.biostr.washington.edu:8089/FME/index.html; FMA-lite. http://obo.sourceforge.net/cgi-bin/detail.cgi?fma_lite.
25. Gene Ontology & GO-slim. http://www.geneontology.org/GO.slims.shtml.
26. OntoViz. http://protege.cim3.net/cgi-bin/wiki.pl?OntoViz.
27. SemTalk, Semtation. http://www.semtalk.com.