Software Modelling Languages: A Wish List

B. Henderson-Sellers
School of Software
University of Technology, Sydney
Broadway, NSW, Australia
brian.henderson-sellers@uts.edu.au

C. Gonzalez-Perez
Institute of Heritage Sciences (Incipit)
Spanish National Research Council (CSIC)
Santiago de Compostela, Spain
cesar.gonzalez-perez@incipit.csic.es

O. Eriksson
Department of Informatics and Media
Uppsala University
Uppsala, Sweden
owen.eriksson@im.uu.se

P.J. Ågerfalk
Department of Informatics and Media
Uppsala University
Uppsala, Sweden
par.agerfalk@im.uu.se

G. Walkerden
Department of Geography and Planning
Macquarie University
Sydney, NSW, Australia

***Abstract*—Contemporary software engineering modelling tends to rely on general-purpose languages, such as the Unified Modeling Language. However, such languages are practice-based and seldom underpinned with a solid theory – be it mathematical, ontological or concomitant with language use. The future of software modelling deserves research to evaluate whether a language base that is compatible with these various elements as well as being philosophically coherent offers practical advantages to software developers.**

***Index Terms*—Foundational ontologies, metamodelling, conceptual modelling, modelling languages, ontology, philosophy**

I. Problems with Contemporary Software Engineering Modelling Languages

In the early days of software engineering, all that mattered was the ability to code, to be highly productive while coding, have good strategies for testing and so on.

In parallel, the information systems community (primarily) developed software development lifecycle methodologies to show how to design as well as code computer-based support systems for business. A prime example here is the early work of Yourdon and Constantine [1]. In parallel was the database modelling work of Edgar Codd [2] and Peter Chen e.g. [3], the latter developing the ER and later the ERA model, which, unfortunately in some ways, was seen only as a means of designing databases, which had begun to become important in business applications in the 1970s and 1980s.

The advent of object technology (OT) in the late 1980s, being seen as a potential business solution initially in the later 1980s and early 1990s, led to a resurgence of software engineering interest in analysis and design, later to be renamed as 'modelling' to encompass the traditions of both analysis and synthesis (a.k.a. design) into a single discipline – a single concept to emphasise the 'seamless' nature of objects. Whilst all object-oriented software engineering approaches of the 1990s and 2000s adopted this idea of a one-size-fits-all, two other strategies were emerging in 'competition': (1) the idea of self-creation of a methodology using Situational Method Engineering (see comprehensive review in [4]) and (2) the emergence of requirements engineering (RE) and conceptual modelling (CM), which to a large degree reintroduce the schism between analysis (now RE) and solution-orientation in design and implementation, including paradigms such as model-based software engineering.

This rupture is exacerbated by the fact that the software development industry has recently shifted to a situation that is arguably closer to that of the early days, by refocussing on coding rather than system design and modelling, as exemplified in agile methods such as [5], which embody a pragmatist spirit, downplaying the role of formal high-level modelling and sometimes even eschewing the need for explicit modelling completely. As a result, formal high-level modelling is often seen as an unnecessary burden that is kept to a minimum or skipped altogether to the benefit of code. Notwithstanding this apparent trend in practice, it has been shown that the quality of software development can be improved by the appropriate use of models [6, 7], which can selectively remove detail and help tackle complex domains in a simpler manner. Good models need good modelling languages (MLs). Current MLs have many flaws – although many are minor and have pragmatic workarounds. Too many workarounds, as noted in [8], can, however, result in fragility. To overcome this negative potential of MLs, we here put together a wish list for future modelling languages. In particular, we address below five identified problems related to contemporary software engineering modelling languages, and in particular how these relate to ontology and philosophy.

Problem 1: Current modelling languages are highly design biased whilst claiming otherwise.

Most of our current 'modelling languages' (MLs) date back to the 1990s and therefore claim to be 'general purpose' – a prime example being the Unified Modeling Language (UML) from the Object Management Group (OMG) [9]. Although claiming to support a wide range of modelling abstraction levels (i.e. analysis and design) in the one package, its history of development clearly indicates that it is highly focussed towards (low-level or detailed) design and even implementation (e.g. Java and C++-style concepts are evident). Some Domain-Specific Modelling Languages (DSMLs) show the same bias, when presented as a 'UML Profile'. Indeed, the very comprehensiveness of UML provides a challenge to modellers since there is no guidance regarding which bits are useful for different modelling contexts (e.g. analysis, implementation planning) nor which elements are included in the standard with only rare usage visualized. Although we focus on UML, we do so as an exemplar only – one that has been highly influential in the history of software engineering modelling languages – both for practical applications of modelling and also as it has become embedded in many university curricula worldwide as the standard modelling language approach.

Problem 2: The current four levels of the OMG/UML hierarchy are not theoretically valid.

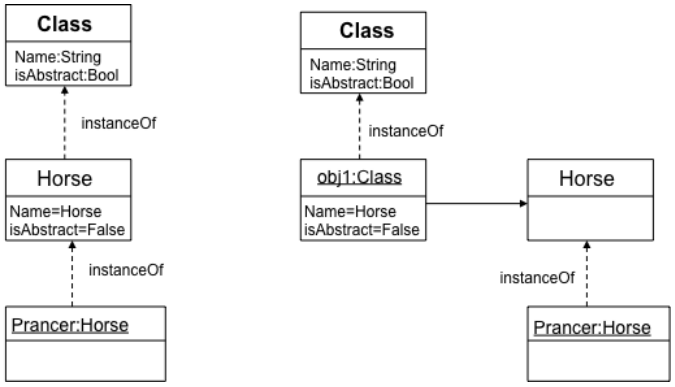

Fig. 1. Typical (but theoretically invalid) double instance chain (left hand side) has to be understood in terms of a mapping between obj1:Class and Horse (right hand side) (after [8])

Modelling languages, as exemplified by UML, are typically based on the existence and relationship between an individual (an object) and its classification (the type of the object). Types model concepts, e.g. [10, 11], and are represented in UML by classes – indeed the type/class dichotomy is generally confounded in OT. Nevertheless, an ML using the type/class dichotomy can be useful but also highly constrained. Classes in a design model must have semantics. This is supplied by a metamodel, also argued to be in a type-instance relationship (metamodel to model elements). And, in the UML in particular, there is even a meta-meta-model. This OMG/UML four level hierarchy, linked by instantiations violates many rules of mathematics and has been widely discussed, e.g. [12-20]. For instance, as seen in Fig. 1, if the language is constrained to set theory (i.e. only sets and members of sets) – rather than permitting sets of sets – then the double instantiation shown in Fig. 1 (left hand side) depicts Horse as an object since it is an instance (of Class). It cannot therefore be further instantiated. Consequently, an implicit mapping is required at each level to associate instances (of types at an upper level) to types (to which instances at a lower level are conformant).

An associated problem occurs when this multi-level architecture is applied to methodologies since methodologies deal not only with work products (for which UML is applicable) but also process-focussed work units. In this context, differentiating between model-focussed entities and real-life entities as occur in a real life project becomes challenging. In other words, entities that seem aligned in real-life map more succinctly to model types in different levels in the OMG architecture e.g. [14]. However, if one goes beyond this architecture by permitting powertypes, both the model level and the enactment level can be fully supported within the one architecture – as seen in Fig. 2 and as employed in ISO/IEC International Standard 24744 [21].

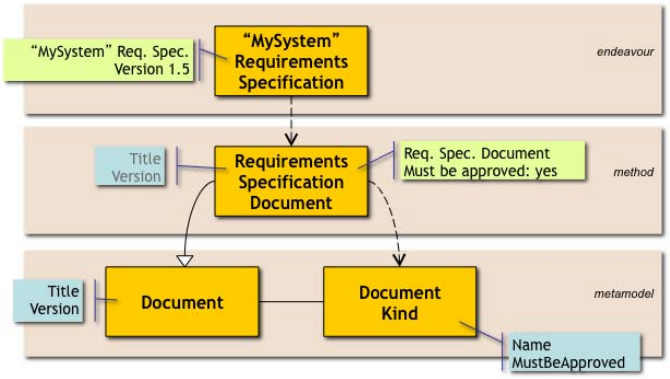

Fig. 2. The use of powertypes permits model language support at both the method and endeavour layers simultaneously (after [22])

Problem 3: Current foundational ontologies and the UML are incompatible.

More recently there have been proposals to extend the UML to include foundational ontological classes [23, 24]. However, it turns out that this ontological work is primarily conceptual (i.e. analysis-focussed) so that merging it into the UML (which is design-focussed) causes another problem – that of incompatibility and focus. UML and OT modelling tends to assume strong and immutable typing as a prelude to MDE-style implementation whereas a foundational ontology like the Unified Foundational Ontology (UFO) [24] is implementation-independent, permitting dynamic typing – as seen for instance in its treatment of role modelling. Figure 3 shows an overview of the proposal for OntoUML – a combination of UFO and UML. These authors [23] argue that an instance of RoleType (a dynamic and anti-rigid type) can be a subtype of an instance of BaseType (a static and rigid type). This breaks the OCL rule in UML for generalization and introduces basic ontological and philosophical questions regarding identity e.g. [25] since role types cannot supply

identity whereas base types can (and must). Indeed, roles have a temporality not seen in base types – this also leading to philosophical questions about the natures of 'identity' and 'temporal change' e.g. [26].

Furthermore, such foundational ontologies, although extended towards social concepts in UFO-C [27], primarily remain materialistically based with the substantial universal as the underpinning paradigm, and thus ignore the full extent of non-material and especially social concepts and relationships [28].

In summary, the problem is not the incompatibility *per se*, but the fact that this is widely ignored in favour of an assumed compatibility, leading to proposals, cited above, to extend the UML by incorporating the UFO.

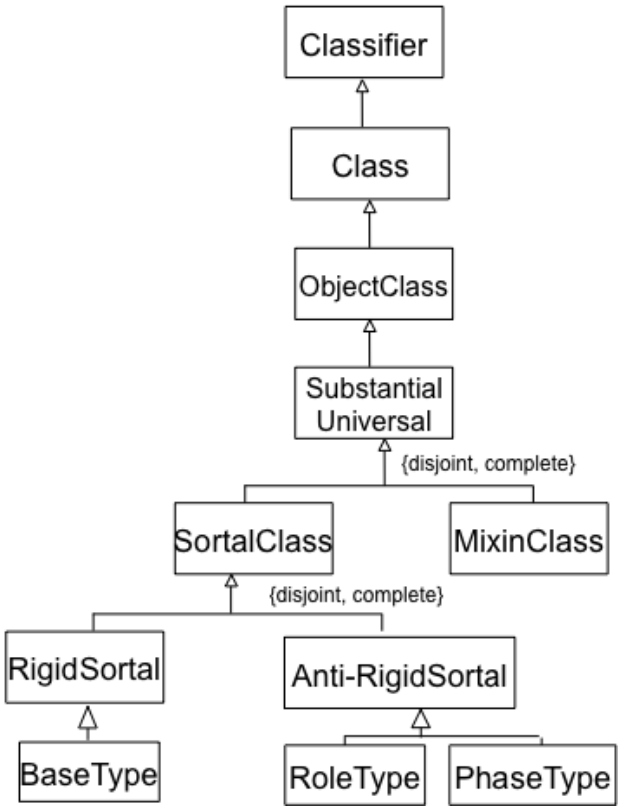

Fig. 3. An overview of the proposal for OntoUML

Problem 4: Language use provides evidence against the current usages of both ontologies and general purpose modelling languages such as UML.

Since a modelling language is a language, recent work seeks advice from the language use community, especially the work of Searle [29-31] and its more recent extensions towards the modelling community [32-34]. Language use emphasises that the materiality focus of foundational ontologies does not permit the modelling of the vast majority of information systems contexts, which are primarily non-material and especially social in nature [28, 35], and often introduce "soft" issues such as subjectivity, temporality and vagueness [36] that are difficult or impossible to model through materialistic ontologies.

A foundational ontology can be used to constrain and guide possible interpretations of the primitives of an ML. Thus, an ontology can be seen as a meta language that allows specifications to represent possible state of affairs in reality. The material focus of most foundational ontologies (e.g. exemplified by the Substantial Universal class in Fig. 3) embraces the idea of an objective reality constituted by substantial things and their properties (brute facts) [30] that are subjectively perceived. These facts can be organized within a model to provide a consistent and universal picture of the domain being modelled. In order to produce such models, the constructs of the modelling language must map 1:1 to the ontological constructs of things, properties and their associations. In contrast, a social focus based on the language use approach acknowledges that reality cannot be described as a set of brute facts alone [37]. Rather, social reality is inter-subjectively constituted through communication (language use) among social actors. The aim of modelling with a language-use approach is to formalize the informal rules used for successful communication and social interaction in a domain into an explicit design 'grammar' – one that is well situated within this specific domain and context. The constituent rules of the grammar are not to be found in the physical world or in a mapping of it onto mental states. Rather, such rules are social and articulate a common institutionalized meaning system that includes distinctive actors and associated behavioral routines. Together, these enable common symbolic interaction in a domain (a common language game). Accordingly, symbolic systems depend upon rules without which they do not exist, and through which language use is established and understood as an institutional activity taking into account societal as well as materialistic entities [29-31, 35, 38].

An ML that suggests strict correspondence between language constructs and brute facts may mislead modellers. For instance, one may be led to believe that a personal identification number, as used in many countries, identifies biological persons (brute facts) rather than citizens (institutional facts). Such a misconception may cause overuse of the afore-mentioned identifiers with significant costs associated with maintenance and evolution of the resultant information infrastructure [32].

Problem 5: The philosophical underpinnings of modelling approaches are neither identified nor, in many cases, coherent.

Philosophical notions are generally either ignored in modelling or are implicit in the language definition and, worse, come from different and incompatible philosophical discourses within the single software engineering approach, e.g. [39]. Indeed, although ontologies and MLs in software engineering subscribe to the idea of a universal language, this may not be a realistic goal *per se* – since the work of Wittgenstein [38] showed that such a language cannot exist because the meaning of language constructs are always confined to the particular social setting in which the language is used. Meaning is not objective but emerges among language

users. As a consequence, it is nonsensical to talk about the meaning of words and constructs in the absence of users of the language. Since meaning is socially constructed, one cannot assume context-free meaning, meaning is dependent on social settings and institutionalised practices – what Wittgenstein referred to as 'language games'. The so-called 'linguistic turn' that followed from Wittgenstein's work has had a tremendous impact on many areas of scholarly enquiry [35] – however, it has had little influence on software engineering. To us, it seems that software engineering research is relatively insensitive to the philosophical assumptions – commitments and contradictions – implicit in its 'grammar and vocabulary' both relying on 'commonsense' philosophical intuitions and tending to disregard tensions and incommensurability amongst its philosophical assumptions.

We argue that, despite the relative indifference of current research and practice, explicating and critically evaluating philosophical intuitions can make substantial contributions to software engineering and, in particular, to the evolution of modelling languages for software engineering. It seems to us likely that carefully considered philosophical foundations for software modelling languages would discipline ontological analysis and modelling in ways that would have many practical benefits in SE [40]. One example is the distinction that ConML [41, 42] makes between 'null' and 'unknown'; the former refers to ontological absence of information whereas the latter refers to presence of information but epistemic absence of knowledge about it. For example, and according to ConML, the value of the Person.Age attribute for an instance of Person can never be null, since every person has an age; but it can be unknown; however, the value of Person.SocialSecurityNumber may be null, i.e. totally non-existent. Making an explicit difference between ontological and epistemic absence of information is something that UML and most modelling languages lack, but helps software modellers create more expressive models that arguably result in more useful implementations.

Note that, given the diversity in philosophical thought – philosophy's constitution more on the model of an extended conversation or evolutionary speciation (rather than convergence on an a single disciplinary stance in the way that is paradigmatic in natural science) – we envisage that the way philosophical thought should be leveraged to support development of SE is to use it to inform explication of implicit philosophical assumptions, and identification of alternative approaches, with a view to improving coherence and clarity in SE. The orienting stance needs to be 'what appears helpful for SE?' rather than 'which philosopher is right, and what do their views imply?' The contribution, in other words, of disciplined philosophical thinking is – from the perspective of theoretical development of SE – metaphoric and hermeneutic.

Having identified five problems with current software engineering modelling languages, we seek an innovative way forward.

II. A Way Forward

We argue here that integrative research is needed to create future software engineering modelling languages to eventually replace the current, flawed four-level metamodelling framework by a better and more theoretically sound one – one that is mathematically valid and that provides an ontological commitment in a proscribed and clearly stated modelling context and is consistent with the use of language and with consistent philosophical underpinnings. Following practical evaluation, we might anticipate that this could then form the basis for standardization (e.g. through ISO (International Organization for Standardization) and OMG: the Object Management Group) of a new and more reliable suite of conceptual modelling languages (both general and domain specific).

With this new theory and innovative multi-level architecture, we would foresee the following benefits:

- the philosophical basis of a modelling language is made explicit and as consistent as possible within the 'rules' of philosophy research and application
- conceptual modelling becomes deeply aligned with theories of language use and speech acts
- the ontological commitment of a modelling language becomes explicit and well understood
- general purpose modelling languages like UML as well as domain-specific modelling languages can become standards with improved semantics
- modelling language support for process and product are integrated in a seamless manner
- two specific currently controversial areas: whole-part relationships (not discussed here – see e.g. [42], [43]) and role modelling e.g. [44] are rationalized within the new framework with this new, well-developed theory
- soft issues such as institutional facts, subjectivity, temporality and vagueness can be captured in models with relative ease
- pragmatically, industry usage of a conceptual modelling language becomes significantly simpler and, more importantly, consistent across all users i.e. the semantics are unambiguous and well understood. This also makes teaching conceptual modelling in both industry and academe significantly easier.
- process models and work product models are consequently of higher quality than can be achieved using contemporary modelling languages, leading to huge potential cost savings for industry software development.
- knowledge of a new, sound philosophy of software development will strengthen graduate attributes for a global workplace.

It can be so argued that, if/when this is achieved, software modelling tasks would become more reliable and productive. Giving specific evidence for this, however, would need additional empirical research.

We have thus highlighted important research areas that, together, ought to provide a solid and theoretically sound basis for future modelling languages, as well as provide useful modelling support in practice. Our wish list!

REFERENCES

[1] E. Yourdon and L. L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and System Design. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[2] E. F. Codd, "A relational model of data for large shared data banks," Communications of the ACM, vol. 13(6), 1970, pp. 377–387.

[3] P. P. Chen, "The entity-relationship model—toward a unified view of data," ACM Trans. Database Systems (TODS), vol. 1(1), 1976, pp. 9-36.

[4] B. Henderson-Sellers, J. Ralyté, P. Ågerfalk, and M. Rossi, M., Situational Method Engineering, Heidelberg: Springer-Verlag, 310pp, 2014.

[5] K. Beck, Extreme Programming Explained, Upper Saddle River: Addison-Wesley, 206pp, 2000.

[6] A. Nugroho, and M.V. Chaudron, "Evaluating the impact of UML modeling on software quality: an industrial case study. In Model Driven Engineering Languages and Systems, A. Schürr and B. Selic, Eds., LNCS 5795, 2009, pp. 181-195.

[7] Arisholm, E., Briand, L.C., Hove, S.E., Labiche, Y., 2006. The impact of UML documentation on software maintenance: An experimental evaluation. IEEE Transactions on Software Engineering 32(6), pp. 365–381.

[8] B. Henderson-Sellers, O. Eriksson, C. Gonzalez-Perez, and P. J. Ågerfalk, "Ptolemaic metamodelling? The need for a paradigm shift," Chapter 4 in Progressions and Innovations in Model-Driven Software Engineering, V. Garcia Diaz, J. M. Cueva Lovelle, B. C. Pelayo García-Bustelo and O. Sanjuán Martínez, Eds., Hershey, PA, USA: IGI Global, 2013, pp. 90-146.

[9] OMG, "OMG Unified Modeling Language™ (OMG ML), Superstructure, Version 2.4.1, formal/2011-08-06, 748pp, 2011

[10] C. Partridge, C. Gonzalez-Perez, and B. Henderson-Sellers, "Are conceptual models concept models?," in Procs. ER2013, W. Ng, V. C. Storey and J. Trujillo, Eds. LNCS 8217, Heidelberg: Springer, 2313, pp. 96-105.

[11] B. Smith, "Beyond concepts: ontology as reality representation," in Proceedings of the Third International Conference (FOIS 2004), Amsterdam: IOS Press, 2004, pp. 73-84.

[12] C. Atkinson, and T. Kühne, "Strict profiles: why and how," in Procs. Third Intl. Conf. on the Unified Modeling Language, LNCS1939, Berlin: Springer-Verlag, 2000, pp. 309-322.

[13] C. Atkinson, and T. Kühne, "The essence of multilevel metamodelling," in M. Gogolla, and C. Kobryn, Eds. «UML»2001 – The Unified Modeling Language. Modeling Languages, Concepts and Tools, LNCS 2185, Berlin: Springer, 2001, pp. 19-33.

[14] C. Atkinson, and T. Kühne, "Processes and products in a multi-level metamodelling architecture," Int. J. Software Eng. and Knowledge Eng., vol. 11(6), 2001, pp. 761-783.

[15] C. Atkinson, and T. Kühne, "Model-driven development: a metamodelling foundation," IEEE Software, vol. 20, 2003, pp. 36–41.

[16] C. Atkinson, and T. Kühne, "Concepts for comparing modeling tool architectures," in Model Driven Engineering Languages and Systems, LNCS 3713, Berlin: Springer, 2005, pp. 398-413.

[17] C. Atkinson, M. Gutheil, and B. Kennel, B., "A flexible infrastructure for multilevel language engineering," IEEE Trans. Software Eng., vol. 35(6), 2009, pp. 742-755.

[18] C. Atkinson, B. Kennel, and B. Goß, "The level-agnostic modeling language," in SLE 2010, B. Malloy, S. Staab, and M. van den Brand, Eds. LNCS 6563, Berlin: Springer, 2011, pp. 269-275.

[19] C. Gonzalez-Perez, and B. Henderson-Sellers, "Modelling software development methodologies: a conceptual foundation," J. Systems Software, vol. 80(11), 2007, pp. 1778-1796.

[20] C. Gonzalez-Perez, and B. Henderson-Sellers, Metamodelling for Software Engineering, Chichester: J. Wiley & Sons, 210pp, 2008.

[21] ISO/IEC, Software engineering - Metamodel for development methodologies, ISO/IEC 24744 edition 2. Geneva: International Organization for Standardization/International Electrotechnical Commission, 2014.

[22] B. Henderson-Sellers, "Method engineering: theory and practice," in Information Systems Technology and its Applications. 5th International Conference ISTA 2006. May 30-31, 2006. Klagenfurt, Austria, D. Karagiannis and H.C. Mayr, Eds., Lecture Notes in Informatics (LNI) – Proceedings, Volume P-84, Gesellschaft für Informatik, Bonn, 2006, pp. 13-23.

[23] G. Guizzardi, G. Wagner, N. Guarino, and M. van Sinderen, "An ontologically well-founded profile for UML conceptual models," in CAiSE 2004, A. Persson and J. Stirna, Eds., LNCS 3084, Berlin: Springer-Verlag, 2004, pp. 112-126.

[24] G. Guizzardi, Ontological Foundations for Structural Conceptual Models, CTIT PhD Thesis Series, No. 05-74, Enschede, The Netherlands, 2005.

[25] B. Henderson-Sellers, O. Eriksson, and P. J. Ågerfalk, "On the need for identity in ontology-based conceptual modelling," H. Köhler and M. Saeki, Eds., CRPIT Vol. 165, 2015, pp. 9-20.

[26] K. Hawley, "Temporal Parts", *The Stanford Encyclopedia of Philosophy* (Winter 2010 Edition), E. N. Zalta, Ed., URL = http://plato.stanford.edu/archives/win2010/entries/temporal-parts/, 2010.

[27] G. Guizzardi, F. Falbo, and R. S. S. Guizzardi, 2008, "Grounding software domain ontologies in the Unified Foundational Ontology (UFO): The case of the ODE software process ontology," in Proceeding of Memorias de la XI Conferencia Iberoamericana de Software Engineering (CIbSE 2008), Recife, Pernambuco, Brasil, February 13-17, 2008.

[28] M. Bergholtz, O. Eriksson, and P. Johannesson, "Towards a sociomaterial ontology," in CAiSE 2013 Workshops, X. Franch and P. Soffer, Eds. LNBIP 148, Berlin: Springer, 2013, pp. 341-348.

[29] J. R. Searle, Speech Acts. An Essay in the Philosophy of Language. Cambridge: Cambridge University Press, 1969.

[30] J. R. Searle, The Construction of Social Reality, New York, NY: The Free Press, 1995.

[31] J. R. Searle, "Social ontology: some basic principles," Anthropological Theory, vol. 6(1), 2006, pp. 12–29.

[32] O. Eriksson, and P. J. Ågerfalk, "Rethinking the meaning of identifiers in information infrastructures," J. Assoc. Information Systems, vol. 11(8), 2010, pp. 433–454.

[33] O. Eriksson, B. Henderson-Sellers, and P. J. Ågerfalk, "Ontological and linguistic metamodelling revisited – a language use approach," Information and Software Technology, vol. 55(12), 2013, pp. 2099-2124.

[34] S. T. March, and G. N. Allen, "Toward a Social Ontology for Conceptual Modelling," Communications of the AIS, vol. 34, Article 70, 2014, pp. 1347–1358.

[35] M. Aakhus, P J. Ågerfalk, K. Lyytinen, and D. Te'eni, "Symbolic Action Research in Information Systems: Introduction to the Special Issue," MIS Quarterly, vol 38(4), 2014, pp. 1187-1200.

[36] C. Gonzalez-Perez, "Modelling Temporality and Subjectivity in ConML", in 7th IEEE International Conference on Research Challenges in Information Science (RCIS 2013), R. Wieringa and S. Nurcan, Eds. IEEE Computer Society. Paris (France), 2013, pp. 1-6.

[37] P. J. Ågerfalk, "Getting Pragmatic," European Journal of Information Systems, vol. 19(3), 2010, pp. 251-256.

[38] L. Wittgenstein, "Philosophical Investigations," 1953/2001, Blackwell Publishing.

[39] B. Henderson-Sellers, C. Gonzalez-Perez, and G. Walkerden, "An application of philosophy in software modelling and future information systems development," in Advanced Information Systems Engineering Workshops, LNBIP 148, Berlin: Springer, 2013, pp. 329–340.

[40] B. Wyssusek, "Ontological foundations of conceptual modeling," Scandinavian Journal of Information Systems, vol. 18(1), 2006, pp. 63-80.

[41] Incipit, 2013. *ConML Technical Specification*, version 1.4. http://www.conml.org/Resources_TechSpec.aspx

[42] C. Gonzalez-Perez, "A Conceptual Modelling Language for the Humanities and Social Sciences", in Sixth International Conference on Research Challenges in Information Science (RCIS 2012), C. Rolland, J. Castro, and O. Pastor, (eds.). IEEE Computer Society. Valencia (Spain), 2012. Pp. 396-401.

[43] M. Saksena, R. France, and M. Larrondo-Petrie, "A characterization of aggregation," in Proc. Fifth Int'l Conf. Object Oriented Information Systems (OOIS'98), 1998, pp. 11-19.

[44] F. Barbier, B. Henderson-Sellers, A. Le Parc-Lacayrelle, and J.-M. Bruel, "Formalization of the whole-part relationship in the Unified Modeling Language," IEEE Trans. Software Eng., 29(5), 2003, pp. 459-470.

[45] G. Guizzardi, "Agent roles, qua-individuals and *The Counting Problem*," in Software Engineering for Multi-Agent Systems IV, A. Garcia et al., Eds., LNCS 3914, 2006, pp. 143-160.