Ontological Analysis of KAOS Using Separation of Reference *

Raimundas Matulevičius[1], Patrick Heymans[1], and Andreas L. Opdahl[2]

[1] Computer Science Department, University of Namur, Belgium
{rma,phe}@info.fundp.ac.be
[2] Department of Information Science and Media Studies, University of Bergen, Norway
Andreas.Opdahl@uib.no

Abstract. Goal modelling is emerging as a central requirements engineering (RE) technique. Unfortunately, current goal-oriented languages are not interoperable with one another or with modelling languages that address other modelling perspectives. This is a problem because the emerging generation of model-driven information systems are likely to depend on coordinated use of several modelling languages to represent different perspectives on the enterprise and its proposed information system. The paper applies a structured approach to describe a well-known goal-oriented language, KAOS, by mapping it onto a philosophically grounded ontology. The structured approach facilitates language interoperability because, when other languages are described using the same approach, they become mapped onto the same ontology. The approach thereby provides an intermediate language for comparison, consistency checking, update reflection, view synchronisation and, eventually, model-to-model translation both between goal-oriented languages and between different languages.

1 Introduction

Goal-oriented modelling has become an important technique during early IS development stages. Current trends towards *model-driven IS development* and *agent-oriented* IS makes it likely that the importance of goal modelling will continue to increase. One central goal-oriented language is Knowledge Acquisition in autOmated Specification (KAOS) [15]. In addition to the conventional *what*, it offers a multi-perspective approach to specifying the *why*, *who* and *when* of enterprises and their IS. Like other modelling languages, KAOS does not support all aspects and phases of IS development equally well. For example, it offers strong support for *representing*, *reasoning* about and *specifying* trust during analysis and specification, but it offers less support for *tracing* and *realising* trust concerns during design and system generation. In addition, an intermediate language is needed to *support integrated management* of enterprise, IS, and problem domain models expressed in different languages.

* This work is partially supported by the Commission of the European Communities under the sixth framework programme (InterOP Network of Excellence, Contract 508011), URL: http://www.interop-noe.org/.

This paper presents a preliminary ontological analysis of KAOS using *separation of reference* [18, 19], a technique that analyses modelling languages by first breaking each construct down into its ontologically primitive parts and then mapping the parts onto a *common ontology* that elaborates the *Bunge-Wand-Weber* (BWW) *representation model* [26, 27] and Bunge's philosophical ontology [4, 5]. The aim is to facilitate comparison, consistency checking, update reflection, view synchronisation and, eventually, model-to-model translation both between goal-oriented languages and across language families. The current work is a part of a larger effort to establish an intermediate language that supports integrated use of enterprise models expressed in different languages. The approach used in this paper is currently being applied to develop version 2 of the Unified Enterprise Modelling Language (UEML).

The paper is structured as follows: Section 2 provides the theoretical research background. Section 3 describes the research method. Results are presented in Section 4 and discussed in Section 5. Finally, in Section 6 we conclude and give directions for future work.

2 Theory

This section discusses KAOS' main features and introduces the UEML approach.

2.1 KAOS

The paper uses the KAOS definition in [15, 25], the latest self-contained description we know of, although it does not take into account recent language extensions [23]. The main purpose of KAOS is to ensure that high-level goals are identified and progressively refined into precise operational statements, which are then assigned to agents of the software-to-be and its environment. Together, the two form the so-called (*composite*) *system-to-be*. In this process, various alternative goal assignments and refinements are considered until the most satisfactory solution can be chosen.

The KAOS approach consists of a *modelling language*, a *method*, and a *software environment*. In this paper, we will consider only the KAOS *modelling language* – called KAOS, from now on. A KAOS model includes a goal model, an object model, an agent model and an operation model. Each of them has a graphical and a textual syntax. Some constructs can be further defined using the KAOS real-time temporal logic[1] in order to facilitate rigorous reasoning. For sake of brevity, we will introduce KAOS through examples from the London Ambulance Service system, adapted from [15] and shown in Fig. 1. Our focus is the *goal model*, but agent, object and operation models cannot be excluded completely, as all KAOS models are interrelated.

A *goal* is a prescriptive assertion that captures an objective which the system-to-be should meet. Goals are either *maintain*, *avoid*, *achieve* and *cease* goals. For example, goal AccurateLocationInfoOnNonStationaryAmbulance follows the *maintain* pattern in which a property *always* holds. AmbulanceAllocationBasedOnInci-

[1] See FormalDef in the textual goal syntax in Fig. 1.

dentForm follows the *achieve* pattern where a property *eventually* holds. A goal is refined through *G-refinement*, which relates a set of subgoals whose conjunction, possibly together with *domain properties*, contributes to the satisfaction of the goal. A goal can have alternative G-refinements (e.g. AccurateStationaryInfo). A set of goals is *conflicting* if these goals cannot be achieved together (e.g. LocationContactedByPhone and InformationSentByEMail). This means that under some *boundary condition* these goals become logically inconsistent in a considered domain.

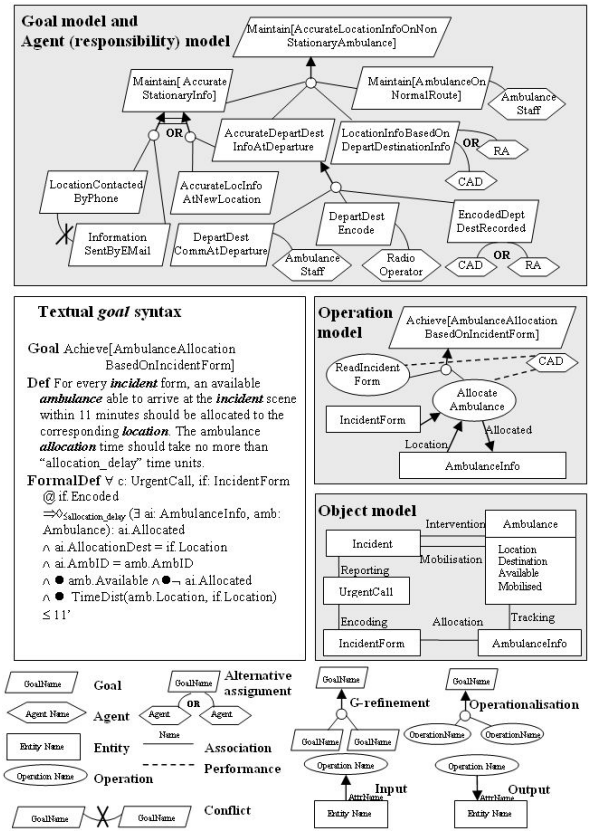

Fig. 1. KAOS model fragment for the London Ambulance Service system (adapted from [15])

An *object* (e.g. Ambulance in the object model in Fig. 1) is a thing of interest in the system. Its instances can be distinctly identified and may evolve from state to state. Objects have *attributes*. Goals *concern* objects and attributes (see Def in textual goal syntax in Fig. 1). An *agent* plays a role towards a goal's satisfaction by *monitoring* or *controlling* object behaviour. Goals are refined until they are *assigned* to individual agents. A goal effectively assigned to a *software agent* (e.g. CAD —

Computer Aided Despatch) is called a *requirement*. A goal effectively assigned to an *environment agent* (e.g. Ambulance Staff) is called an *expectation* (*assumption* in [15]). An *operation* is an *input-output* relation over objects. Operations are characterised textually by *domain* and *required* conditions. Whenever the required conditions hold, performing the operations satisfies the goal. If a goal is *operationalised* and has a responsible agent, the latter *performs* the operations (see operation model in Fig. 1).

2.2 The UEML approach

The UEML approach [18, 19] builds on the BWW model [26, 27] and Bunge's ontology [4, 5] to describe modelling constructs in a way that facilitates precise language integration. The approach adapts BWW-model analysis in two ways: Firstly, whereas the BWW model only maps a construct onto an ontological concept in general, such as class, property, state or event, the UEML approach maps it onto a specific ontological class, property, state or event. Secondly, whereas the BWW model usually maps a construct onto single ontological concept, the UEML approach recognises that a construct may represent a scene where multiple classes, properties, states and events play parts. The UEML approach offers a structured approach to construct description, where the description of each construct is separated into descriptions of:

- *Instantiation level*: Is the construct intended to represent individual things and their particular properties, states and events? Is it intended to represent classes and their characteristic properties, states and events? Can it be used to represent both levels?
- *Classes*: Which thing or class of things in the problem domain is the construct intended to represent? Even when a construct primarily represents a property, state or event, this field remains relevant, because every property, state or event must be a property in, state of, or event in, a specific thing or class. A construct definition can have several class entries, because some constructs are even intended to represent more than one thing or class at the same time.
- *Properties*: Which property or properties in the problem domain is the construct intended to represent? Again, even when a construct primarily represents not a property but, e.g., a state or event, this field is relevant, because every state or event pertains to one or more properties. This entry can also be repeated.
- *Behaviour*: Even when two modelling constructs are intended to represent the same properties of the same things or classes, they may be intended to represent different behaviours. For example, one modelling construct may be intended to represent just their existence, i.e., a static representation. Other modelling constructs may be intended to represent a state of the classes, things or properties, or an event, or a process, i.e., alternative dynamic representations.
- *Modality*: Although we are used to think about enterprise and IS models as stating, or asserting, what is the case, not all modelling constructs are intended for stating assertions. Other types of constructs are intended to represent recommendations, obligations, permission etc. The modality entry is used to identify such constructs.

The various ontological concepts – i.e., the classes, properties, states and events – that are used to describe modelling constructs are maintained in a common ontology. The common ontology was initially derived from the BWW model and Bunge's ontology

and is designed to grow incrementally as additional classes, properties, states and events are introduced in order to describe new modelling constructs. Its classes, properties, states and events are organised in abstraction hierarchies in order to simplify ontology management and facilitate language integration. In consequence, when two modelling constructs – either from the same or from different languages – have both been described using the UEML approach, the exact relationship between them can be identified through their mappings onto the common ontology, paving the way for comparison, consistency checking, update reflection, view synchronisation and, eventually, model-to-model translation.

Initial attempts to use the UEML approach were made by the InterOP partners, who applied it to describe BPMN, coloured Petri-nets, GRL, ISO/DIS 19440, UEML 1.0, selected diagram types of UML 2.0. Language descriptions are currently being negotiated and entered into the Protege-OWL based prototype tool, called *UEMLBase*.

3 Research method

The research method has two main steps: (1) defining KAOS' abstract syntax and (2) applying the UEML approach to selected KAOS constructs from the abstract syntax. Getting a precise view of KAOS' constructs and their interrelations was not an easy task. A part of the metamodel is exposed in [15] through structures and meta-constraints, described in conventional mathematics but intertwined with other topics. In [25], all focus is on the metamodel, but the author uses non-standard constructions to visualise it, omitting some multiplicities, specialisation-related constraints and abstract classes. Furthermore, integrity constraints are only given partially and informally. To facilitate analysis, we represented our understanding of KAOS in a UML 2.0 class diagram shown in Fig. 2[2]. Our metamodel focuses on the goal model, the main object of our analysis, but also includes a few closely related constructs from the other KAOS models.

4 Results

Table 1 outlines how the KAOS constructs have been represented in terms of the common ontology. For each construct, the table indicates (1) the instantiation level (type or instance), (2) the primary class of things or property of this class, (2) the type of behaviour that the construct is intended to represent and (4) modality. Although a construct may be intended to represent more than one class and/or more than one property, we have only indicated the most important ("primary") ones in the table[3].

[2] Our metamodel together with formalised integrity constraints is elaborated in a report available at *http://www.info.fundp.ac.be/_rma/KAOSanalysis/KAOSmetaModel.pdf*

[3] The complete, construct descriptions are available at *http://www.info.fundp.ac.be/_rma/KAOSanalysis/KAOSconstructs.pdf*

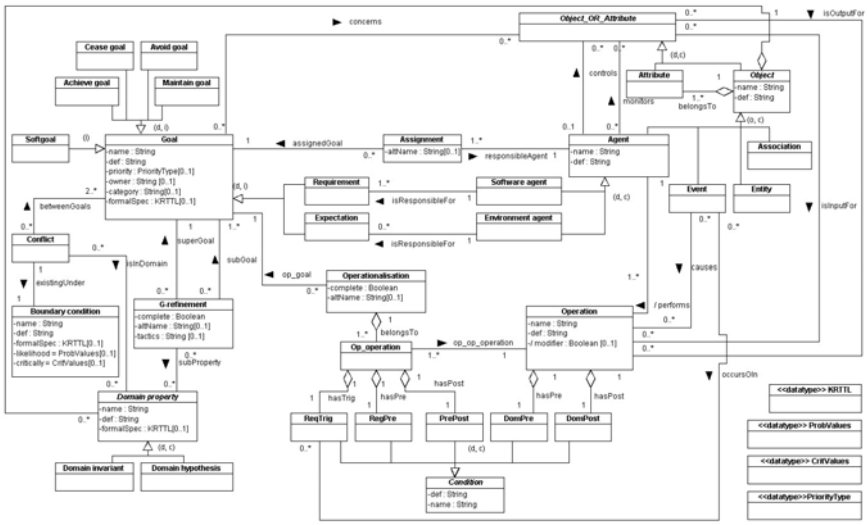

Fig. 2. A metamodel of the KAOS goal model (adapted from [15, 25]).
disjoint, overlapping, complete and incomplete are written d, o, c and i respectively

Table 1. Mapping of KAOS constructs to elements of the common ontology

Construct	Inst	Primary Class/Property	Behaviour	Modality
Achieve goal	Both	Transformation law	Existence	Intent
Agent	Both	Active component thing	Existence	Fact
Assignment	Both	Complex mutual property	Existence	Fact
Avoid goal	Both	State law	Existence	Intent
Boundary condition	Both	State law	State	Fact
Cease goal	Both	Transformation law	Existence	Intent
Conflict	Both	Mutual property	Existence	Fact
Control	Both	Binding mutual property	Existence	Fact
Domain property	Both	Any property	Existence	Fact
Environment agent	Both	Active component thing	Existence	Fact
Event	Type	Changing thing	Event	Fact
Expectation	Both	Complex law property	Existence	Intent
Goal	Both	Complex law property	Existence	Intent
G-refinement	Both	Complex mutual property	Existence	Fact
Input	Type	Binding mutual property	Existence	Fact
Maintain goal	Both	State law	Existence	Intent
Monitor	Both	Binding mutual property	Existence	Fact
Object	Both	Component thing	Existence	Fact
Operation	Type	Transformation law	Process	Fact
Operationalisation	Both	Complex mutual property	Existence	Fact
Output	Type	Binding mutual property	Existence	Fact
Performance	Type	Complex mutual property	Existence	Fact
Requirement	Both	Complex law property	Existence	Intent
Softgoal	Both	Law property	Existence	Intent
Software agent	Both	Component software thing	Existence	Fact

In order to illustrate the approach, we will present the Goal construct, which is central in KAOS, in greater detail. We will discuss the five parts of the UEML approach – instantiation level, classes, properties, behaviour and modality – separately.

Instantiation level: A Goal can be used to restrict either *individual* Objects or Objects that represent *classes* of individuals. The instantiation level of Goal is therefore *both*, i.e., either instance level or type level.

Classes: A KAOS Goal says something about two things (if at the instance level) or two classes of things (if at the type level): There is the *goalOwner*, i.e., the thing or class that holds the intent of reaching the Goal, and there is the *concernedObject*, i.e., the thing or class for which a certain state must be maintained or avoided or a certain event must be achieved or ceased for the goal to be reached. In the common ontology, the *goalOwner* is mapped onto the class of *ActiveThings*, because attempting to reach a goal entails activity. Possibly, *goalOwner* can be mapped onto a more specific class of *ActorThings*, i.e., *ActiveThings* that also hold Goals. The *concernedObject* is mapped onto the class of *ActedOnComponentThings* in the common ontology, a common subclass of *ActedOnThings* and *ComponentThings*. They are *ActedOnThings* because attempting to reach a goal entails acting on the object. They are *Component-Things* because a *concernedObject* in KAOS is always a component of the system.

Both the *goalOwner* and the *concernedObject* can be composite, e.g., to account for complex Goals held by a group of people, or for goals about a collection of things/classes. However, we do not map them, even more specifically, onto the classes of *CompositeActiveThings* and *CompositeActedOnThings*, respectively, because they do not have to represent composite things.

Properties: A KAOS Goal represents a particular group of properties of *goalOwners* and *concernedObjects*. The "primary" property represented by a KAOS Goal is a *complex law* property of the *goalOwner*. We call this property simply *theGoal*. It is a *law* because it restricts the states or transformations that the *concernedObject* can undergo. It is *complex* because it enforces this restriction by restricting the various properties (or attributes) that a *concernedObject* may possess.

Fig. 3 illustrates this situation informally[4]. The *goalOwner* class/thing possesses a single complex *theGoal* law property. The *concernedObject* class/thing possesses one or more *objAttribute* non-law properties. These properties are also sub-properties of *theGoal* : They are the properties that *theGoal* restricts, thereby also restricting the states and transformations that *concernedObject* can undergo. In the common ontology, *theGoal* is mapped onto *ComplexLawProperty*.

Fig. 3 thus captures the gist of our interpretation of Goal, making it possible to discuss its semantics on a very detailed level, which might go into further depth:

- A KAOS Goal has further characteristics, namely name, def, formalSpec, priority and category, which are subproperties of the *theGoal* property too.
- In KAOS there is a difference between *objAttributes* that are explicitly mentioned in the Goal's def and those that are left implicit that are not known by the analyst.

[4] Fig. 3 simplifies an UML object diagram that instantiates the metameta model, presented in [18]. Similar object diagrams are shown in [19], but they are too detailed for this brief outline.

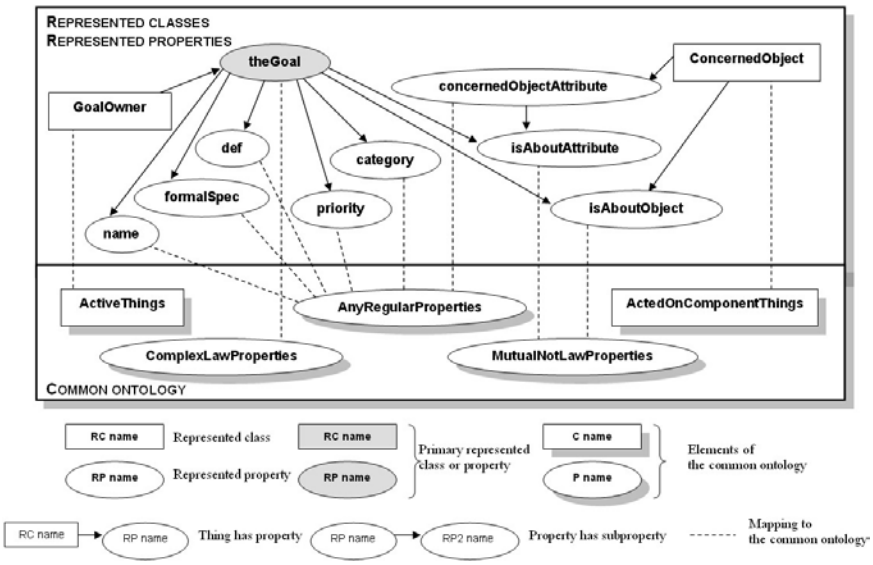

Fig. 3. The classes and properties represented by KAOS goal

A KAOS Goal has even further properties, such as the ability to be assigned, refined, operationalised and conflicting, which are left out here because they are considered in descriptions of other constructs (like Assignement, G-refinement, Operationalisation and Conflict).

Behaviour: The *behaviour* entry for Goal is simple. A Goal just states the *existence* of property *theGoal*, as opposed to constructs that represent particular states of, or events in, one or more properties.

Modality: The *modality* entry makes it clear that a Goal does not just assert a fact, but denotes the desire or *wish* of the *goalOwner* to reach *theGoal* with respect to the *concernedObject*.

The Goal subtypes MaintainGoal, AchieveGoal, AvoidGoal, and CeaseGoal, just refine the ontological grounding of their supertype by indicating which kind of law *theGoal* is. In Table 1 MaintainGoal and AvoidGoal are *state laws*, indicating the allowable (resp. forbidden) states *concernedObjects* can be in. AchieveGoal and CeaseGoal indicate that a *change* is required between a state where the *concernedObjects'* properties are false (resp. true) and one where they are true (resp. false). In BWW, this amounts to a *transformation law*.

5 Discussion

The section discusses KAOS and the UEML approach with respect to our analysis. It also compares our approach with related work of language evaluation and integration.

5.1 KAOS

The first problem we encountered was the lack of a standard, consistent description of the KAOS syntax. Hence, our proposal for abstract syntax – a standard notation like UML class diagrams complemented with OCL constraints –could serve as a basis for a complete metamodel of KAOS. Regarding concrete syntax, we could observe variations between the sources, too. Once a metamodel is in place, concrete visualisations and variations can be established more clearly. Our major concern is semantics. KAOS goal model is now equipped with an ontological semantics defined through the UEML approach. The approach could be used to identify discrepancies in language definitions. However, this paper concentrates on making the semantics explicit for later evaluation, comparison and integration. Still, some observations can be made. We note the intensive use of BWW-model law, mutual and complex property to ground KAOS constructs ontologically (see Table 1). Indeed, we could observe intensive use of complex properties in previous analyses [7, 18] too. The use of laws might be influenced by the goal-oriented context, because all the four types of KAOS goal map to law properties, elegantly matching the BWW model's distinction between state and transformation law. This demonstrates a convergence of KAOS' ontological semantics with its existing formal (modal logic-based) semantics. A detailed evaluation of KAOS will be envisaged when the other KAOS models will be analysed.

5.2 Goal-oriented approaches

Besides KAOS the most notable goal-oriented approaches include i* [28], GRL [10], Tropos [3], GBRAM [1], NFR [6] and Lightswitch [21]. Kavakli and Loucopoulos examined 15 and classified them along four dimensions: "*usage*" (what RE activity does goal modelling contribute to?), "*subject*" (what is the nature of goals?), "*representation*" (how are goals expressed?) and "*development*" (how are goal models developed and used?) [12]. The survey shows that, taken separately, each approach tends to focus only on one "usage". For example, i* focuses on understanding the current organisational situation, while KAOS and NFR concentrate on relating business goals to system components. At the "subject" level, approaches use different definitions and categories of goals. At the "representation" level, approaches come with their own language syntax, semantics and degree of formality. The survey concludes that (1) the research area is fragmented; but (2) "contributions from different frameworks seem to complement each other". The authors argue for more integration efforts to "obtain a stronger framework that takes advantage of the many streams of goal-oriented research" [12]. In further work [11], "usage"-level integration is proposed through a unifying goal-oriented method meta-model. Processes of existing approaches were modelled as a move towards integrated goal-oriented processes. However, a main obstacle to such integration is fragmentation at the "subject" and "representation" levels. In our work we address "*representation*" and "*subject*" levels by providing the base for KAOS semantic integration with other languages.

Recently, a comparison of various notions of goal in RE was performed by Regev and Wegmann in [22]. The authors examine the definitions of goal and related con-

cepts found in KAOS, GBRAM and GRL. They propose to redefine the notions based on the "regulation" concept borrowed from system theory. The work results with the definitions of, and the interrelations between the key concepts made more precise and theoretically grounded. But the authors suggest formalisation of definitions using the BWW model. By mapping KAOS and other goal-oriented languages to a common ontology based on BWW, we open the way for an accurate, systematic and tool-supported comparison of notations at the syntactic and semantic levels.

IS and enterprise modelling require considering goals but also many other perspectives like data, process, and architecture. Integration is thus not only necessary between goal techniques but also across different modelling perspectives. Groundbreaking work reported in [2] combined goals with scenarios to improve business process re-engineering. Others have investigated the interplay between goals and other modelling techniques (e.g. [8, 16, 17]). Three results directly concern KAOS. In [24], a procedure is devised to infer KAOS goals from scenarios. In [14], formal behavioural software specifications are built by stepwise refinement of KAOS models. Finally, [9] suggests to embed KAOS within the UML by proposing a KAOS UML profile. However, cross-language relationships are either informal [2, 16] or ad-hoc [8, 14, 17, 24]. They are not based on an explicit mapping to an intermediate semantic representation. Defining KAOS as a UML profile [9] can make the transformation definition more standard by relying on general-purpose mechanisms developed around the UML. However, the UML definition is the source of many problematic referential issues [13, 20], so semantic consistency of transformations is not guaranteed.

5.3 Evaluation of the UEML approach

The KAOS analysis indicates that the UEML approach is difficult to use because it is based on a particular way of thinking. It is hard to (1) determine exactly which language part constitutes a modelling construct; (2) find the appropriate classes, properties, states and events in the common ontology to use when describing a construct; (3) judge when to choose an existing class, property, state or event in the ontology and when to define a new one. In consequence, the UEML approach, with its ambition to facilitate inter-subjective construct descriptions, can produce subjective results. Problems of this kind are not surprising given that the UEML approach is ambitious and still new. We are currently looking for ways to countering them, including extended tool support, better documentation and improving/simplifying the approach itself.

The KAOS analysis also highlights several advantages the UEML approach. It offers (1) a detailed advice on how to proceed when analysing individual language constructs; (2) construct description at a high level of detail, which tends to integrate languages at a fine-grained level which leads to complete and easily comparable construct descriptions; (3) ontological analysis in terms of particular classes, properties, states and events, and not just in terms of the concepts in general; (4) a path towards tool supported, integrated use of models expressed in different languages, through the structured format in combination with the common ontology. Finally, it has positive externality, in the sense that each construct becomes easier to incorporate as more constructs are already added to UEML and each language becomes easier to incorpo-

rate as more languages are already added. The UEML approach offers a systematic way of revealing and managing the ontological assumptions inherent in a language.

With respect to the semiotic quality framework [13], the UEML approach suggests an analytically sound way of reaching the goals of syntactic and semantic quality. But it leaves aside other quality types, such as pragmatic, empirical, physical, perceived semantic and social quality.

6 Conclusions

We have presented our use of the UEML approach to provide an preliminary ontological grounding of the core concepts of the KAOS language. Mapping KAOS and other goal-oriented languages to a common well accepted ontological model such as the BWW model is a way to reduce the observed fragmentation in this research area. Once discussed, compared and validated by the community, these mappings can serve as a basis either (1) to create a unique, integrated, expressively complete5 goal-modelling language, or (2) to devise semantic-preserving model transformations, consistency rules or view synchronization mechanisms to cross language families and hence exchange goal models between process fragments. Model interoperability across modelling perspectives can be achieved in the same way and will be explored in the InterOP project on the basis of our results.

Currently a prototype tool, UEMLBase, is under development. It helps automating the UEML approach for defining and for finding common integration points based on the mappings. The tool will also help future negotiation of construct definitions made by different (groups of) researchers in order to reach consensus and include these constructs to the next UEML version.

References

1. I. Anton. Goal-based Requirements Analysis. In Proceedings of the 2nd Int. Conference on Requirements Engineering (ICRE '96), IEEE Computer Society, 1996.
2. A. I. Anton, W. M. McCracken, and C. Potts. Goal Decomposition and Scenario Analysis in Business Process Reengineering. In Proc. of the 6th Int. conference on Advanced Information Systems Engineering (CAiSE'94), Springer-Verlag, 1994, pp 94-104.
3. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An Agent-oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems, 8(3), 2004, pp. 203-236.
4. M. Bunge. Ontology I: The Furniture of the World. In Treatise on Basic Philosophy, volume 3. Reidel, Boston, 1977.
5. M. Bunge. Ontology II: A World of Systems. In Treatise on Basic Philosophy, volume 4. Reidel, Boston, 1979.
6. K. L. Chung, B. Nixon, J. Mylopoulos, and E. Yu. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Boston, 2000.
7. G. Dallons, P. Heymans, and I. Pollet. A Template-based Analysis of GRL. In Proceedings of the 10th Int. Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'05), Porto, June 2005. FEUP Edicoes, pp. 493-504.

8. E. Dubois, E. Yu, and M. Petit. From Early to Late Requirements: a Process-control Case Study. In Proc. of the 9th Int. Workshop on Software Specification and Design (IWSSD'98), Isobe Japan, April 1998.
9. W. Heaven and A. Finkelstein. UML Profile to Support Requirements Engineering with KAOS. IEE Proceedings - Software, 151(1), February 2004, pp. 10–27.
10. ITU. Recommendation Z.151 (GRL) - Version 3.0, September 2003.
11. E. Kavakli. Goal-oriented Requirements Engineering: a Unifying Framework. Requirements Engineering Journal, 6(4), 2002, pp. 237–251.
12. E. Kavakli and P. Loucopoulos. Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods. In J. Krogstie, T. Halpin, and K. Siau, (eds.), Information Modeling Methods and Methodologies, IDEA, 2005, pp. 102–124.
13. J. Krogstie. Evaluating UML Using a Generic Quality Framework. In UML and the Unified Process, IDEA, Hershey, PA, USA, 2003, pp. 1–22.
14. R. D. Landtsheer, E. Letier, and A. van Lamsweerde. Deriving Tabular Event-based Specifications from Goal-oriented Requirements Models. In Proc. of the 11th IEEE Int. Conference on Requirements Engineering (RE'03), 2003. IEEE Computer Society, pp. 200–210.
15. E. Letier. Reasoning about Agents in Goal-Oriented Requirements Engineering. PhD thesis, Universit´e Catholique de Louvain, 2001.
16. L. Liu and E. Yu. Designing Information Systems in Social Context: a Goal and Scenario Modelling Approach. Journal of Information Systems, 29(2), 2004, pp. 187–203.
17. J. Mylopoulos, M. Kolp, and J. Castro. UML for Agent-oriented Development: the Tropos Proposal. In Proc. of the 4th Int. Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools, 2001, pp. 422-441.
18. A. L. Opdahl and B. Henderson-Sellers. A Template for Defining Enterprise Modeling Constructs. Journal of Database Management (JDM), 15(2), 2004, pp. 39–73.
19. A. L. Opdahl and B. Henderson-Sellers. Template-Based Definition of Information Systems and Enterprise Modelling Constructs. In P. Green and M. Rosemann, (eds.), Business Systems Analysis with Ontologies. IDEA, 2005.
20. A. L. Opdahl and B. Henderson-Sellers. A Unified Modeling Language without Referential Redundancy. Data and Knowledge Engineering (DKE). Special Issue on "Quality in Conceptual Modelling", 2005.
21. G. Regev. A Systemic Paradigm for Early IT System Requirements Based on Regulation Principles: The Lightswitch Approach. PhD thesis, Swiss Federal Institute of Technology (EPFL), August 2003.
22. G. Regev and A. Wegmann. Where do Goals Come From: the Underlying Principles of Goal-oriented Requirements Engineering. In Proc. of the 13th IEEE Int. Conference on Requirements Engineering (RE05), Paris, France, August 2005.
23. A. van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-models. In Proc. of the 26th Int. Conference on Software Engineering (ICSE'04), IEEE Computer Society, 2004, pp. 148–157.
24. A. van Lamsweerde and L. Willemet. Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Trans. on Soft. Eng., 24(12), 1998, pp. 1089–1114.
25. A. van Lamsweerde. The KAOS Meta-model: Ten Years After. Technical report, Universite Catholique de Louvain, 1993.
26. Y. Wand and R. Weber. On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. Journal of Information Systems, 3, 1993, pp. 217–237.
27. Y. Wand and R. Weber. On the Deep Structure of Information Systems. Journal of Information Systems, 5, 1995, pp. 203–223.
28. E. Yu. Towards Modeling and Reasoning Support for Early-phase Requirements Engineering. In Proc. of the 3rd IEEE Int. Symposium on Requirements Engineering (RE'97), IEEE Computer Society, 1997.