Reasoning about Distributed Knowledge-Transforming Peer Interactions

Marco Schorlemmer and David Robertson

Abstract—We address the problem of how to reason about properties of knowledge transformations as they occur in distributed and decentralized interactions between large and complex artifacts, such as databases, web services, and ontologies. Based on the conceptual distinction between specifications of interactions and properties of knowledge transformations that follow from these interactions, we explore a novel mixture of process calculus and property inference by connecting interaction models with knowledge transformation rules. We aim at being generic in our exploration, hence our emphasis on abstract knowledge transformations, although we exemplify it using a lightweight specification language for interaction modeling (for which an executable peer-to-peer environment already exists) and provide a formal semantics for knowledge transformation rules using the theory of institutions. Consequently, our exploration is also an example of the gain obtained by linking current state-of-the-art distributed knowledge engineering based on web services and peer-based architectures with formal methods drawn from a long tradition in algebraic specification.

Index Terms—Distributed knowledge systems, interaction models, knowledge transformation, formal specification.

1 Introduction

In recent years there has been an increased interest in exploring the multiagent and peer-to-peer paradigms to support knowledge engineering, coordination, and management activities [40], [41]. The reason for this has been that these paradigms seem to better fit the distributed, decentralized, and intersubjective nature of knowledge management within an organization. Traditional knowledge management techniques and architectures, on the contrary, were still embodying a classical, objectivist approach to knowledge management, according to which knowledge is an objectifiable and thus codifiable commodity [5], [8].

Bonifacio et al., for example, proposed a distributed knowledge-management architecture that takes the locality, subjectivity, and thus context-dependency of knowledge into account [3], [4]. Their purpose was to favor the semantic autonomy of organizational units for managing its local knowledge and to enable knowledge exchange via semantic negotiation and coordination [2]. The knowledge management practice considered was document management and retrieval, albeit retaining local control of terminology and of classification of locally generated documents. Multiagent technology was applied by associating a software agent to each knowledge-managing organizational unit, and by endowing each agent with the necessary semantic matching capability in order to relate foreign with local terminology.

Today's schools of knowledge management stress—in addition to the subjective and context-dependent aspect of knowledge—also the view that knowledge is not an objectifiable commodity but that it resides in individuals. Consequently, for information technology to contribute to successful knowledge management it has to support the information flow in interpersonal communication and interaction, and favor the coordination of capabilities [8], [10]. According to this view, interaction is a knowledge-transforming activity, and the value resides on how to combine capabilities by coordinating interaction in a distributed manner [8].

- M. Schorlemmer is with the Artificial Intelligence Research Institute, IIIA-CSIC, Campus UAB, E-08193 Bellaterra (Barcelona), Catalonia, Spain. E-mail: marco@iiia.csic.es.
- D. Robertson is with the School of Informatics, The University of Edinburgh, Informatics Forum, Crichton Street, Edinburgh EH8 9AB, Scotland, United Kingdom. E-mail: dr@inf.ed.ac.uk.

Manuscript received 19 Oct. 2007; revised 14 Aug. 2009; accepted 10 Dec. 2009; published online 23 Dec. 2010.
Recommended for acceptance by B.C. Ooi.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2007-10-0521.
Digital Object Identifier no. 10.1109/TKDE.2010.265.

1.1 Distributedly Interacting Peers: Protocols and Artifact

There is currently much interest in systems for which knowledge-transforming interactions concern large and complex artifacts, such as databases, ontologies, or web pages, and where multiple distributed agent-mediated services in a peer-based system must interact in complex ways to create or maintain these. Interaction is usually conducted through message passing, but since we want to reduce the amount of bandwidth used when managing large and complex artifacts, one makes a distinction between the *protocol* used to control the interaction and the *artifacts* constructed as a consequence of following the protocol.

Consider, for example, the knowledge-engineering process of Ecolingua—an ontology for the description of ecological data, and which is explained in detail in [9]. Here the engineers of a complex artifact (an ontology) chose not to build it from scratch and thus interact with services available on the Internet (such as the the Ontolingua Server [13]) in order to

- reuse and collate a number of publicly available ontologies of related fields into a single (initial) ontology;
- then translate the resulting combined ontology (originally in KIF) to a version adapted to Prolog, the representation language of choice of the engineers;

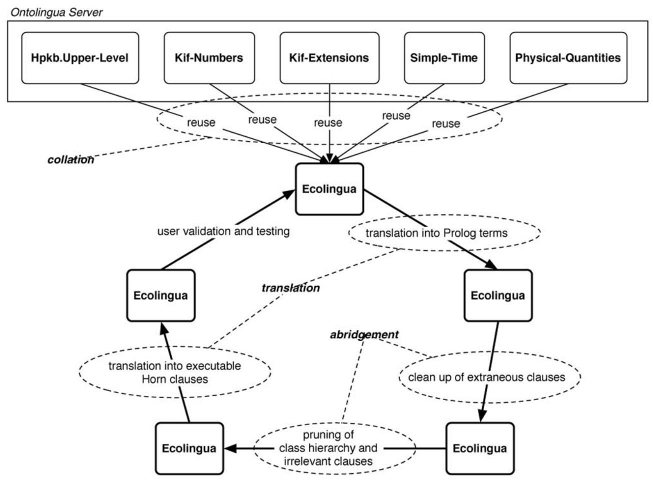


Fig. 1. Ecolingua's knowledge-transformation lifecycle.

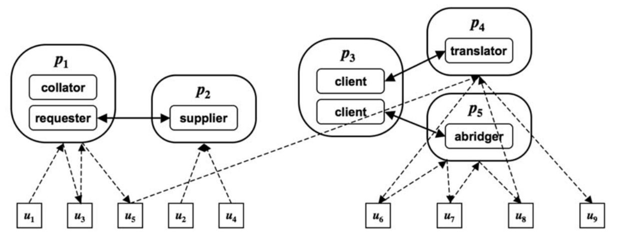
- then abridge the resulting ontology (by deleting over-general facts, definitions of self-subclasses, duplicated classes, etc., and then pruning the class hierarchy to remove irrelevant clauses) until it was small enough to be readable by humans (the resulting ontology after translation was, at 5.3 Mb, too large for human reading);
- to be translated (again) into executable Prolog code (because the first translation service only rendered the ontological axioms originally specified in KIF as Prolog terms).

The knowledge-transformation process is schematically shown in Fig. 1. This example is one of the earliest documented cases of this sort of large scale ontological engineering but it remains paradigmatic of the problem:

- **Large artifacts.** The ontologies are large enough, and the stages in processing them persistent enough, that we would prefer not to pass them as parameters to (very large) messages. Instead, each service responsible for processing the ontology at each stage yields, for instance, a new resource on the web containing the appropriate ontology in its corresponding language.
- **Complex interaction.** The interaction is reasonably complex if (as we later show) we generalize the patterns involved, and the link between message passing and the knowledge transformations involved is not simple to unravel.
- **No global view.** The view of the ontology from any individual component of the process is incomplete. Each part of the process involves only a limited segment of the entire knowledge transformation.

The distinction between interaction protocols and artifacts becomes more stark when we consider the assembly of agent-mediated services into a peer-based architecture that would naturally implement a distributed solution to this problem in order to keep the expertise at local level. Five different peers[1] are required (assuming we use different

1. We prefer to use the term *peer* instead of *agent* in this paper, since we do not focus on the autonomy and intelligent capabilities usually ascribed to agents in multiagent systems, but consider only their reactive behavior to message passing as specified in interaction protocols.



Peers p_1 to p_5 mediate between distributed services. The roles undertaken by peers are depicted as lozenges inside the appropriate peer (e.g., p_1 has the roles *collator* and *requester*, while peer p_3 has two different *client* roles). Solid arcs show interactions between roles as a consequence of message passing. Dashed arcs show which URIs are accessed or created by each peer as a consequence of these interactions.

Fig. 2. Knowledge-transforming peer interaction for Ecolingua's lifecycle.

peers for each of the main tasks): peer p_1 for collating the ontologies; peer p_2 for supplying component ontologies from a library; peer p_3 for managing the translation and abridgement processes; peer p_4 for performing translation; and peer p_5 for performing abridgement. The overall assembly is given in Fig. 2 (illustrating for simplicity the reuse of only two ontologies).

The overall interaction can be formally modeled by protocols, which we shall call *interaction models*. They specify the message passing behavior for each of the roles that peers can play in the interaction. For our knowledge-transforming interaction of Fig. 2, the web resources at u_1 to u_5 are constructed as a consequence of the portion of the interaction for collating ontologies, carried out by peers in the *collator*, *requester*, and *supplier* roles. The portion of the interaction for translation is enacted twice by roles *client* and *translator* to construct web resources at u_6 and u_9. Finally, the portion of the interaction for abridgement is also enacted twice by roles *client* and *abridger* to construct web resources at URIs u_7 and u_8. Later, in Section 2, we will specify in detail each role for each of the portions of our knowledge-transforming interaction.

1.2 Reasoning about Knowledge Transformations

All of the web resources at the nine URIs, above, are closely connected by several knowledge transformations, but the connection cannot be reconstructed in depth simply by looking at the information formally given so far (which already is more than that conventionally available). For instance, although we can know the sequence of creation of the web resources and the identity of the peers creating them, we cannot formally infer that the ontology located at u_5 contains all the information in the preceding ontologies, and hence do not know the provenance of the bodies of knowledge represented in these web resources.

There is, therefore, a need for formality in order to be able to provide automated reasoning support about knowledge transformations. The aim of formality in this area is twofold: to give a concise account of what is going on, and to use this account for practical purposes in handling and maintaining histories of knowledge transformations. Peers with the ability to understand these histories would be able to know the provenance of a body of knowledge and act accordingly—for instance, by deciding their actions depending on their degree of trust in the original source of a body of knowledge (e.g., based on the completeness or correctness of the knowledge representation) and of particular properties of the specific knowledge transformations performed on it (such as preservation of completeness, subsumption of original sources of knowledge, or retention of equivalent levels of expressive power). Different sorts of knowledge transformations preserve different properties of the representations to which they are applied. Being able to infer such property preservation from the history of transformations that a body of knowledge has gone through may be useful for peers, as it can help them decide which reasoning services to use in order to perform deductions without requiring the inspection of the knowledge represented within the web resources themselves.

Knowing whether these kinds of properties are preserved across knowledge transformations would be useful, especially in distributed and decentralized environments where bodies of knowledge are most likely to be translated between different representation languages, mapped into different ontologies, and further specialized or generalized in order to be reused together, in association with other problem solvers or in other domains. Thus, having a formal framework with a mathematically sound and explicitly represented semantics in which we could record transformations of bodies of knowledge and their effect on certain key properties would allow for an enhanced automation of services that make use of the additional information contained in knowledge-transformation histories.

1.3 Our Approach

In this paper, we integrate well-known techniques resulting from a long tradition in algebraic specification with current distributed knowledge coordination techniques in new ways, to support reasoning about knowledge transformation conducted in a peer-based distributed environment. In particular, we connect interaction protocols and knowledge artifacts using a novel mixture of process calculus and property inference within a general framework that permits deployment in a peer-based architecture. For this we require, on one hand, a formal interaction modeling language for the declarative specification of first-class protocols [25], i.e., referencable, sharable, manipulable protocols that can be inspected at runtime to distributively determine the state of an interaction, and hence be capable to infer knowledge artifact properties holding in this state. On the other hand, we need a formal and generic description of knowledge transformations at a level of abstraction that do not describe the details of particular techniques for transforming knowledge components, but that are concrete enough as to say interesting things about them.

In Section 2, we describe how to use a particular interaction modeling language to specify and coordinate the distributed management of knowledge components and their transformations, although other modeling languages such as those reviewed in [25] could have been chosen. Our aim, however, is to use the most easily applied formal language for this engineering task that we could conceive and for which an executable peer-to-peer environment already exists, choosing thus the *Lightweight Coordination Calculus* (LCC) [32]. LCC is also the executable interaction modeling language underlying the distributed and decentralized peer-based knowledge-coordination system [30] developed in the scope of the European *OpenKnowledge* project [33].

In Section 3, we argue for a set of abstract knowledge transformation rules that can be connected, in a generic way via property rules, to interaction models. We do that by looking first at an actual knowledge engineering example carried out at Boeing, which will also help us delimiting the scope of our abstraction. In Section 4, we show how these can be connected to interaction models, allowing properties to be inferred. We do not attempt, thus, to give a catalog of knowledge transformations. In practice, the choice of level of abstraction is an engineering decision—we could have many different sets of knowledge transformation rules depending on the degree of specificity required by an application domain.

Knowledge transformation rules are introduced in an intuitive manner, but to infer properties from the history of a distributed knowledge-management coordination process in an automated fashion we will need a precise semantics for knowledge components and their transformation. We give one such semantics for abstract knowledge transformation in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265. Section 5 concludes the paper.

2 The LCC Language and Its Use in Coordination

LCC is an executable specification language that is used to constrain interactions between distributed components. It is neutral to the infrastructure used for message passing between components, although for the purposes of this paper we assume components are peers in some form of peer-to-peer network.

LCC is a generic modeling language that can be used to describe specific communication styles, such as those compliant with FIPA. This is done by constraining an interaction model to use specific constructs as message content and ensuring that the message passing sequences conform to the required semantics (the Agentlink Argumentation Interchange Format [42] is a good example of the versatility of LCC in this respect). LCC is not, of itself, a requirements analysis or design methodology (such as one might find in Tropos [6] or Gaia [43]). It is complementary to these in the sense that if one develops a requirements/design methodology in which one of the products is a model of the multiagent interaction then LCC could be used as an executable specification of that interaction. Compared to most business process enactment or workflow languages (e.g., BPEL or YAWL) or agent specification languages (e.g., AORML), LCC is a compact language. Its purpose is to offer the minimum number of concepts from which other, task-specific concepts may be built. This contrasts with other “richer” but less compact languages that build in more concepts from the target domains or user groups. To demonstrate the flexibility of LCC as a means of enacting other languages we have built translators from well-known workflow languages (e.g., BPEL4WS) and service orchestration languages (e.g., the Scufl language used by the Taverna system [29]) to LCC.

$$\begin{array}{rcl}
Interaction\_Model & ::= & \{Clause, \ldots\} \\
Clause & ::= & Peer :: Dn \\
Peer & ::= & a(Role, Id) \\
Dn & ::= & Peer \mid Message \mid Dn\ then\ Dn \mid Dn\ or\ Dn \mid \\
 & & Dn\ par\ Dn \mid null \leftarrow C \\
Message & ::= & M \Rightarrow Peer \mid M \Rightarrow Peer \leftarrow C \mid \\
 & & M \Leftarrow Peer \mid C \leftarrow M \Leftarrow Peer \\
C & ::= & Term \mid C \wedge C \mid C \vee C \\
Role & ::= & Term \\
M & ::= & Term
\end{array}$$

Where $null$ denotes an event which does not involve message passing; $Term$ is a structured term (e.g., a Prolog term) and Id is either a variable or a unique identifier for a peer.

Fig. 3. Syntax of LCC interaction models.

$$\begin{array}{l}
a(collator(L, U, U_f), X) :: \\
\quad \left(\begin{array}{l} a(requester(L, N, U, U_m), X)\ then \\ a(collator(L, U_m, U_f), X) \end{array} \right)\ or \\
\quad null \leftarrow completed(U) \wedge U_f = U \\
a(requester(L, N, U, U_m), X) :: \\
\quad request\_onto\_source(L, N) \Rightarrow a(supplier, S)\ then \\
\quad merge(L, U, U_n, U_m) \leftarrow \\
\qquad\qquad\qquad ontology\_source(L, N, U_n) \Leftarrow a(supplier, S) \\
a(supplier, S) :: \\
\quad request\_onto\_source(L, N) \Leftarrow a(requester(L, N, U, U_m), X)\ then \\
\quad ontology\_source(L, N, U) \Rightarrow a(requester(L, N, U, U_m), X)\ then \\
\quad a(supplier, S)
\end{array}$$

Where $merge(L, U, U_n, U_m)$ is satisfied when the peer can take the union of resources at URIs U and U_n to produce a combined resource at URI U_m in language L.

(a)

$$\begin{array}{l}
a(client(F, U_1, L_1, U_2, L_2), X) :: \\
\quad reqst\_transl(F, U_1, L_1, L_2) \Rightarrow a(translator, T)\ then \\
\quad translation(U_2) \Leftarrow a(translator, T) \\
a(translator, T) :: \\
\quad reqst\_transl(F, U_1, L_1, L_2) \Leftarrow a(client(F, U_1, L_1, U_2, L_2), X)\ then \\
\quad translation(U_2) \Rightarrow a(client(U_1, L_1, U_2, L_2), X) \leftarrow \\
\qquad\qquad\qquad translate(F, U_1, L_1, U_2, L_2)
\end{array}$$

Where $translate(F, U_1, L_1, U_2, L_2)$ is satisfied when a translation named F is applied to the resource at URI U_1 (which uses language L_1) to produce a new resource at URI U_2(which uses language L_2).

(b)

$$\begin{array}{l}
a(client(U_1, L, U_2), X) :: \\
\quad request\_abridgement(U_1, L) \Rightarrow a(abridger, R)\ then \\
\quad abridgement(U_2) \Leftarrow a(abridger, R) \\
a(abridger, R) :: \\
\quad request\_abridgement(U_1, L) \Leftarrow a(client(U_1, L, U_2), X)\ then \\
\quad abridgement(U_2) \Rightarrow a(client(U_1, L, U_2), X) \leftarrow abridge(U_1, L, U_2)
\end{array}$$

Where $abridge(U_1, L, U_2)$ is satisfied when a resource at URI U_1 (which uses language L) has been abridged to produce a new resource at URI U_2 (which also uses language L).

(c)

Fig. 4. Knowledge-transforming interaction models in LCC. (a) An interaction model for ontology collation. (b) An interaction model for language translation. (c) An interaction model for knowledge-base abridgement.

An LCC specification describes (in the style of a process calculus) a protocol for interaction between peers in order to achieve a collaborative task. The nature of this task is described through definitions of roles, with each role being defined as a separate LCC clause. The set of these clauses forms the LCC interaction model. An interaction model provides a context for each message passed between peers by describing the current state of the interaction (not of the peer) at the time of message sending. Coordination is achieved between peers by communicating this state along with the appropriate messages. Since roles are independently defined within an interaction model it is possible to distribute the computation to peers performing roles independently, with synchronization occurring only through message passing. Should the application demand it, however, LCC can also be used in more centralized, server-based style. The model for deriving properties of knowledge components that we present in this paper is independent of this choice.

Fig. 3 shows the main definitions of LCC's syntax. A detailed discussion of LCC, its semantics and the mechanisms used to deploy it, lies outside the scope of this paper. For these, the reader is referred to [32]. In this paper, though, we explain enough of LCC to demonstrate how to represent interactions. We return to our example and describe interaction models for each of the three tasks of Fig. 1: ontology collation (Fig. 4a), language translation (Fig. 4b), and knowledge base abridgement (Fig. 4c).

An interaction model in LCC is a set of clauses, each of which defines how a role in the interaction must be performed. Roles are described in the head of each clause by the type of role (and its parameters) and an identifier for the individual peer undertaking that role. The definition of performance of a role is constructed using combinations of the sequence operator ("*then*") or choice operator ("*or*") to connect messages and changes of role. Messages are terms, and are either outgoing to another peer in a given role ("$\Rightarrow$ ") or incoming from another peer in a given role ("$\Leftarrow$ "). Message input/output or change of role can be governed by a constraint to be solved before (when at the right of "$\leftarrow$ ") or after (when at the left of "$\leftarrow$ ") message passing or role change. Constraints are defined using the normal logical operators for conjunction, disjunction, and negation. If they are subject to fail, the interaction may proceed along alternative paths (e.g., those specified with operator "*or*"). Notice that there is no commitment to the system of logic through which constraints are solved—on the contrary we expect different peers to operate different constraint solvers.

The interaction models in Fig. 4 are generic in the sense that they give different interactions depending on how the variables (in uppercase) in the clauses are bound at runtime—this depending on the choices made by peers when satisfying the constraints within these clauses. A form of unfolding is used to record the current state of the interaction, in which a copy of each clause is used to store the current state of the corresponding role. Clauses may require subroles to be undertaken as part of the completion of a role and when this occurs the definition of the subrole is unfolded into the clause copy that stores the current state. Fig. 5 gives an example of an unfolded clause using the interaction model of Fig. 4a. We can "replay" the interaction by following the nested structure of the unfolded clause. Note the use of the term $c(M)$ to denote that a message passing event has been completed in the interaction.

Fig. 6 gives an overview of the basic LCC interpretation mechanism and its extension to the addition of property information. At the top of the diagram is the LCC

```
$a(collator(\mathrm{kif}, u_1, u_5), p_1) ::$
    $a(requester(\mathrm{kif}, \mathrm{hpkb}, u_1, u_3), p_1) ::$
        $c\Big(request\_onto\_source(\mathrm{kif}, \mathrm{hpkb}) \Rightarrow a(supplier, p_2)\Big)$ then
        $merge(\mathrm{kif}, u_1, u_2, u_3) \leftarrow$
            $c\Big(ontology\_source(\mathrm{kif}, \mathrm{hpkb}, u_2) \Leftarrow a(supplier, p_2)\Big)$ then
    $a(collator(\mathrm{kif}, u_3, u_5), p_1) ::$
        $a(requester(\mathrm{kif}, \mathrm{kifn}, u_3, u_5), p_1) ::$
            $c\Big(request\_onto\_source(\mathrm{kif}, \mathrm{kifn}) \Rightarrow a(supplier, p_2)\Big)$ then
            $merge(\mathrm{kif}, u_3, u_4, u_5) \leftarrow$
                $c\Big(ontology\_source(\mathrm{kif}, \mathrm{kifn}, u_4) \Leftarrow a(supplier, p_2)\Big)$
        $a(collator(\mathrm{kif}, u_5, u_5), p_1) ::$
            $c\Big(null\Big) \leftarrow completed(u_5) \wedge u_5 = u_5$
```

Peer p_1 performs the role of *collator* for a KIF ontology starting with URI u_1 and ending with URI u_5. To do this it became a *requester* that took u_1 and contacted peer p_2 (in the role of *supplier*) to obtain the URI for the HPKB ontology which p_1 then merged with u_1 to obtain u_3. A similar requesting process was used to merge u_3 with u_4 (the URI for the KIF-Numbers ontology) to obtain u_5 (which merges u_3 and u_4). The interaction terminated when p_1 considered u_5 to be completed.

Fig. 5. Unfolded interaction model for ontology collation.

interpreter which takes an interaction model plus (if the interaction is continuing rather than newly initiated) the unfolded interaction model recording the current interaction state, and updates the unfolded interaction model to record the new state. The interpreter is run on a peer and, as part of the peer's own mechanism for satisfying constraints, the peer interacts with knowledge components. As it does so, the appropriate component names are recorded in the unfolded interaction model (via variable matching on its constraints). The unfolded model is accessed by an interaction property inference system that is also given sets of bridging and property rules, enabling a property model for the knowledge components to be constructed. This property inference process is the subject of the remainder of this paper.

3 Knowledge Transformation

We have described how peers can store the temporal structure (the state) of interactions as an instance of an LCC interaction model. This is part of the solution to reasoning about interactions, since it enables peers to communicate in a standard way what the interaction required, but it is not the complete solution because it still does not allow to infer properties of the artifacts constructed during a knowledge-transforming interaction. There is no unique set of such properties appropriate to all contexts, so it is good engineering practice to allow specification of knowledge transformations and their properties in a way that allows them to be independently combined with interaction models (so we can mix and match properties to interactions as the application demands).

In this section, we introduce knowledge transformations by means of a real example of knowledge engineering. It will help us to highlight the need of formality we want to address and to justify the level of abstraction at which we will carry out the formalization. Then, in Section 4, we will describe how to connect abstract knowledge transformation rules with interaction models for property inference.

As said before, we do not attempt to give a catalog of knowledge transformations, as these will depend on the

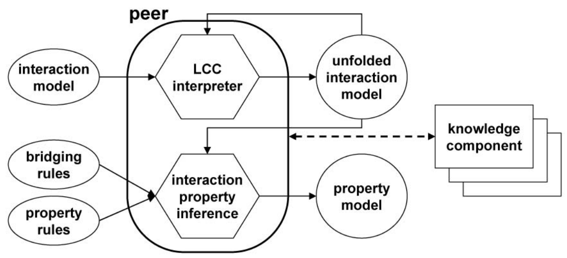


Fig. 6. Architectural overview for LCC interpretation with property inference.

particular application domain. In [35], [37], for instance, abstract transformation rules have been used to conduct generic knowledge-component brokering in a distributed environment.

3.1 The Boeing Experiment

Uschold, Haely, Williamson, Clark, and Woods describe an experiment of knowledge engineering conducted at Boeing Applied Research and Technology in which a layout design application of stiffened panels is enhanced with capabilities of unit conversion and dimensional analysis by reusing and deploying an ontology of engineering mathematics that is publicly available at the Ontolingua Server, hosted at Stanford University's Knowledge Systems Laboratory [39].

The context is a layout design application that determines the placement of "lightening holes" (for saving weight) on a panel, whose production code has been reverse engineered and reimplemented in Slang, the language of Specware, a system for the specification and formal development of software maintained at Kestrel Institute [38].

In order to enhance the application with the capabilities to perform unit conversion and dimensional analysis, Uschold et al., looked at the fragment of the Engineering Math Ontology that defined the concepts of Physical Quantity, Physical Dimension, Magnitude, Unit of Measure, and all necessary Algebraic Operations.

Since this ontology was represented in Ontolingua, an extension of KIF, a translation into Slang was necessary in order to be able to use the composition capabilities of Specware to combine the ontology with the specification of the application. Uschold et al., report that this alone was not sufficient in order to obtain the enhanced specification, because, although the engineering math ontology was designed to make it possible to perform these tasks, they were not explicitly specified in the ontology. Therefore, and previous to the final integration, these tasks had to be identified with a task-specific component and combined with the Slang specification of the ontology. Finally, the result of the combination was integrated into the specification of the panel layout application.

3.1.1 *Transforming Knowledge Components*

What the Boeing experiment illustrates is that rarely can a knowledge component, like the Engineering Math Ontology, be used *as is*, and that, in most cases, components need to be adapted in order to be combined and integrated satisfactorily into a knowledge-based system. In the Boeing experiment, this adaptation was primarily a manual and labor-intensive task that comprised:

1. the selection of a fragment of the Engineering Math Ontology;
2. the translation of the fragment into Slang;
3. the combination of the translated fragment with a Slang specification of the new tasks for the application;
4. the combination of ontology and tasks with the specification of the application.

The four items above are examples of what we will call in this paper *knowledge transformations*: transformations of representations that occur at the knowledge-level, i.e., transformations of knowledge representations, which we shall also call *knowledge components*. They also comprise three different sorts of knowledge transformation that we find useful to distinguish explicitly. The first sort takes a knowledge component and modifies its specification without changing the representation language in which the specification was written. In the Boeing experiment, it reduced to a very simple modification by removing unnecessary definitions and keeping only a selected fragment of the original specification. The second sort takes a knowledge component and rewrites it entirely into a new representation formalism. Unlike the previous sort of knowledge transformation, here one does not want the intended meaning of the specification to change, and an equivalent knowledge-level formulation is sought after; but obviously this will depend, among other things, upon the expressive power of the target representation language. According to Uschold et al., such equivalent translation from Ontolingua to Slang is generally feasible. Finally, the third and fourth transformations compose knowledge representations into a combined one. In the Boeing experiment, this was done using Specware's combination functionality, which is based on category-theoretic colimit construction.

3.1.2 The Need for a Formal Approach

Besides the need of adaptation, and the manual effort that this required, Uschold et al., also point at several problems they encountered during the experiment. For instance, with respect to the translation step carried out in their experiment Uschold et al., make the following observation:

> To be confident of mapping classes and subclasses to sorts and subsorts, semantic differences would have to be carefully checked. If inference were possible in one but not the other, work would have to be done to identify and contain the semantic differences responsible for such inconsistencies, in order to produce correct translations. We have not resolved this issue. [39, pages 184-185]

We are not going to resolve this issue here, either. Nevertheless we argue that any solution to this problem will have to tackle knowledge transformations formally, by specifying with precision what kind of translations a component has gone through. Only in this way we would *know* if inferences in the original component continue to be valid in the translated component.

Uschold et al., conclude that it would have taken significantly longer to design the knowledge content of the engineering math ontology from scratch in the panel layout application. Furthermore they point out that,

> if the engineering math ontology becomes to some degree a standard, there is the longer-term potential that this and other applications built with it can interoperate more easily, as they conform to the same physical-quantities vocabulary. [39, page 192]

But this interoperation is only possible if the applications *know* their provenance, namely that the engineering math ontology was, at some point in their respective knowledge-transformation histories, integrated by some knowledge transformation performed upon them. Furthermore, they also need to *know* if the entire ontology was used, or only a fragment, and in the latter case, which fragment. And they would also need to *know* how subsequent knowledge transformations performed on them (translations into other representation formalisms, combinations with other components, generalizations, specializations) may have affected the original axioms of the ontology.

In each of the above cases what is missing is an explicit representation of the *knowledge-transformation history* of a knowledge component, of the series of transformations that changed the knowledge representation of the component. In the remainder of this section, we propose to base such explicit representation on abstract knowledge-transformation rules.

3.2 Abstract Knowledge Components and their Transformations

By observing the sort of knowledge transformations occurring in knowledge engineering in general, and in in the Boeing experiment or the Ecolingua example of Fig. 1 in particular, we start adopting the following idealization for knowledge components:

An abstract knowledge component as a pair $C = \langle L, S\rangle$, consisting of:

- a knowledge representation *language* L;
- a *specification* S, which is written in L.

The language L determines the set of all *sentences* we can use in order to write S and is itself modeled as a pair $L = \langle I, O\rangle$, consisting of

- a *logical system* I, which fixes syntax and semantics of sentences;
- a *ontology* O that provides the necessary nonlogical vocabulary for writing down sentences.

In our example of Fig. 1, the ontologies obtained from the Ontolingua Server constitute abstract knowledge components whose specifications are written in the *Ontolingua* language. *Ontolingua* is based on KIF (providing a concrete syntax for first-order logic) and includes the additional nonlogical vocabulary required for specifying ontologies.

For the purposes of this paper, it will suffice to adopt an intuitive understanding of "language," "specification," "logical system," and "ontology," but for the formal treatment of properties in connection to interaction models as advocated in Section 2 we expect them to be modeled by some mathematical structure. For instance, the language L could be the system of first-order logic together with a particular logical theory as ontology, and S may be a subset of well-formed first-order sentences. In the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265, we provide one such formal foundation for abstract knowledge components in the framework of *institutions*[2] [14].

By saying that an *ontology* O provides the necessary nonlogical vocabulary for writing down sentences one might think that we treat an ontology as nothing more than a logical signature. A signature can certainly be seen as an ontology, but usually an ontology also constrains the possible meaning of the nonlogical symbols by additional axioms stated in an ontology representation language. If this is the case, it will be useful to consider an ontology itself as an abstract knowledge component $O = \langle L_O, S_O \rangle$ specified by S_O, written in language L_O. Here $L_O = \langle I_O, O_O \rangle$ will consist of a logical system I_O, which may or may not be the same as I. As we have said, L_O could well be a knowledge representation language specifically designed for representing ontologies, where I_O is different from I and where O_O is the metaontology that provides the necessary representation vocabulary.[3] In these cases some sort of "compatibility" between institutions I_O and I would be needed,[4] capturing the ontological commitment of language L to the conceptualization as specified by O. This general ontological commitment can be modeled for instance as an *institution morphism* [15], but a full treatment of this lies outside the scope of the paper. We refer the interested reader to [34].

In this paper it is our concern to investigate only transformations performed on knowledge components, assuming the above abstract characterization. Under these circumstances, given an abstract knowledge component $C = \langle L, S \rangle$, it is natural to distinguish between knowledge transformations that

1. revise the specification S;
2. translate the language $L = \langle I, O \rangle$, which itself can involve transformations that
   a. change the ontology O;
   b. change the logical system I.

In our example of Fig. 1, the deletion of extraneous clauses and pruning of the class hierarchy and of irrelevant clauses are sorts of "specification revisions," while the translation from Ontolingua into Prolog terms and from Prolog terms into executable Horn clauses are sorts of "language translations," and in particular they are based on changes of the underlying logical systems. In addition, reusing knowledge components from the Ontolingua Server and putting them together in Ecolingua also constitutes a sort of knowledge transformation, a sort of "component combination," for which we shall give an abstract treatment, too.

2. Institutions originated in the late 1970s and early 1980s for studying model-theoretic properties of logics and they have since given semantics to powerful module systems of both imperative and declarative programming languages, multilogic specification languages, databases, and ontologies. Most recently they have been also applied to provide logic-independent semantics to semantic web languages [24]. An institution captures the essential aspects of logical systems that underlie any formal specification of a computer program: a notion of a signature system, of well-founded formulas over a signature, and for each signature, notions of a system of models and a satisfaction relation between models and formulas.

3. The ontology representation language OWL is paradigmatic of this sort of nesting: I_O is RDF Schema and O_O is the vocabulary defined (in RDF Schema) at http://www.w3.org/2002/07/owl. An OWL ontology O then provides the terminology for a knowledge base S to be interpreted in the context of the logical system I of OWL, which is the $\mathcal{SHOIN}(D)$ description logic.

4. Making OWL downward compatibility with RDF Schema has been one of its major design issues.

3.3 Revising the Specification

We shall distinguish two sorts of knowledge transformations that revise the specification: those that perform a generalization and those that perform a specialization. Knowledge transformations that generalize the specification of a knowledge component are, for instance, belief-base contraction or those based on machine learning systems that generalize by rule induction on a set of examples. In the context of ontology engineering, transformations that fall into this category are, for instance, those that select subontologies by computing views over an ontology [27]. Other transformational systems do the opposite, specializing and refining a specification—for example, belief-base expansion or specialization operators of knowledge-base refiners. In the context of ontology engineering, we might think of those ontology reasoning services that infer additional subsumption relations between classes or that classify instances to particular classes [18], [19]. Out of this simple distinction will fall, of course, all those transformational systems that neither generalize nor specialize because they do some combination of both. In these cases, and in order to keep our set of knowledge transformation rules compact, we may consider them as systems that perform a sequence of more elementary transformational steps that eventually involve only generalization or only specialization.

The discussion above yields the distinction of our first two abstract knowledge-transformation rules:

$$\text{Specification Specialization (SS):} \quad \frac{\langle L, S \rangle}{\langle L, S' \rangle}, \quad \text{if} \quad S' \sqsubseteq S,$$

$$\text{Specification Generalization (SG):} \quad \frac{\langle L, S \rangle}{\langle L, S' \rangle}, \quad \text{if} \quad S' \sqsupseteq S.$$

In the rules above $S' \sqsubseteq S$ means that specification S' is *subsumed* by specification S (analogously, for $S' \sqsupseteq S$). The precise semantics of subsumption depends on the mathematical framework chosen to model knowledge components and transformations. In the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265, we provide one such semantics for subsumption in the institutional framework.

3.4 Translating the Language

Since we look at a language as constituted of a logical system and an ontology, we shall distinguish between two kinds of language translation, depending if either the ontology is changed—and so the vocabulary used and its intended meaning—or the logical system—and hence the construction of well-formed sentences and how they have to be interpreted.

Changing the language $L = \langle I, O \rangle$ of knowledge component $C = \langle L, S \rangle$ not only affects the language L, but also the specification S, because we may need to translate the specification according to the new language. We mentioned before that we do not commit to a specific mathematical structure used for languages and their constituent parts (although we provide one such structure in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265). But if there is a change from language L to, say, L', the structure that characterizes L' as a language constituted of a logical system and an ontology should be preserved. These structure-preserving transformations are called *morphisms* in mathematics, and, consequently, it is necessary to study the translation of languages in particular, and the transformation of knowledge components in general, by means of *language morphisms*. We shall denote with $L \xrightarrow{f} L'$ such a morphism; and we shall denote the application of this language translation to the specification S written in L with $f[S]$, which will produce a specification written in L'. We refer to the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265 for one possible precise formalization of language morphisms. The direction of the language morphism needs not to coincide with that of the knowledge transformation. A knowledge transformation due to a forward-translation of the language is called *Language Specialization* and one due to a backward-translation, *Language Generalization*. This is justified in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265.

Hence, when a knowledge is transformed due to a change of language, we shall distinguish between two knowledge-transformation rules:

$$\text{Language Specialization (LS):} \quad \frac{\langle L, S\rangle}{\langle L', S'\rangle}, \quad \text{if } \begin{array}{c} S' \sqsubseteq f[S] \\ L \xrightarrow{f} L' \end{array}$$

$$\text{Language Generalization (LG):} \quad \frac{\langle L, S\rangle}{\langle L', S'\rangle}, \quad \text{if } \begin{array}{c} f[S'] \sqsupseteq S \\ L \xleftarrow{f} L' \end{array}$$

In the rules above, the annotations $L \xrightarrow{f} L'$ and $L \xleftarrow{f} L'$ mean that the application of the rules is conditioned to the existence of a language morphism between the respective languages. In the next two sections, we distinguish how these arise when either the ontology or the logical system of the language changes. This is analogous to the distinction between what is also called *semantic* and *syntactic* translation [11].

3.4.1 Changing the Ontology

There are multiple ways to translate the language by changing its ontology: one may include, remove or rename concepts and relations, alter the concept hierarchy, or modify the set of ontological constraints, for instance. Typically, this sort of translation may be preceded by an ontology mapping process where two (or more) ontologies are matched to elicit semantically related concepts and relations [12], [21], [31]. The resulting map (or alignment) between ontologies is then fed into the knowledge transformation system that translates a knowledge component between the mapped ontologies.

In order to model the change of the ontology part of a language $L = \langle I, O\rangle$, we shall make the assumption that we do not consider an ontology as a knowledge component with its own language and specification part (see Section 3.2). Instead, we will adopt a "logician's perspective" and consider an ontology O to be a *presentation* of a logical theory $O = (\Sigma, \Gamma)$ with signature Σ and axioms Γ as if it had been expressed in the knowledge component's logical system I. This perspective is consistent with an abstract view of ontologies as an explicit account of the intended models of a logical language (see, for instance, [17]). If we focus only on structure-preserving transformations, or *morphisms*, we model the transformation from O to O' as an ontology morphism that arises from a map of signature elements of the signature Σ to those of a signature Σ', whenever the axioms in Γ are translated accordingly. We make this more precise in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265.

Were we to look at an ontology as a knowledge component $O = \langle L_O, S_O\rangle$ an ontology morphism could also arise because we refine its specification S_O, or when we change the underlying ontology representation language L_O (together with the corresponding translation of the ontological axioms S_O). For ease of presentation we only consider ontology morphisms as they arise within a unique ontology representation language.

3.4.2 Changing the Logical System

An example of a transformational system that acts upon the logical system of a knowledge component is Ontolingua [16], which uses KIF as an expressive *interlingua* through which knowledge components that are to share knowledge are translated back and forth. Other examples are the translation services provided by Protégé [28] to represent ontologies in various representation formats such as RDF(S) or OWL, or the translator into Horn clauses implemented by Brilhante and Robertson for our example of Fig. 1.

When changing the logical system we not only translate the sentences specifying our knowledge component into the syntax and grammar of the target logical language, but we also adopt a new understanding of how the translated sentences are to be interpreted, and how models may or may not satisfy these sentences. Furthermore, the translation has to be coherent with the way sentences are interpreted and satisfied by models. In the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265 we drawn form the exhaustive research on abstract characterizations of logical systems by means of *institutions* to model the change performed on logical systems underlying knowledge representation languages.

3.5 Combining Knowledge Components

So far we have discussed transformations that alter knowledge components by refining their specifications, or translating them into another knowledge representation language either by changing their ontologies or their underlying logical system. But, ultimately, in order to construct knowledge-intensive artifacts, knowledge engineers combine different knowledge components, for example by connecting web services, or by enhancing web services with semantic annotations drawing from ontologies. We shall distinguish between two sorts of component combination: union and intersection.

3.5.1 Union of Components

In order to make the union of two knowledge components, that happen to be specified in the same language, only their specifications need to be merged. This merging itself may be complex, but it is at a lower level of abstraction than that of the knowledge transformation rules considered in this section. In general, however, knowledge components will be specified using different ontologies or using separate underlying logical systems, and some sort of *alignment* between languages will be needed.

The issue of aligning two languages L_1 and L_2 is by itself a challenging knowledge engineering task and has been studied extensively (see, e.g., [12], [21], [31]). Since we model a language as a pair consisting of a logical system together with an ontology, we shall tackle the issue of aligning languages by distinguishing between the alignment of logical systems and the alignment of ontologies.

- *Aligning ontologies:* Assuming the specifications of the components to be connected are based on the same logical system, the alignment of two separate ontologies O_1 and O_2 can be modeled with the help of the notion of *bridge*—an agreed understanding that we model as a common ontology O_0 together with ontology morphisms $O_0 \xrightarrow{f_1} O_1$ and $O_0 \xrightarrow{f_2} O_2$. Zimmermann et al., provide a detailed discussion on abstract characterizations of ontology alignments in terms of bridge ontologies (using category theory) [44]. A very simple form of bridge is, for instance, when $O_0 = O_1$, f_1 is the identity morphism on O_1, and f_2 is an ontology morphism from O_1 to O_2, hence making O_2 a *global ontology* in which the specifications of components are merged together. Another trivial bridge is the empty bridge, that is, no explicit alignment between O_1 and O_2 is provided. In this case, the global ontology O is the disjoint union of O_1 and O_2. Typically, *bridge ontologies* are computed by means of ontology mapping techniques that feed into the merging process to provide the means to carry out a more informed merge than just a disjoint union [1].
- *Aligning logical systems:* If the logical systems I_1 and I_2 of the components differ, the connection has to be done through a logical system I that captures both. In the simplest case this may be one or the other, but if neither system is expressive enough to capture all possible statements of the other, some "combined" logical system has to be determined in order to connect both components. Interlinguas such as KIF, or more recently the Common Logic international standard [20], are examples of logical systems that attempt to play this role.

In any case, if the connection of separate knowledge components requires the alignment of ontologies or of logical systems or of both, this can be described as a connection through a *global language* L into which there exist language morphisms from the components' languages.

The abstract knowledge-transformation rule for the union of components, stated for two knowledge components and a *bridge language* L_0 (which may consist, for instance, of the least expressive of both logical systems and a bridge of ontologies) is the following:

Component Union (CU):

$$\frac{\langle L_1, S_1\rangle \quad \langle L_2, S_2\rangle}{\langle L, g_1[S_1] \sqcap g_2[S_2]\rangle,} \quad \text{if} \quad g_{1,2} : \left\{L_1 \xleftarrow{f_1} L_0 \xrightarrow{f_2} L_2\right\} \triangleright L.$$

In the rule above, $\{L_1 \xleftarrow{f_1} L_0 \xrightarrow{f_2} L_2\}$ is called an *alignment* of L_1 and L_2 through *bridge language* L_0. The annotation $g_{1,2} : \{L_1 \xleftarrow{f_1} L_0 \xrightarrow{f_2} L_2\} \triangleright L$ means that the application of the rule requires the existence of the above alignment together with language morphisms $g_1 : L_1 \to L$ and $g_2 : L_2 \to L$—called *injections* and satisfying $g_1 \circ f_1 = g_2 \circ f_2$—that allow us to express the knowledge components involved in the connection within a *global language* L that respects the morphisms from the bridge language L_0 into the respective component languages L_1 and L_2. And $g_1[S_1] \sqcap g_2[S_2]$ is a specification that "unifies" S_1 and S_2, and hence is subsumed by both $g_1[S_1]$ and $g_2[S_2]$. The choice of the operator "$\sqcap$" together with a precise account of these concepts is given in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265.

3.5.2 Intersection of Components

Dually to the need of making the union of knowledge components, in knowledge engineering it is often necessary to establish commonalities and differences of two knowledge components, e.g., for version tracking and maintenance [23], [26]. Consequently, Component Union has a dual Component Intersection rule which is stated for two knowledge components with respect to a *reference language* L. In the particular case of knowledge component evolution in the scope of one single language L, for instance, this language also plays the role of reference language.

Component Intersection (CI):

$$\frac{\langle L_1, S_1\rangle \qquad \langle L_2, S_2\rangle}{\langle L_0, g_1^{-1}[S_1] \sqcup g_2^{-1}[S_2]\rangle}, \qquad \text{if } g_{1,2} : L_0 \triangleleft \left\{L_1 \xrightarrow{f_1} L \xleftarrow{f_2} L_2\right\}.$$

In the rule above, $\{L_1 \xrightarrow{f_1} L \xleftarrow{f_2} L_2\}$ is called an *integration* of L_1 and L_2 with respect to *reference language* L. The annotation $g_{1,2} : L_0 \triangleleft \{L_1 \xrightarrow{f_1} L \xleftarrow{f_2} L_2\}$ means that the application of the rule requires the existence of the above integration together with language morphisms $g_1 : L_0 \to L_1$ and $g_2 : L_0 \to L_2$—called *projections* and satisfying $g_1 \circ f_1 = g_2 \circ f_2$—that allow us to express the knowledge components involved in the connection in a *shared language* L_0 that respects the morphisms from the component languages L_1 and L_2 into the reference language L. And $g_1^{-1}[S_1] \sqcup g_2^{-1}[S_2]$ is a specification that "intersects" S_1 and S_2, and hence subsumes both $g_1^{-1}[S_1]$ and $g_2^{-1}[S_2]$ (or, in other words, its image along g_1 subsumes S_1, and its image along g_2 subsumes S_2). The choice of the operator "$\sqcup$", the meaning of the inverse images g_1^{-1} and g_2^{-1} together with a precise account of these concepts is given in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265.

4 Inferring Properties from Interaction Models

Having defined in Section 3 a set of knowledge transformation rules, we now connect these to interaction models.

Knowledge-transformation predicates:

$$
\begin{array}{l}
ss(K\_Comp, K\_Comp)\\
sg(K\_Comp, K\_Comp)\\
ls(L\_Morph, K\_Comp, K\_Comp)\\
lg(L\_Morph, K\_Comp, K\_Comp) \mid\\
cu(Align, L\_Moprh, L\_Morph, K\_Comp, K\_Comp, K\_Comp)\\
ci(Integr, L\_Moprh, L\_Morph, K\_Comp, K\_Comp, K\_Comp)
\end{array}
$$

Knowledge-component terms:

$$
\begin{array}{rcl}
K\_Comp & ::= & c(Lang, Spec)\\
Lang & ::= & l(Log\_Sys, Onto)\\
Onto & ::= & K\_Comp \mid o(Sig, Ax)\\
L\_Morph & ::= & m(Fun, Lang, Lang)\\
Align & ::= & a(L\_Morph, L\_Morph)\\
Integr & ::= & i(L\_Morph, L\_Morph)
\end{array}
$$

Arguments in predicates and terms can also be dereferencable identifiers. (This is always the case for *Spec*, *Log_Sys*, *Sign*, *Ax*, and *Fun*.)

Fig. 7. Syntax of knowledge transformations and components.

Although both interaction models and rules are (inevitably) application specific, the mechanism for combining them is generic. We begin by restating knowledge transformation rules in a more convenient syntactic form as predicates over terms (see Fig. 7) in order to link them to the LCC language in a more direct manner.

Terms represent the basic entities of our conceptualization (knowledge components, languages, ontologies, etc.), and are generated over dereferencable identifiers (such as URIs) pointing to the actual entities, and also over concrete constant symbols. To specify the properties on the components involved in a knowledge transformation, we write a *property rule* for each knowledge-transformations type in Section 3. We do this by moving all syntactic constraints into the premise of a rule and by specifying the actual properties in the conclusion, dereferencing identifiers that occur in the premise (see Fig. 8).

Notice that in interaction models (for example those of Fig. 4) the only points at which peers can access or transform knowledge components are the constraints associated with message input and output. Therefore, it is necessary to bridge between constraints in interaction models with sets of knowledge transformations. Fig. 9 gives an example set of bridging rules for the constraints in Fig. 4.

The first of these rules (bridging the constraint *merge* in interaction model Fig. 4a) says that, if resources at URIs U and U_n are merged to form a new one at U_m, then the knowledge component with language referenced by L and specification at U_m is a component union of the components of the same language and with specifications at URIs U, U_n, and U_m, respectively. This union is with respect to the trivial alignment $\{*L \overset{id}{\leftarrow} *L \overset{id}{\rightarrow} *L\}$, via the identity language morphism for the language referenced by L.

$$
\begin{array}{l}
\mathcal{S} \models merge(L, U, U_n, U_m)\\
\qquad \Longrightarrow \Big\{ cu\Big(a(m(id, L, L), m(id, L, L)), m(id, L, L), m(id, L, L),\\
\qquad\qquad c(L, U), c(L, U_n), c(L, U_m)\Big)\Big\}\\[1ex]
\mathcal{S} \models translate(F, U_1, L_1, U_2, L_2)\\
\qquad \Longrightarrow \Big\{ ls\Big(m(F, L_1, L_2), c(L_1, U_1), c(L_2, U_2)\Big)\Big\}\\[1ex]
\mathcal{S} \models abridge(U_1, L, U_2)\\
\qquad \Longrightarrow \Big\{ sg\Big(c(L, U_1), c(L, U_1)\Big)\Big\}
\end{array}
$$

Where the expression $\mathcal{S} \models T$ denotes that the constraint T holds in state $\mathcal{S}$ of the interaction, and '*id*' is the functor symbol for identity morphisms.

Fig. 9. Bridging rules between interaction models and transformations.

The second rule (bridging the constraint *translate* in interaction model Fig. 4b) says that, if the resource at U_2 is a translation of the one at U_1 (by function at F from the language at L_1 into the language at L_2), then the knowledge component with language at L_2 and specification at U_2 is a language specialization of the component with language at L_1 and specification at U_1. The specialization is with respect to language morphisms $*L_1 \overset{*F}{\rightarrow} *L_2$.

The third rule (bridging the constraint *abridge* in interaction model Fig. 4c) says that, if the resource at U_2 (written in the language referenced by L) is an abridgement of U_1, then the component with language at L and specification at U_2 is a generalization of the component of the same language and with specification at U_1.

We now have the representational elements we need: a means of obtaining a model of the current state of interaction (e.g., Fig. 5); bridges relating sets of knowledge transformations to constraints in the model (e.g., Fig. 9); and rules stating properties associated with these knowledge transformations (e.g., Fig. 8). Finally, it is necessary to provide a mechanism for determining when a constraint T actually holds in a particular interaction state $\mathcal{S}$ (i.e., $\mathcal{S} \models T$). The definition of $\models$ is given in Fig. 10. This is essentially an interpreter that lifts out appropriate constraints that are

$$
\begin{array}{lll}
ss(c(L, S_1), c(L, S_2)) \longrightarrow & *S_2 \sqsubseteq *S_1 & \\
sg(c(L, S_1), c(L, S_2)) \longrightarrow & *S_2 \sqsupseteq *S_1 & \\[1ex]
ls(m(F, L_1, L_2), c(L_1, S_1), c(L_2, S_2)) \longrightarrow & *S_2 \sqsubseteq (*F)[*S_1] & \\
lg(m(F, L_2, L_1), c(L_1, S_1), c(L_2, S_2)) \longrightarrow & (*F)[*S_2] \sqsupseteq *S_1 & \\[1ex]
cu(a(m(F_1, L_0, L_1), m(F_2, L_0, L_2)), & & \\
\quad m(G_1, L_1, L), m(G_2, L_2, L), & & \\
\quad c(L1, S1), c(L_2, S_2), c(L, S)) \longrightarrow & *S \sqsubseteq (*G_1)[*S_1] \wedge & \\
 & *S \sqsubseteq (*G_2)[*S_2] \wedge & \\
 & (*G_1) \circ (*F_1) = (*G_2) \circ (*F_2) & \\
ci(i(m(F_1, L_1, L), m(F_2, L_2, L)), & & \\
\quad m(G_1, L_0, L_1), m(G_2, L_0, L_2), & & \\
\quad c(L1, S1), c(L_2, S_2), c(L_0, S)) \longrightarrow & (*G_1)[*S] \sqsupseteq *S_1 \wedge & \\
 & (*G_2)[*S] \sqsupseteq *S_2 \wedge & \\
 & (*G_1) \circ (*F_1) = (*G_2) \circ (*F_2) &
\end{array}
$$

Where '$*$' is the dereference operator.

Fig. 8. Property rules for knowledge-transformation types.

$$
\begin{array}{rcl}
\mathcal{S} \models T & \text{if} & E \in \mathcal{S} \ \wedge \ E \models T\\
(R :: B) \models T & \text{if} & B \models T\\
(A_1 \ or \ A_2) \models T & \text{if} & \neg closed(A_2) \ \wedge \ A_1 \models T\\
(A_1 \ or \ A_2) \models T & \text{if} & \neg closed(A_1) \ \wedge \ A_2 \models T\\
(A_1 \ then \ A_2) \models T & \text{if} & A_1 \models T\\
(A_1 \ then \ A_2) \models T & \text{if} & closed(A_1) \ \wedge \ A_2 \models T\\
(C \ \leftarrow \ M_{in}) \models T & \text{if} & closed(M_{in}) \ \wedge \ subterm(C, T)\\
(M_{out|null} \ \leftarrow \ C) \models T & \text{if} & closed(M_{out|null}) \ \wedge \ subterm(C, T)
\end{array}
$$

Where E is the expression of a (partially) unfolded interaction model and $subterm(C, T)$ holds when T is a goal contained in constraint C. An expression E is decided to be closed as follows (recall that $c(X)$ denotes that an event X has been completed):

$$
\begin{array}{l}
closed(c(X))\\
closed(A \ or \ B) \ \leftarrow \ closed(A) \ \vee \ closed(B)\\
closed(A \ then \ B) \ \leftarrow \ closed(A) \ \wedge \ closed(B)\\
closed(X :: D) \ \leftarrow \ closed(D)
\end{array}
$$

Fig. 10. Definition of "holds" predicate $\mathcal{S} \models T$.

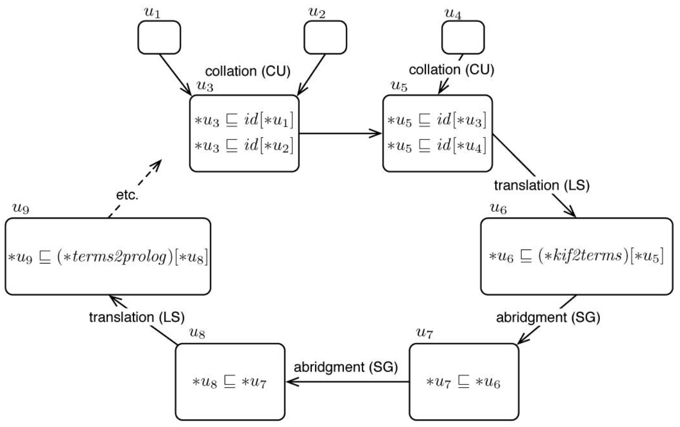

u_1
u_2
u_4
collation (CU)
collation (CU)
u_3
$*u_3 \sqsubseteq id[*u_1]$
$*u_3 \sqsubseteq id[*u_2]$
u_5
$*u_5 \sqsubseteq id[*u_3]$
$*u_5 \sqsubseteq id[*u_4]$
etc.
translation (LS)
u_9
$*u_9 \sqsubseteq (*terms2prolog)[*u_8]$
u_6
$*u_6 \sqsubseteq (*kif2terms)[*u_5]$
translation (LS)
abridgment (SG)
u_8
$*u_8 \sqsubseteq *u_7$
abridgment (SG)
u_7
$*u_7 \sqsubseteq *u_6$

Fig. 11. Sequence of property information in Ecolingua's lifecycle.

buried in closed (i.e., completed) sections of (possibly partially) unfolded interaction models (which represent the interaction state).

For example, assuming the state $\mathcal{S}$ to be the set containing the unfolded interaction model in Fig. 5, we have that:

$$\mathcal{S} \models merge(\text{kif}, u_1, u_2, u_3), \qquad \mathcal{S} \models completed(u_5),$$
$$\mathcal{S} \models merge(\text{kif}, u_3, u_4, u_5), \qquad \mathcal{S} \models u_5 = u_5.$$

Applying the bridging rules of Fig. 9 that are applicable to the constraints holding in $\mathcal{S}$ (in our case to the two on the left) we obtain the knowledge transformations represented by the following two ground predicates:

$$cu(a(m(id, kif, kif), m(id, kif, kif)),\ m(id, kif, kif),$$
$$m(id, kif, kif),\ c(kif, u_1),\ c(kif, u_2),\ c(kif, u_3)),$$
$$cu(a(m(id, kif, kif), m(id, kif, kif)),\ m(id, kif, kif),$$
$$m(id, kif, kif),\ c(kif, u_3),\ c(kif, u_4),\ c(kif, u_5)).$$

These then allow us, using the rules of Fig. 8, to infer the following properties about the referenced entities:[5]

$$*u_3 \sqsubseteq id[*u_1], \qquad *u_5 \sqsubseteq id[*u_3],$$
$$*u_3 \sqsubseteq id[*u_2], \qquad *u_5 \sqsubseteq id[*u_4].$$

If we apply this same process across the interactions described for the entire sequence of knowledge transformations of Ecolingua's lifecycle described in Section 1, we obtain property information for each of the nine resources of the lifecycle, depicted in Fig. 11. The boxes in this diagram enclose the property information (except for trivial properties) about the resources pointed by URIs. Arcs are labeled with the knowledge transformation (and its type in parentheses) of the Ecolingua lifecycle (see Fig. 1). The detailed example that we described above corresponds to the first two boxes (at top) of the diagram.

The procedure above allows information about properties of resources at URIs to be accumulated as part of the interactions in which those URIs are involved. By adding further axioms defining a background theory of relationships between properties and their semantics (drawn from a theory such as the one of the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.265) we can "mine" further information that is not immediately obvious from this repository.

For example, we define below the predicate $contains(U_1, U_2)$, which holds when the information specified at URI U_2 is included in (i.e., can be inferred from) that specified at URI U_1:

$$contains(U, U),$$
$$contains(U_1, U_2) \leftarrow *U_1 \sqsubseteq *U_2 \ \vee\ *U_1 \sqsubseteq id[*U_2],$$
$$contains(U_1, U_2) \leftarrow contains(U_1, U_3) \ \wedge\ contains(U_3, U_2).$$

Taking these along with the information in Fig. 11 we can, for example, infer that $contains(u_5, u_1)$ and $contains(u_6, u_8)$ hold, but that $contains(u_9, u_1)$ does not hold. In this way, we can explore chains of relationships between URIs without needing to look at the actual content of the URIs themselves.

5 Conclusion

We have addressed a need of formality for providing automated reasoning support for knowledge transformations that occur in the context of peer interactions. Our aim of formality in this area has been twofold: to give a concise account of what is going on, and to use this account for practical purposes in handling and maintaining states of distributed knowledge-transforming interactions. As more and more knowledge components such as ontologies, web services, and databases are made publicly available on the Internet, it becomes natural and necessary to study and record the unfolding of knowledge-transforming interactions. Peers with the ability to understand these interactions and to infer from the state of an interaction if certain key properties of knowledge components are preserved or not—as shown in this paper—may consequently decide their actions accordingly, without requiring to inspect knowledge representations in the knowledge components themselves.

Others have explored similar problems in more conventional architecture settings. Perhaps the best known of these is in the database community, where data provenance is a key issue (see for example [7]). For this community, provenance is a form of metadata that is used to record the derivation history and lineage of data sets. No generic one-size-fits-all solution has been discovered for propagating

5. We leave out the obviously a trivial consequence $id \circ id = id \circ id$.

and maintaining provenance information. Instead, practical solutions are always instances of general methods or frameworks that are produced for a given domain or task. The methods we provide in this paper can be understood (in a peer-based setting) in a similar spirit. The framework we supply is based on a process calculus and in this sense is similar to provenance propagation efforts based on process languages from the e-Science community (a recent survey of these appears in [36]). Our contribution here is to show how these efforts may be reoriented (in a general way) to peer-based architectures.

We have focused on the connection of interaction models with property rules, and to describe this connection in a generic way that is independent of domain-specific particularities of knowledge-transforming services. On one hand, this has led us to look at knowledge transformation from a very abstract angle, grounding its semantics on the heavy-weight formal theory of institutions. On the other hand, the generic way in which we have defined this connection shows that heavy-weight formal theories for rich property inference need not be at odds with a light-weight formal approach to interaction modeling such as the one based on LCC.

Actually, as the web (and the semantic web) are continuously challenging the way we see and carry out software and knowledge engineering, we are faced with the necessity to shift our focus to an interaction-centred, open-ended engineering paradigm. Traditional engineering techniques need to be rethought along with their underlying mathematical models, but this paper also serves as an example of the gain obtained by linking current state-of-the-art distributed knowledge engineering based on web services and peer-based architectures to formal methods resulting from a long tradition in algebraic specification going back to the early 1,970s. Its abstractness and domain-independence gains new momentum in the context of these new challenges. However, this paper is the first description of this form of activity that is both practical (in the sense that the system we describe is supported by an existing peer-based knowledge sharing system [30]) and has a formal semantics (in the sense that both interaction and transformation components are given a declarative description).

Acknowledgments

This work is supported under EU project OpenKnowledge (FP6-027253), under Spain's Consolider-Ingenio 2010 project Agreement Technologies (CSD2007-00022), and by the Generalitat de Catalunya (2009-SGR-1434).

References

[1] P. Bernstein, A. Halevy, and R. Pottinger, "Model Management: Managing Complex Information Structures," *ACM Special Interest Group on Management of Data,* vol. 29, no. 4, pp. 55-63, 2000.

[2] M. Bonifacio, P. Bouquet, and R. Cuel, "Knowledge Nodes: The Building Blocks of a Distributed Approach to Knowledge Management," *J. Universal Computer Science,* vol. 8, no. 6, pp. 652-661, 2002.

[3] M. Bonifacio, P. Bouquet, G. Mameli, and M. Nori, "Peer-Mediated Distributed Knowledge Management," *Proc. Int'l Symp. Agent-Mediated Knowledge Management (AMKM '03),* Revised and Invited Papers, LNCS, vol. 2926, pp. 31-47, 2003.

[4] M. Bonifacio, P. Bouquet, and P. Traverso, "Enabling Distributed Knowledge Management: Managerial and Technological Implications," *Informatik/Informatique,* vol. III, no. 1, pp. 24-30, 2002.

[5] M. Bonifacio, P. Camussone, and C. Zini, "Managing the KM Trade-Off: Knowledge Centralization *versus* Distribution," *J. Universal Compter Science,* vol. 10, no. 3, pp. 162-175, 2004.

[6] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems,* vol. 8, no. 3, pp. 203-236, 2004.

[7] P. Buneman, S. Kharma, and W. Tan, "Why and Where: A Characterisation of Data Provenance," *Proc. Eighth Int'l Conf. Database Theory (ICDT),* LNCS, vol. 1973, pp. 316-330, 2001.

[8] F. Corra da Silva and J. Agustí, *Knowledge Coordination.* Wiley, 2003.

[9] F. Corra da Silva, W. Vasconcelos, D. Robertson, V. Brilhante, A. de Melo, M. Finger, and J. Agustí, "On the Insufficiency of Ontologies: Problems in Knowledge Sharing and Alternative Solutions," *Knowledge-Based Systems,* vol. 15, no. 3, pp. 147-167, 2002.

[10] K. Devlin, *InfoSense: Truning Information into Knowledge.* W.H. Freeman and Company, 2001.

[11] D. Dou, D. McDermott, and P. Qi, "Ontology Translation on the Semantic Web," *J. Data Semantics II,* LNCS, vol. 3360, pp. 35-57, 2005.

[12] J. Euzenat and P. Shvaiko, *Ontology Matching.* Springer, 2007.

[13] A. Farquhar, R. Fikes, and J. Rice, "The Ontolingua Server: A Tool for Collaborative Ontology Construction," *Int'l J. Human-Computer Studies,* vol. 46, no. 6, pp. 707-727, 1997.

[14] J. Goguen and R. Burstall, "Institutions: Abstract Model Theory for Specification and Programming," *J. ACM,* vol. 39, no. 1, pp. 95-146, 1992.

[15] J. Goguen and G. Roşu, "Institution Morphisms," *Formal Aspects of Computing,* vol. 13, pp. 204-307, 2002.

[16] T. Gruber, "A Translation Approach for Portable Ontology Specifications," *Knowledge Eng.,* vol. 5, no. 2, pp. 199-220, 1993.

[17] N. Guarino, "Understanding, Building and Using Ontologies," *Int'l J. Human-Computer Studies,* vol. 46, pp. 293-310, 1997.

[18] V. Haarslev and R. Möller, "RACER System Description," *Proc. Int'l Joint Conf. Automated Reasoning (IJAR),* LNCS, vol. 2083, pp. 701-705, 2001.

[19] I. Horrocks, "Using an Expressive Description Logic: FaCT of Fiction?," *Proc. Sixth Int'l Conf. Principles of Knowledge Representation and Reasoning,* pp. 636-647, June 1998.

[20] Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 32, "Information Technology—Common Logic (CL): A Framework for a Family of Logic-Based Languages," *Int'l Standard ISO/IEC 24707:2007(E),* 2007.

[21] Y. Kalfoglou and M. Schorlemmer, "Ontology Mapping: The State of the Art," *The Knowledge Eng. Rev.,* vol. 18, no. 1, pp. 1-31, 2003.

[22] R.E. Kent, "The Information Flow Foundation for Conceptual Knowledge Organization," *Proc. Sixth Int'l Conf. Int'l Soc. for Knowledge Organization,* July 2000.

[23] M. Klein and N. Noy, "A Component-Based Framework for Ontology Evolution," *Proc. IJCAI '03: Workshop Ontologies and Distributed Systems,* 2003.

[24] D. Lucanu, Y.F. Li, and J.S. Dong, "Semantic Web Languages—Towards an Institutional Perspective," *Algebra, Meaning, and Computation,* LNCS, vol. 4060, pp. 99-123, Springer, 2006.

[25] T. Miller and J. McGinnis, "Amongst First-Class Protocols," *Proc. Eighth Int'l Workshop Eng. Societies in the Agents World VIII,* Revised Selected Papers, LNCS, vol. 4995, pp. 208-223, 2008.

[26] N. Noy and M. Musen, "Ontology Versioning in an Ontology-Management Framework," *IEEE Intelligent Systems,* vol. 19, no. 4, pp. 6-13, July/Aug. 2004.

[27] N. Noy and M. Musen, "Specifying Ontology Views by Traversal," *Proc. Third Int'l Semantic Web Conf. Semantic Web (ISWC),* LNCS, vol. 3298, pp. 713-725, 2004.

[28] N. Noy, M. Sintek, S. Decker, M. Crubezy, R. Fergerson, and M. Musen, "Creating Semantic Web Contents with Protégé-2000," *IEEE Intelligent Systems,* vol. 16, no. 2, pp. 60-71, Mar./Apr. 2001.

[29] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li, "Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows," *Bioinformatics,* vol. 20, no. 17, pp. 3045-3054, 2004.

[30] A. Perreau de Pinninck, D. Dupplaw, S. Kotoulas, and R. Siebes, "The OpenKnowledge Kernel," *Int'l J. Applied Math. and Computer Sciences,* vol. 4, no. 3, pp. 162-167, 2007.

[31] E. Rahm and P.A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *The Int'l J. Very Large Data Bases,* vol. 10, no. 4, pp. 334-350, 2001.

[32] D. Robertson, "Multi-Agent Coordination as Distributed Logic Programming," *Proc. 20th Int'l Conf. Logic Programming (ICLP),* LNCS, vol. 3132, pp. 416-430, 2004.

[33] D. Robertson, F. Giunchiglia, F. van Harmelen, M. Marchese, M. Sabou, M. Schorlemmer, N. Shadbolt, R. Siebes, C. Sierra, C. Walton, S. Dasmahapatra, D. Dupplaw, P. Lewis, M. Yatskevich, S. Kotoulas, A. Perreau de Pinninck, and A. Loizou, "Open Knowledge—Coordinating Knowledge Sharing through Peer-to-Peer Interaction," *Proc. First Int'l Workshop Languages, Methodologies and Development Tools for Multi-Agent Systems (LADS),* LNAI, vol. 5118, pp. 1-18, 2008.

[34] M. Schorlemmer and M. Atencia, "Semantic Alignment in the Context of Agent Interaction," *Proc. Workshop Formal Approaches to Multi-Agent Systems (FAMAS '07),* pp. 117-134, Sept. 2007.

[35] M. Schorlemmer, S. Potter, D. Robertson, and D. Sleeman, "Knowledge Life-Cycle Management Over a Distributed Architecture," *Expert Update,* vol. 5, no. 3, pp. 2-19, 2002.

[36] Y. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in e-Science," *ACM Special Interest Group on Management of Data,* vol. 34, no. 3, pp. 31-36, 2005.

[37] D. Sleeman, S. Potter, D. Robertson, and M. Schorlemmer, "Ontology Extraction in Distributed Environments," *Knowledge Transformation for the Semantic Web: Frontiers in Artificial Intelligence and Applications,* vol. 95, pp. 80-91, IOS Press, 2003.

[38] Y.V. Srinivas and R. Jüllig, "Specware: Formal Support for Composing Software," *Proc. Conf. Math. of Program Construction (MPC '95),* LNCS, vol. 947, pp. 399-422, 1995.

[39] M. Uschold, M. Healy, K. Williamson, P. Clark, and S. Woods, "Onology Reuse and Application," *Proc. First Int'l Conf. Formal Ontology in Information Systems (FOIS '98),* Frontiers in Artificial Intelligence and Applications, vol. 46, pp. 179-192, 1998.

[40] J. van Diggelen and V. Dignum, "Special Issue on Agent-Mediated Knowledge Management," *Int'l J. Knowledge-Based and Intelligent Eng. Systems,* vol. 10, no. 4, pp. 259-261, 2006.

[41] L. van Elst, V. Dignum, and A. Abecker, eds., *Proc. Int'l Symp. Agent-Mediated Knowledge Management (AMKM '03),* Revised and Invited Papers, LNCS, vol. 2926, 2004.

[42] S. Willmott, G. Vreeswijk, M. South, C. Chesñevar, G. Simari, J. McGinis, and I. Rahwa, "Towards an Argument Interchange Format for Multiagent Systems," *Proc. Third Int'l Workshop Argumentation in Multi-Agent Systems,* 2006.

[43] M. Wooldridge, N.R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems,* vol. 3, pp. 285-312, 2000.

[44] A. Zimmermann, M. Krötzsch, J. Euzenat, and P. Hitzler, "Formalizing Ontology Alignment and Its Operations with Category Theory," *Proc. Int'l Conf. Formal Ontology in Information Systems (FOIS),* Frontiers in Artificial Intelligence and Applications, vol. 150, pp. 277-288, 2006.

Marco Schorlemmer received the licenciate and doctorate degrees in informatics from the Technical University of Catalonia. He is currently a tenured scientist at CSIC's Artificial Intelligence Research Institute in Barcelona. He is an author of more than 50 publications in the fields of formal specification and automated theorem proving, diagrammatic representation and reasoning, distributed knowledge coordination, and semantic interoperability of ontologies.

David Robertson received the BSc Hons degree in ecological science from the University of Edinburgh and the PhD degree in artificial intelligence from the Autonomous University of Barcelona. He is a professor and the head of the School of Informatics at the University of Edinburgh. He has been a principal investigator on more than 10 national and international projects and is an author of more than 100 publications on the design and coordination of multiagent systems.