Tool Interoperability from the Trenches: the Case of DOGMA-MESS

Stijn Christiaens and Aldo de Moor *

VUB STARLab
Semantics Technology and Applications Research Laboratory
Vrije Universiteit Brussel
Pleinlaan 2 B-1050 Brussels, Belgium
stichris/ademoor@vub.ac.be

Abstract. Whereas conceptual structures tools have great potential to facilitate and improve collaboration in knowledge-intensive communities, so far few examples exist of them being successfully used in actual, messy practice. We contend that one major reason is that these tools are often developed and deployed in isolation from other information system components. In this paper, we examine tool interoperability as a gradual process of co-evolution of the requirements and the functionality components that a collaborative community uses. The result is a socio-technical system increasingly tailored to the community's needs. In these systems, conceptual structures tools are embedded in functionalities closer to the end users' needs, including functionalities optimized for input, output, data management, and visualization. By presenting the case of the initial stages of the development of the DOGMA-MESS interorganizational ontology engineering system, we examine some of the tool interoperability problems impeding the development of useful socio-technical knowledge systems that can serve complex knowledge requirements of realistic communities of practice.

1 Introduction

Over time, a wide range of conceptual structures tools has been developed. These tools have different formal origins (e.g. Conceptual Graphs [10], Formal Concept Analysis [4], or Description Logics [1]) and satisfy different knowledge needs (ranging from knowledge structure representation and visualization, to supporting complex reasoning and analysis operations).

Whereas conceptual structures tools have great *potential* to facilitate and improve collaboration in knowledge-intensive communities, few examples exist so far of them being successfully used in actual, messy practice. We contend that one major reason is that these tools are often developed and deployed in *isolation* from other information system components. Although knowledge representation and analysis functionalities are core elements of today's information systems, they are not sufficient. Many

* The research described in this paper was partially sponsored by EU Leonardo da Vinci CODRIVE project B/04/B/F/PP-144.339 and the DIP EU-FP6 507483 project. The authors wish to thank Ruben Verlinden for his aid in the development of the DOGMA-MESS system, and Ulrik Petersen for his work on upgrading the functionality of Prolog+CG

other functionality components are needed to implement working and usable information systems, including functionalities optimized for input, output, data management, and visualization.

Two fundamentally different strategies are conceivable to address this problem: (1) to develop all functionalities in-house, thus in effect building a customized information system from scratch; (2) to view an information system as an ensemble of existing functionality components such as tools and web services that need to be combined into distributed systems of properly selected, linked, and configured components. This second strategy of component- and service based systems development is starting to prevail [8, 3].

Whereas *systems* development is becoming more oriented towards alignment and integration of functionality components, many *tools* are still created as if they exist all alone in the world [12]. Many tools can be characterized by proprietary data formats, incompatible interfaces, and functionalities that only partially support the workflow requirements of their users. To make things worse, these requirements are increasingly becoming collaborative, whereas many tools still focus on the needs of an amorphous end-user. Thus, although tools generally can be used stand-alone, they only really become useful for satisfying real collaborative applications when indeed seen as part of a system.

Tools, often necessarily so, suboptimize a particular type of functionality. A database is particularly good at managing structured data, a web forum at supporting asynchronous, distributed discussion, and so on. Similarly so for conceptual structures tools. These often specialize in particular forms of formal knowledge representation, visualization, and analysis. To better understand their role in the bigger picture, we start investigating them from a web services perspective. Web services can be loosely defined as self-describing, interoperable and reusable business components that can be published, orchestrated and invoked through the Internet, even when they reside behind a company's firewall [3].

Let us interpret these aspects in detail. Properties like self-describing, interoperable, and reusable are necessary conditions for services, or tools, to be put as meaningful pieces in the larger systems puzzle. However, what *makes* them part of this larger puzzle? Much theory on web services composition remains in the abstract. What kind of tool interoperability problems occur in practice? What kind of ad hoc solutions are being developed to interoperability problems? How can these lessons learnt be systematized? To what extent do conceptual structures tools pose specific interoperability problems?

Much anecdotal evidence of interoperability problems exists. However, comprehensive frameworks to describe, compare, and analyze interoperability problems across cases are lacking. This means that many projects keep struggling with these issues, without making much progress and applying relevant lessons learnt in other cases. The goal of this paper is to outline a simple tool interoperability problems comparison framework that can help focus on and learn from interoperability experiences.

By presenting the case of the initial development of the DOGMA-MESS interorganizational ontology engineering system, we examine some of the conceptual structures tool interoperability problems impeding the development of holistic and useful

knowledge systems that can serve real-world knowledge requirements of communities of practice.

In our analysis, we examine tool interoperability as a gradual co-evolution process of the requirements and the functionality components which a collaborative community uses. The result is a continuously developing socio-technical system increasingly tailored to the community's needs. In Sect. 2, we introduce a conceptual model of tool interoperability. Sect. 3 uses this model to sketch the main interoperability problems encountered over time in the case of DOGMA-MESS. Sect. 4 focuses on one problem in more detail: how to integrate a standalone conceptual graphs tool into the larger ontology engineering system? We end the paper with conclusions.

2 A Conceptual Model of Tool Interoperability

What is interoperability? A very simple definition is that it is the need to make heterogeneous information systems work in the networked world [13]. However, this definition does not yet explain how this need can be satisfied. A more both operational and extensive definition is given by Miller in [6], who defines it as the ongoing process of ensuring that the systems, procedures, and culture of an organisation are managed in such a way as to maximise opportunities for exchange and re-use of information, whether internally or externally. Thus, not only technical system and process aspects need to be taken into account, but also the culture of the users. A community of knowledge engineers will be much more forgiving about importing raw data from a formal knowledge analysis tool than a group of business users who need the knowledge representations in their decision making processes. Also, two important goals in interoperability are the exchange of information and the potential for reusability in applications not foreseen in their elicitation.

Given the vast amount of interoperability aspects (system, procedure, and culture-wise), what is a good conceptualization of tool interoperability issues? How to simultaneously take account of the incredible variety, while also discovering enough common ground to recognize patterns in interoperability problems and solutions? In this paper, we present a simple framework that we used to reflect on our experiences with a very real knowledge system: the interorganizational ontology engineering system DOGMA-MESS. The framework is coarse and premature. However, we have found it useful to order our experiences, so we present it, as well as the results of the case analyzed, as one way of coming to grips with the complexity. According to Hasselbring [5] there are three viewpoints to information system interoperability: application domain, conceptual design and software systems technology. Our framework is focused on supporting the conceptual design viewpoint.

Two important dimensions of our framework are *functionality requirements* and *functionality components*. Functionality requirements (i.e. conceptual descriptions of required operations on data, information, and knowledge, such as data management, knowledge visualization etc.) are to be distinguished from the components in which these functionalities are implemented (e.g. applications, modules, web services, information systems). We contend that mixing up requirements and components is a common source of interoperability problems. A third dimension are *interoperability aspects*.

These we loosely define as those aspects taken into account by system developers when making specification and implementation decisions. The combination of functionality requirements, functionality components, and interoperability aspects gives us enough structure to describe and analyze our tool interoperability problems encountered in practice.

2.1 Functionality Requirements

What types of high-level functionalities does a knowledge-based information system usually comprise?

First, there are some conceptual structures-specific functionalities, which all revolve around knowledge processes. The core strength of conceptual structures tools is in *knowledge representation and analysis*. Another very important category of functionalities is *knowledge visualization*. *Knowledge elicitation* from users, as well as *knowledge interpretation* by users are other necessary knowledge functionalities. Finally, *knowledge use* is a separate process of selecting and presenting relevant knowledge to human or machine users, for example using that (and only that) knowledge that is most useful for enriching particular business processes or workflows.

However, knowledge processing in itself is not sufficient. Any information system requires *database* functionality, which deals with efficient data storage and retrieval. Of course, sometimes implementation-wise, this category will overlap with knowledge representation, but given the focus of our community, separating the two categories makes sense. Also, any collaborative system requires *workflow support* and *discussion* functionality as well as *user interface* functionality.

This functionality classification is not exhaustive. However, it is useful for decomposing an information system into its main elements, which can be combined in many different ways to describe the overall functionality of an information system.

Fig. 1 gives a functionality requirements decomposition of knowledge-intensive information systems for collaborative human communities. It shows how knowledge elicitation feeds into formal knowledge representation and analysis, which in turn is accessed by knowledge visualization. Discussion and workflow modules retrieve and return data from and to the database management module. All these modules interact with the user interface, which in turn is accessed by the community of (human) users. The database and knowledge representation and analysis modules also interact. We will use this particular requirements decomposition as the basis for the analysis of DOGMA-MESS later in this paper.

2.2 Functionality Components

The implementation of web-service based information systems consists of *components* of different granularity. At the lowest granularity level, we talk about *systems* of tools or services. The next level consists of the *tools* or *services* themselves. One level further are the *modules* out of which the tools or services are comprised. Finally, we distinguish the particular *functions* grouped in a module.

Functionality components of different levels of granularity can be *linked* in different ways. The *interface* between two functionality components A and B is described by the

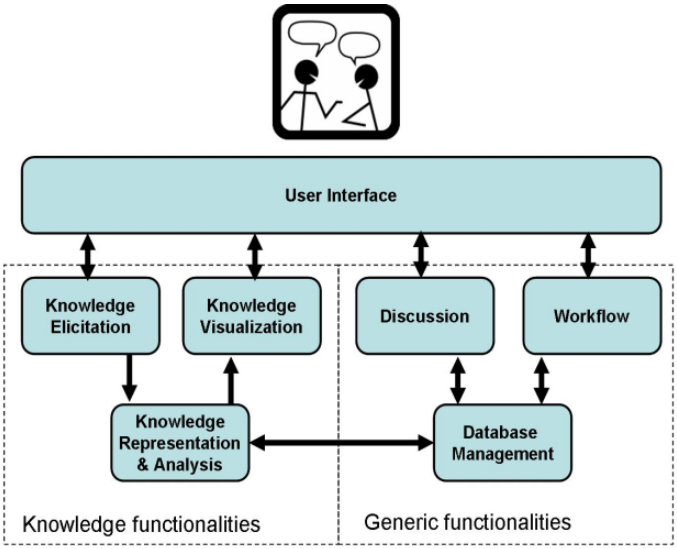

Fig. 1. A Functionality Requirements Decomposition of a Knowledge-Intensive IS for Collaborative Communities

externally visible representation of the knowledge structures used in A and B, as well as the possible communication processes between these two components. The inner process implementation of the functionality components is not relevant for interoperability analysis. This can be considered to be a black box as long as the external interfaces of knowledge structure representations are known to the other functionality components.

Fig. 1 shows a link between Discussion and Database Management. In Fig. 2 we give an example of a possible implementation of functionality component decomposition of an interoperable discussion storage process. Whereas Fig. 1 focused on the conceptual functionality *specification*, we now concentrate on the *implementation* interoperability aspects of this process. Assume that the implementation is of a Discussion Support System that allows people to conduct discussions, while also making the results of these discussions systematically accessible, searchable etc. The system consists of two main tools, a Discussion Forum and a Database Management System (DBMS). Each of these tools has many functionality modules, for example ”Manage Entries” in case of the Discussion Forum, and ”Store Data” in case of the DBMS. Each of these modules supports a range of functions.

The function ”Add Post” needs to send new posts to the Store Data-Module in the DBMS. However, here an interoperability problem occurs. It turns out that the Discussion Forum allows attachments in the posts. These cannot be stored as such in the Store Data-Module, because the DBMS only knows two data types: XML and binary files, which each are stored by a different function. One important interoperability problem therefore is how to design the splitting of posts into threads and attachments, and then converting threads into XML. Preferably, this function should be done by a dedicated export or import-function in one of the two tools. In practice, with today’s limited tool

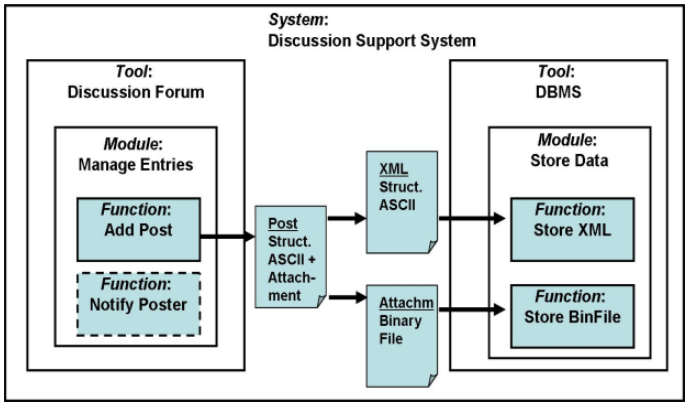

Fig. 2. A Functionality Component Decomposition of an Interoperable Discussion Storage Process

interoperability, special converters will need to be written. These should be incorporated in an update of the tool as much as possible, so other systems can benefit from the increased interoperability as well. Ideally, these converters also become part of a public library, so that at least these interoperability hurdles can be reduced.

2.3 Interoperability Aspects

Understanding which functionality requirements are concerned and by what components to implement them is a first step. When making the final implementation decisions, many interoperability aspects need to be taken into account as well. Some aspects that have guided our decisions are given in the table below. For each aspect, we give an informal definition and a typical example. Even though the list is informal and incomplete, we believe it could be useful for other developers who can compare them with their own views and evaluate their own interoperability problems against them.

Aspect	Definition	Example
Efficiency	The speed at which the tool performs the requested functionality.	How many projection operations per second can a conceptual graph tool perform?
Maturity	The degree of functionality completeness and lack of errors of the tool.	Is there a roadmap to implement lacking functionality? Are there (still) many bugs?
Compatibility with standards	The ability of the tool to use common standards for its functionality, import, and export.	Is the conceptual graph tool capable of importing from and exporting to CGIF?
Integration with legacy	The ability of the tool to interoperate with legacy systems.	The user's system was written for Linux and the tool runs only on Windows.
Usage complexity	The level of user pre-requisite knowledge and expertise to operate the tool.	The tool only allows expression of data in CGIF.

Aspect	Definition	Example
Deployment complexity	The amount of non-trivial in-house work required to interoperate with the tool.	The tool was written in C++ and one's system in Java. Interoperability is possible, but would be easier if the tool had a Java interface.
Testability	The ability to test the tool before incorporation.	A tool with a user interface can be tested immediately. A public library without UI requires programming.
Maintainability	The ease with which a tool can be updated.	New versions of the tool are not backwards compatible.
Contactability	The ease of contacting and influencing tool creators, developers, maintainers, and users.	Is there an active contact e-mail address? Is there a (bug) forum and/or mailinglist? Are suggestions taken into account?
Documentation	The amount and quality of information about the tool.	A manual for installation and/or use of the tool.
Extensibility	The possibility to extend the tool beyond primary requirements.	The tool is capable of handling third-party plugins.

3 Interoperability Analysis of DOGMA-MESS

We use the developed conceptual model to reflect upon our practical experiences with conceptual structures tool interoperability in developing the DOGMA-MESS system [2]. As explained before, we describe interoperability in terms of functionality requirements, functionality components, and interoperability aspects.

Our informal methodology consists of the following steps: (1) *functionality requirements decomposition* (see Fig. 1 for a high-level decomposition of DOGMA-MESS); (2) *tool selection* for high-level functionality types; (3) *low-level functionality component decomposition*. The first step conceptually specifies what functionality is needed, the second step produces a tool-level implementation design, which is refined and extended in the third step for actual implementation by defining interoperability aspects at the module and function level.

These steps are reiterated in a process of co-evolution of specification and implementation. Each iteration is a complex interaction between specification and implementation. Naturally, new specifications lead to new implementations (e.g. new selections and configurations of tool functionality components or even the creation of new tool components). However, we have also observed that updated implementations can also lead to revised specifications by allowing for new user behaviors. User-tests after a number of iterations proved to be very valuable in triggering and calibrating co-evolution processes.

In the remainder of section, we illustrate how DOGMA-MESS system specification (i.e. functionality requirements) and implementation (i.e. functionality components) co-evolved in a number of stages. In the next section, we examine in more detail the co-evolution of one particular stage: the knowledge representation and analysis functionality as implemented by Prolog+CG.

3.1 DOGMA-MESS: Co-Evolution of Specification and Implementation

The DOGMA (Designing Ontology-Grounded Methods and Applications) approach to ontology engineering, developed at VUB STARLab, aims to satisfy real-world knowledge needs by developing a useful and scalable ontology engineering approach [11]. Its main tool is DOGMA Studio, which offers a range of functionalities to represent and analyze basic units of linguistically grounded meaning, called lexons, ontological commitments, and contexts. DOGMA-MESS (DOGMA Meaning Evolution Support System) is a community layer on top of the DOGMA ontology engineering functionality. Its main goal is to support community-grounded, interorganizational ontology engineering. DOGMA-MESS distinguishes three different user roles. A *Knowledge Engineer* takes care of the system and performs in-depth semantical analysis. A *Core Domain Expert* creates a common concept type hierarchy and templates (in the form of conceptual graphs), while *Domain Experts* each build their organizational specializations of the templates. More details can be found in [2]. Here we focus on functionality requirements and component co-evolution that resulted in the current DOGMA-MESS.

As a system, DOGMA-MESS combines the specialized functionality of many different tools, presenting different functionalities to different user roles. Fig. 3 shows the selected tools for the high-level functionality requirements decomposition of DOGMA-MESS (Fig. 1) after a number of iterations.

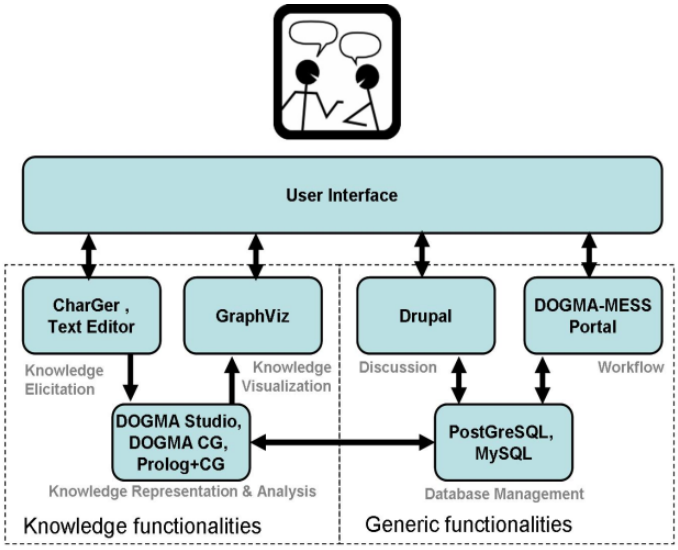

Fig. 3. High-Level Tool Selection in DOGMA-MESS

In the development of DOGMA-MESS so far, we have observed four stages, which we name after the key functionality requirement being implemented in that stage: (1) knowledge elicitation, (2) knowledge representation and analysis, (3) knowledge visualization, and (4) knowledge interpretation.

3.2 Knowledge elicitation

Before anything else, knowledge has to be written down and collected. Our first iteration therefore concerned knowledge elicitation. We chose CharGer[1] as our elicitation tool for the DOGMA-MESS templates. CharGer is a conceptual graph editor which allows a user to draw conceptual graph(s) and associated type hierarchies. As such, it is a visual and potentially efficient way to elicit knowledge. Of CharGer's different modules (user interface, output to XML, input from XML, some matching operations, ...) we only used the user interface and the output to XML. The specific XML format allows for relatively easy data exchange and conversion into our simple native DOGMA-CG format (which is amongst other things limited to simple referents). It also stores positioning info for graphs. However, for graphs *generated* elsewhere (e.g. as a result of some DOGMA-MESS analysis operation) this information would not be present and as such, CharGer would not be able to display them properly.

During initial concept type elicitation sessions it became clear that the usage complexity was too large when the type hierarchy grew large (Fig. 4a). Strongly urged by our test-users, we switched to a different approach in which a type hierarchy is elicited using a plain text editor and a simple indented format for subtypes (Fig. 4b)

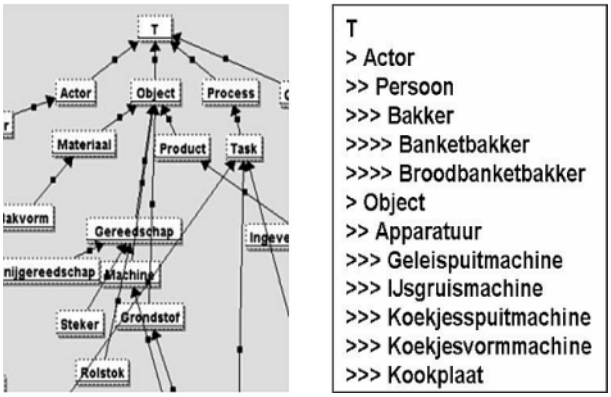

Fig. 4. Concept Type Hierarchy Elicitation Re-Implementation

Similar graph and type elicitation functionality is available on our portal, so the user can elicit knowledge without using any external tool. However we do keep CharGer and the text editor as input tools, since for expert users they can be more efficient. The usage complexity is higher however, as CharGer has to be downloaded, Java installed, and a new tool learnt. In line with the idea mentioned before that interoperability needs to take into account user culture, CharGer is available to the Core Domain Expert for initial high-volume template generation, whereas Domain Experts only use the simpler to use, but reduced functionality graph editing functions available on the portal for creating their organizational specializations of the templates.

[1] http://sourceforge.net/projects/charger/

3.3 Knowledge representation and analysis

In the second iteration, we implemented the core functionality to represent and reason about the collected knowledge, not at the external tool level, but at the DOGMA-MESS system-level. In DOGMA-MESS, there are two subsystems for knowledge representation and analysis. One concerns user-defined templates, specializations and type hierarchies (represented as conceptual graphs). As the main tool for conceptual graph analysis, we opted for Prolog+CG. The other subsystem concerns ontological analysis across contexts, for which we used and extended DOGMA Studio's handling of lexons and commitments, the core DOGMA knowledge structures. Besides providing the ontological analysis, DOGMA Studio delivers persistency functionality to DOGMA-MESS as well. It uses a PostgreSQL database to store templates, specializations, lexons, commitments and other data used by DOGMA and DOGMA-MESS. At this point, DOGMA is a central knowledge representation into which we convert the elicited conceptual graphs and type hierarchies. In future iterations, we will look for other representations that can convert into DOGMA and back again. In order to incorporate conceptual graphs into DOGMA-MESS we added DOGMA-CG, a module to handle (a subset of) conceptual graph theory inside DOGMA Studio. As we did not intend to build core CG-operations ourselves, this module mainly consists of the minimum required functionality to enable interfacing (open data model, conversion from CharGer format, high-level abstraction interfaces with Prolog+CG, etc.).

At the end of this iteration we had a system that could store and analyze various types of elicited and generated knowledge, perform analysis, and convert knowledge between the different functionality components. Next, we needed functionality to deliver the results (e.g. from a projection operation) back to the user, so the focus shifted to knowledge visualization.

3.4 Knowledge visualization

After and during analysis the user needs to be able to examine newly generated knowledge. As we were dealing with system-generated conceptual graphs, we required a tool that was capable of providing automatic layouts for graphs. We chose AT&Ts GraphViz [2], a tool that provides several algorithms with numerous configuration parameters for graph layouting. It was not trivial to implement this as it runs as an independent system process and cannot be called in another way. However, GraphViz works with a language called *dot*. The language is easy to learn and provides many options to visualize graphs. Moreover, GraphViz reads dot files that describe a graph and output it to a picture. We therefore used dot-files as an intermediate representation between different DOGMA-MESS functionality modules.

In DOGMA-MESS, we require three different visualizations: for templates, specializations of these templates and concept type hierarchies. Templates are displayed as simple graphs. Inspired by the CharGer layout, we use a different color and shape to indicate the difference between concepts and relations (Fig. 5a). Specialisations of templates are pictured in the same way as their templates, with one difference: we cluster

[2] http://www.graphviz.org/

the specialised concepts and we label these clusters with the type label of the matching concept in the template (Fig. 5b). In tests with end-users we found that this extra visual aid made the editing process much easier, as a user now always knows what category of concepts are collected in a box, e.g. the '*Product* Bread'.

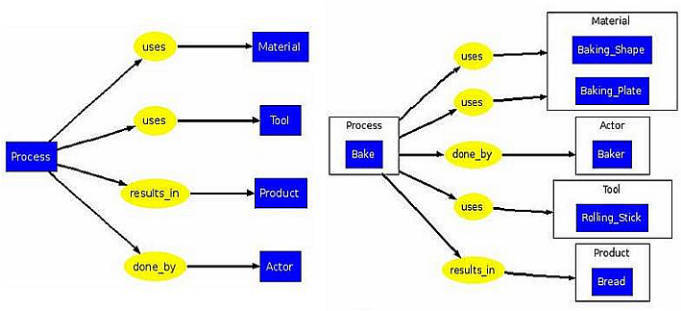

Fig. 5. Template and Specialization Visualization

In an initial test, we displayed type hierarchies in a similar way: as a large graph oriented from top to bottom. We found that this was feasible as long as we were experimenting only with our own small test type hierarchies. As soon as we started working with larger bases of user-provided material, however, we discovered that a realistic type hierarchy is too large to be displayed properly (including keeping a good overview) in this manner. For this reason we adopted the indented format used in knowledge elicitation and decided to use this for visualization as well.

Although GraphViz satisfied most of our visualization needs, there are still some unsolved visualization issues (e.g. the ordering of concepts is random and different between template and specialization, see Fig. 5). In the future, we will therefore also consider other visualization tools such as Java-based Jung[3].

At the ontological level, DOGMA Studio is currently extended to provide visualization in the form of so-called NORM trees, which are sequences of connected relations. A NORM tree is created by starting your browsing in a node of the ontology and traversing over the relations to neighbouring nodes.

3.5 Knowledge Interpretation

The (current) fourth iteration is dedicated to tying all components together into a portal and enabling workflow support across components. Using this portal, the users are guided through the actions they should or can perform, in this way orchestrating the collaboration between the users. The portal also provides support for informal yet focused discussion of the emerging knowledge definitions using the Drupal platform (a community content management tool with a large variety of plugin modules[4]. Other

[3] http://jung.sourceforge.net/
[4] http://drupal.org/

functionality besides discussion functionality can be added when there is need for it (eg. voting, file management, ...). At this point Drupal and the rest of DOGMA-MESS are not tightly coupled yet (other than in look and feel), but this is definitely to be done in a future iteration.

Having presented a bird's eye view of the co-evolution of specification and implementation of DOGMA-MESS, we now zoom in on one stage: the implementation of knowledge representation and analysis through Prolog+CG. This should give the reader a better idea of the third stage of our methodology, the low-level functionality component decomposition.

4 Zooming In: Integrating Prolog+CG

In order to improve our system with the reasoning power of conceptual graphs, we had the choice of two solutions; build that part ourselves or interface with an existing tool/API. Building functionality like that requires in-depth knowledge of the theory and a serious amount of time and development resources. As we lacked in both these requirements, we needed to find a third party implementation that covered most of our needs. Basing ourselves on our list of interoperability aspects described in Sect. 2.3, we came up with the following requirements. We required a tool that: (1) has an easy-to-use Java interface, (2) is mature enough, (3) is still being upgraded or maintained, (4) has the necessary operations (projection and maximal join), (5) comes with good documentation and support, and (6) has capabilities for extending reasoning functions. In the ideal case, this tool would act as much like a black box as possible, by having powerful operations with clearly described interfaces. This way we can use all the functionality without needing to know (too much of) the internal workings.

A good overview of CG tools is given in [5], which compares the tools on over 160 criteria. Using this comparison in combination with our criteria, we initially opted for Amine (a platform that holds the latest version of Prolog+CG). We started using Amine, but we soon found that this was too complex for our intended black box use. As Amine is not just a CG tool, but a complete AI platform, it is more complex to integrate and use it. Since we only needed its Prolog+CG capabilities, we therefore considered Prolog+CG itself. Even though Prolog+CG was only maintained and not further developed, and because the maintainer of the tool, Ulrik Petersen, was still willing to do bug fixes and minor functionality upgrades, we decided to use the latest available version of Prolog+CG instead.

Following good software practices, we kept the influence of this legacy choice as small as possible so that we have minimal refactoring work if we ever decide to use another CG tool with better functionality. This means that other modules that need CG functionality interact only with DOGMA-CG inside our system, and that DOGMA-CG is the only module that interfaces with Prolog+CG directly. In this case, only DOGMA CG needs to be reworked/updated if we were to use another CG tool in the future.

Prolog+CG is a tool composed of several modules (User interface, Prolog interpreter, File storage, CG operations, ...). We only needed interfacing with two of them;

[5] http://en.wikipedia.org/wiki/CG_tools

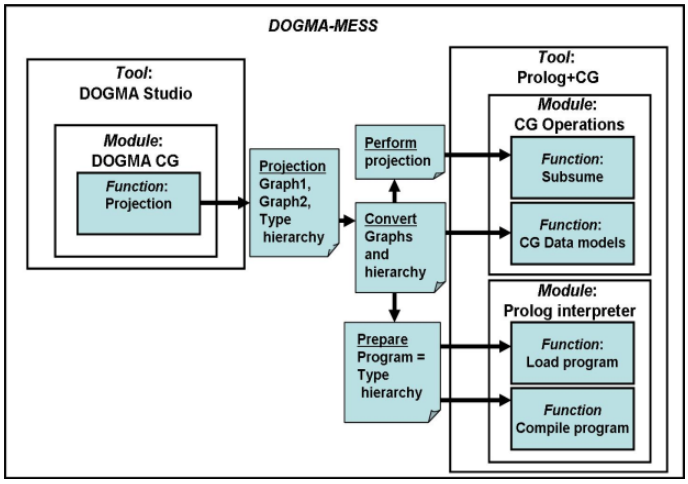

Fig. 6. A Low-Level Functionality Component Decomposition of a Projection Operation

the Prolog interpreter (to prepare the system for operations) and the CG operations. More specifically, we needed two CG operations from Prolog+CG: projection and maximal join. Fig. 6 explains the interface between DOGMA CG and Prolog+CG for the projection operation. This process consists of three steps; (1) the graphs and the type hierarchy are converted into Prolog+CG data models, (2) the type hierarchy is loaded and compiled as a Prolog+CG program and (3) the subsume operation is called. The result is then converted to DOGMA-CG data models again by our system. The maximal join operation process behaves in a similar way.

When a user of DOGMA-MESS creates a specialization, the system has to check if the graph she created is really a specialization of the template or, in other words, the template has to subsume the specialization, ie. do a projection. In DOGMA-MESS, the user can also create so-called index graphs (graphs that describe learning materials) and store them in the system. With a query graph she can then look for other matching index graphs, which are also found by performing a projection of the query graph on all stored index graphs. When the user creates the index graph, the DOGMA-MESS system will automatically look for matches (of the index being created) and should perform a maximal join on all matches to return one large index of related indices that the user can compare to his index. This way the system assists the user actively in creating his index in several steps.

During the integration of Prolog+CG into DOGMA-MESS, we encountered many difficulties, but two of them turned out to be major problems. We will explain them using example data (which is a simplified translated version of actual user data from the Dutch bakery case described in [2]).

The first problem concerned the projection operation (see Fig. 7). We expected that a projection of graph 2 onto graph 1 would be possible (as graph 2 subsumes graph 1). However Prolog+CG returned no result, so we either misinterpreted the theory or

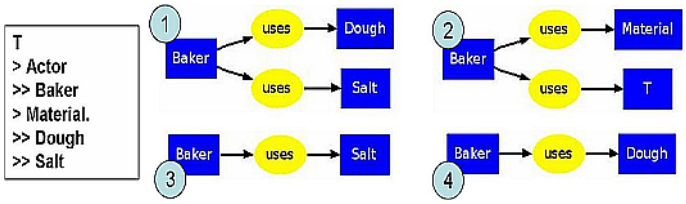

Fig. 7. Problems with CG operation implementations

Prolog+CG had a bug. The maintainer confirmed that it was indeed a bug and fixed it. With this updated version of Prolog+CG, our first problem was resolved.

The second problem was more difficult to solve and involved the maximal join operation. Our expectation was that a maxJoin of graph 3 and graph 4 would result in graph 1. However, once again we did not get a result from Prolog+CG. This time, our interpretation of the definition of the maximal join operation turned out to be slightly different than the one used in standard theory. According to [10] our example would only work if Salt and Dough had a maximum common subtype which is greater than absurd (e.g. 'SaltyDough'). And even then, the result would only be [Baker]→ uses → [SaltyDough]. Although our expected behaviour is very natural for human users, and thus practically required for DOGMA-MESS (and most probably for other systems as well), it seemed not possible. We were forced to look for solutions and found many, including new problems. We could (1) force the users to use different relations (like UsesByProduct and UsesHerb) but this is not natural, (2) automatically create such relations, but that just moves the problem to finding a way to do that (which is equally difficult), (3) force the users to adapt their type hierarchy (eg. add a Material subtype SaltyDough, with Salt and Dough as subtypes), which is again not natural, (4) implement an adapted maximal join ourselves, but that requires in-depth knowledge and extensive time and testing and lastly (5) look for a way to do it without maximal join. Since the first four solutions came with a heavy drawback, we opted for the last one.

We needed maximal join to present the user with a system-generated index when he is creating an index himself. In our original specification, the system would look for matching projections of the index-in-progress and join them all into one large index, to show related relations to the index being edited by the user. As the maximal join failed our requirements, we could not use this approach. Instead we just displayed the matching graphs to the user in sequence. Whereas we originally saw this as a drawback, we now believe this ad hoc solution to actually be a feature, since it allows authors of index graphs to see the matching graphs separately, thus making it much clearer what the original elicitation context of each graph is than would have been possible if all those graphs had been merged into a single large one. This, thus, is a good example of implementation driving specification.

The incorporation of Prolog+CG into DOGMA-MESS was definitely not an easy one. Part of the difficulty originated from the fact that Prolog+CG is legacy code, and mainly designed as a stand-alone tool. Another part was caused by our specific needs which did not completely match the theory. Several code upgrade iterations were spawned by this process, facilitated by the good contact between the first author and

the Prolog+CG maintainer. This ultimately resulted in a major speed increase (about tenfold both due to the new version of the tool and the more efficient way we could interact with the tool) and an improved version of Prolog+CG. The human factor in interoperability is definitely an important one, not only relating to user culture, but also pertaining to social requirements of and interactions between other roles in the software process, such as system and tool developers and maintainers.

5 Conclusions

In this paper, we told a tale from the trenches of tool-based information systems development. This is a long-term, co-evolution process of requirements and tools, in which many interoperability issues, from the very technical to the very social ones play their intricate roles. We provided a simple conceptual model of tool interoperability, using functionality requirements, components, and interoperability aspects as its main vocabulary. We used our model to describe the first stages in the tool-based development of the DOGMA-MESS system for interorganizational ontology engineering. It is a system that ties together several tools, modules, and functions, aiming to balancing the particular strengths of these functionality components while minimizing their weaknesses.

What have we learnt from our reflection on DOGMA-MESS using our informal methodology? First, we found that it is very important to carefully select the tools which to incorporate in one's system, since tool selection decisions have far reaching consequences. The preference of one tool over another is something that must be decided based on a careful analysis of as many of the interoperability aspects as possible. After having examined the relevant characteristics of the available tools, one must rank the aspects according to one's own priorities and preferences before making a decision. To optimize this difficult process, much research is still needed on typologies, operationalizations, and measurement procedures of interoperability aspects. General quality approaches such as [7] are useful starting points, but not specific enough to provide sufficient guidance for tool-based system implementation. Once everything is properly ranked and a tool chosen, one should stick with it for as long as possible. Only when the tool results in critical problems for the overall system, should a replacement tool be considered. Dependencies between system components should be minimized. Next to good documentation, a support forum and mailing-list, personal (electronic or otherwise) contacts with tool developers or maintainers is essential in order to at least address bug fixes and minor functionality upgrades. Without such opportunities for tool evolution, system evolution is jeopardized.

Summarizing, tool-based systems development comes with many interoperability issues. Many, more advanced frameworks for examining interoperability issues exist, for example focusing on different layers of interoperability or semantic interoperability [9, 13]. However, these approaches focus especially on formal and technical aspects, not on charting many aspects of the evolutionary, socio-technical process of tool-based systems development in practice. Of course, our light-weight method for tool-based systems analysis and design is only in its infancy, but has already helped us make more systematic sense of some of these aspects, and guide many of our design decisions.

References

1. F. Baader and W. Nutt. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. A. de Moor, P. De Leenheer, and R.A. Meersman. DOGMA-MESS: A meaning evolution support system for interorganizational ontology engineering. In *Proc. of the 14th International Conference on Conceptual Structures, (ICCS 2006), Aalborg, Denmark*, Lecture Notes in Computer Science. Springer-Verlag, 2006.
3. P. Fremantle, S. Weerawarana, and R. Khalaf. Enterprise services. *Communications of the ACM*, 45(10):77–82, 2002.
4. B. Ganter and R. Wille. *Applied Lattice Theory: Formal Concept Analysis*. Technical Univ. of Dresden, Germany, 1997.
5. Wilhelm Hasselbring. Information system integration. *Communications of the ACM*, 43(6):32–38, 2000.
6. A.V. Pietarinen. Peirce's theory of communication and its contemporary relevance. In K. Nyiri, editor, *Mobile Learning: Essays of Philosophy, Psychology and Education*, pages 46–66. Passagen Verlag, Vienna, 2003.
7. L.L. Pipino, Y.W. Lee, and R.Y. Wang. Data quality assessment. *Communications of the ACM*, 45(4), 2002.
8. S. Sawyer. A market-based perspective on information systems development. *Communications of the ACM*, 44(11):97–102, 2001.
9. A.P. Sheth. Changing focus on interoperability in information systems: From system, syntax, structure to semantics. In M.F. Goodchild, M.J. Egenhofer, R. Fegeas, and C.A. Kottman, editors, *Interoperating Geographic Information Systems*, pages 5–30. Kluwer, 1998.
10. John F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, Massachusetts, 1984.
11. P. Spyns, R. Meersman, and M. Jarrar. Data modelling versus ontology engineering. *SIGMOD Record*, 31(4):12–17, 1998.
12. M. Stonebraker. Integrating islands of information. *EAI Journal*, Sept 1999, http://www.eaijournal.com/DataIntegration/IntegrateIsland.asp.
13. G. Vetere and M. Lenzerini. Models for semantic interoperability in service-oriented architectures. *IBM Systems Journal*, 44(4):887–903, 2005.