A Knowledgebased Approach to Merging Information

Anthony Hunter and Rupert Summerton
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK

December 13, 2005

Abstract

There is an increasing need for technology for merging semi-structured information (such as structured reports) from heterogeneous sources. For this, we advocate a knowledgebased approach when the information to be merged incorporates diverse, and potentially complex, conflicts (inconsistencies). In this paper, we contrast the goals of knowledgebased merging with other technologies such as semantic web technologies, information mediators, and database integration systems. We then explain how a system for knowledgebased merging can be constructed for a given application. To support the use of a knowledgebase, we use fusion rules to manage the semi-structured information that is input for merging. Fusion rules are a form of scripting language that defines how structured reports should be merged. The antecedent of a fusion rule is a call to investigate the information in the structured reports and the background knowledge, and the consequent of a fusion rule is a formula specifying an action to be undertaken to form a merged report. Fusion rules are not necessarily a definitive specification of how the input can be merged. They can be used by the user to explore different ways that the input can be merged. However, if the user has sufficient confidence in the output from a set of fusion rules, they can be regarded as a definitive specification for merging, and furthermore, they can then be treated as a form of meta-knowledge that gives the provenance of the merged reports. The integrated usage of fusion rules with a knowledgebase offers a practical and valuable technology for merging conflicting information.

1 Introduction

There is an increasing need to develop semi-automated and automated techniques for merging information obtained from heterogeneous sources. A number of approaches have been proposed, but none address all application problems. Amongst the wide range of proposals, it is surprising that knowledgebased techniques have not been harnessed to their full potential. In this paper, we respond to this situation by advocating the use of knowledgebased merging. We start by introducing the types of problem that we are interested in addressing, and then follow this with a comparative review of other approaches to merging information, in particular semantic web technologies, information mediators, and database integration systems. We follow this with an exposition of our approach to knowledgebased merging. For this, we will argue that fusion rules are an ideal way of controlling knowledgebased merging.

For our discussions, we are particularly interested in merging semi-structured information such as structured reports. These reports are in the form of XML documents, where the textentries are restricted to individual words or simple phrases (such as names and domain-specific terminology), dates, numbers and units. Types of report that can be put into structured reports include news reports, weather reports, clinical

records, and records on methods/results for scientific experiments. Structured reports do not require natural language processing, though they may be obtained as output from information extraction technology. Many structured reports can also be obtained from relational and semi-structured data.

In order to merge heterogeneous structured reports intelligently, we need to take account of the contents of each report. Different kinds of content, and different types of conflict (whether qualitative or quantitative) need to be merged in different ways. In many cases, the kind of merging required to exploit the content available in structured reports goes far beyond the mere combination of information envisaged by some approaches to information integration. This is because a source may provide information that *conflicts* with, rather than *complements* that provided by another source; that is, instead of providing additional information about different events or phenomena (as when one source describes what is showing at one cinema in the neighbourhood, whilst another says what is showing at a different cinema), a source may provide different information about the same events or phenomena (as when one weather report says it will rain tomorrow, whilst another predicts sunshine). Conflicting information, especially when it is qualitative, requires the use of logical reasoning, with axioms and rules contained in a knowledgebase, in order to resolve the conflict and merge the information. In general, the sources of conflicts fall into one or another of the following categories.

Dispositional conflicts These are conflicts that arise because there is a difference in values or interests involved on the part of those compiling the reports, as for example when there is a discrepancy in the casualty figures as reported by different sides in a war or conflict, or when one source describes a group of people as terrorists whilst another describes them as resistance fighters, etc. Political reporting, especially during election campaigns, provides another obvious example of conflicting descriptions of the same events. Some dispositional conflicts arise through less conscious systematic bias in reporting information. For example, some sources of weather tend to be optimistic and describe the weather prediction as sunny even if there is a substantial amount of cloud that will obscure the sun for protracted periods. In contrast, other sources would tend to be pessimistic, and highlight rain in a forecast even if the probability of rain is actually quite low. Meta-level knowledge can be harnessed to ameliorate these kinds of conflicts.

Epistemic conflicts These are conflicts that arise for epistemic reasons: Different sources have different beliefs. This can arise because the event or phenomenon being described has not yet happened and so the conflicting reports are *forecasts*. Obvious examples here include weather reports, horse racing tips, and predictions of the stock market or of individual stocks. Consider two sources of weather information that have different instruments for sensing the weather conditions. These two sources are unlikely to always have identical beliefs about the weather. Equally important is where the nature or existence of the events in question is in dispute because they occurred in the past and the sources of information have an incomplete and not totally correct understanding of the past events being described. An obvious example is reports written by detectives during the course of a major inquiry. Meta-level and/or uncertainty knowledge can be harnessed to ameliorate these kinds of conflicts.

Ontological conflicts These are conflicts that arise because of differences in terminology and in uses of terminology: Different terms can be used by different sources for the same concept and/or the same term can be used by different sources for different concepts. Knowledge about ontological relationships [Cru86], including synonymy, polysemy, homonymy, antonymy, meronymy, and hyponymy, can be harnessed to ameliorate these kinds of conflicts.

Often specific applications will involve conflicts which fall into multiple categories further reinforcing the argument that we need to reason with the information provided by sources to determine the kind of conflicts arising, and then to find an appropriate and intelligent resolution.

In all of these cases something more than mere conjunction is required if information from different sources is to be integrated. In this respect, the project of integration is different from what is required in cases such as: find out what films are on in Birmingham city centre; what Chinese restaurants are there in West

London?; what are the economy flights from London to Athens in July?, etc. In these cases what we want is a list, with duplicates removed, obtained by conjoining all the information sources.

By contrast, in cases where the sources conflict much more is involved if the information is to be integrated. Even a simple approach to the case of horse racing would require, for example, a knowledgebase containing preferences over sources, or various voting functions (majority, first past the post, threshold, etc.). But obviously much more sophisticated knowledgebases could be constructed, taking into account the going (i.e. the condition of the track), the horses' previous results, handicapping information, and so on. Preferences over sources could also be used to settle differences between opinion polls concerning future election results (prefer Gallup to Mori, etc.); or an average amongst the polls would be another way in which they could be merged (see Example 1.1); or simply some form of voting function.

Example 1.1 *Consider the following two conflicting (and imaginary) reports concerning opinion polls for the political parties in the UK. The clients are national newspapers.*

```
⟨opinionpoll⟩
        ⟨source⟩ Mori ⟨/source⟩
        ⟨client⟩ Guardian ⟨/client⟩
        ⟨date⟩ 19/3/04 ⟨/date⟩
        ⟨parties⟩
                ⟨party⟩ Labour ⟨/party⟩
                ⟨poll⟩ 41% ⟨/poll⟩
                ⟨party⟩ Conservative ⟨/party⟩
                ⟨poll⟩ 39% ⟨/poll⟩
                ⟨party⟩ Liberal Democrat ⟨/party⟩
                ⟨poll⟩ 21% ⟨/poll⟩
        ⟨/parties⟩
⟨/opinionpoll⟩
```

```
⟨opinionpoll⟩
        ⟨source⟩ Gallup ⟨/source⟩
        ⟨client⟩ Daily Telegraph ⟨/client⟩
        ⟨date⟩ 20 March 2004 ⟨/date⟩
        ⟨parties⟩
                ⟨party⟩ Lab ⟨/party⟩
                ⟨poll⟩ 38% ⟨/poll⟩
                ⟨party⟩ Con ⟨/party⟩
                ⟨poll⟩ 42% ⟨/poll⟩
                ⟨party⟩ Lib Dem ⟨/party⟩
                ⟨poll⟩ 19% ⟨/poll⟩
        ⟨/parties⟩
⟨/opinionpoll⟩
```

Using a suitable knowledgebase, dealing with alternative party names and date formats, we can produce the following merged report. In this case the entry for each `poll` *result is determined by finding the mean of the* `poll` *results in the two reports.*

```
⟨pollofpolls⟩
        ⟨sources⟩ Mori and Gallup ⟨/sources⟩
        ⟨clients⟩ Guardian and Daily Telegraph ⟨/clients⟩
        ⟨date⟩ 19/3/04 ⟨/date⟩
        ⟨parties⟩
                ⟨party⟩ Labour ⟨/party⟩
                ⟨poll⟩ 39.5% ⟨/poll⟩
                ⟨party⟩ Conservative ⟨/party⟩
                ⟨poll⟩ 40.5% ⟨/poll⟩
                ⟨party⟩ Liberal Democrat ⟨/party⟩
                ⟨poll⟩ 20% ⟨/poll⟩
        ⟨/parties⟩
        ⟨summary⟩ Conservative lead of 1% ⟨/summary⟩
⟨/pollofpolls⟩
```

Voting and preferences over sources could also be used in the case of stock tips, though a further refinement that could be used in this case would be to index the preferences by industry sector since certain forecasters might be thought to have a better track record in, say, the energy sector, whilst others are to be preferred in financial services, etc. (see Example 1.2). And of course all sorts of other information could be incorporated into a knowledgebase about the track records of the stocks, discounting of tips from banks or investment houses that hold large quantities of the stocks, etc.

Example 1.2 *Consider the following four conflicting (and imaginary) reports concerning stock in the Royal Bank of Scotland.*

```
⟨stockreport⟩
        ⟨stock⟩ RBS ⟨/stock⟩
        ⟨sector⟩ financial services ⟨/sector⟩
        ⟨date⟩ 19/3/04 ⟨/date⟩
        ⟨source⟩ Financial Times ⟨/source⟩
        ⟨price⟩ 209.5 ⟨/price⟩
        ⟨recommendation⟩ buy ⟨/recommendation⟩
⟨/stockreport⟩
```

```
⟨stockreport⟩
        ⟨stock⟩ Royal Bank of Scotland ⟨/stock⟩
        ⟨sector⟩ finance ⟨/sector⟩
        ⟨date⟩ 21 March 2004 ⟨/date⟩
        ⟨source⟩ HSBC ⟨/source⟩
        ⟨price⟩ 209 ⟨/price⟩
        ⟨recommendation⟩ sell ⟨/recommendation⟩
⟨/stockreport⟩
```

```
⟨stockreport⟩
        ⟨stock⟩ RBS ⟨/stock⟩
        ⟨sector⟩ financial services ⟨/sector⟩
        ⟨date⟩ 20 − 03 − 04 ⟨/date⟩
        ⟨source⟩ Waterhouse ⟨/source⟩
        ⟨price⟩ 209.5 ⟨/price⟩
        ⟨recommendation⟩ buy ⟨/recommendation⟩
⟨/stockreport⟩
```

```
⟨stockreport⟩
        ⟨stock⟩ RBS ⟨/stock⟩
        ⟨sector⟩ banking ⟨/sector⟩
        ⟨date⟩ 18th March 2004 ⟨/date⟩
        ⟨source⟩ Morgan Stanley ⟨/source⟩
        ⟨price⟩ 209.25 ⟨/price⟩
        ⟨recommendation⟩ buy ⟨/recommendation⟩
⟨/stockreport⟩
```

Using a suitable knowledgebase we can produce the following merged report, allowing for alternative company names, date formats, and sector descriptions. In this case the entry for `overallrecommendation` *is determined by a preference for HSBC over the others as a source, and so is* `sell`*.*

```
⟨stockreport⟩
        ⟨stock⟩Royal Bank of Scotland ⟨/stock⟩
        ⟨sector⟩ financialservices ⟨/sector⟩
        ⟨date⟩ 18/3/04 − 21/3/04 ⟨/date⟩
        ⟨price⟩ 209 − 209.5 ⟨/price⟩
        ⟨sources⟩
                ⟨source⟩ Financial Times ⟨/source⟩
                ⟨recommendation⟩ buy ⟨/recommendation⟩
                ⟨source⟩ HSBC⟨/source⟩
                ⟨recommendation⟩ sell ⟨/recommendation⟩
                ⟨source⟩ Waterhouse ⟨/source⟩
                ⟨recommendation⟩ buy ⟨/recommendation⟩
                ⟨source⟩ Morgan Stanley ⟨/source⟩
                ⟨recommendation⟩ buy ⟨/recommendation⟩
        ⟨/sources⟩
        ⟨overallrecommendation⟩ sell ⟨/overallrecommendation⟩
⟨/stockreport⟩
```

An alternative way of merging these reports, with the entry for `overallrecommendation` *being* `buy` *instead, would be a simple first past the post, or a majority, voting function.*

Consideration of examples such as these shows, then, that in cases where there is conflict, the satisfactory integration or merging of information requires not just the gathering or bringing together of information from different sources (admittedly, something that is in itself a significant challenge), but the production of a distinct and novel source of information that resolves the conflict. We will argue that to produce such a report requires additional background knowledge and reasoning abilities, both things that can be provided by a knowledgebase. This is the basis of our case for knowledgebased merging.

We now turn to our approach to knowledgebase merging as an illustration of how knowledgebase merging can be undertaken. Essentially, in our approach, a range of predicates in a logical knowledgebase capture useful relationships between pieces of information in structured reports, and so a set of structured reports can then be analysed or merged as queries to a knowledgebase. In this way, merging some structured report can be handled by calls to the knowledgebase to merge the subtrees or textentries in the structured reports. This gives a context-dependent logic-based approach to merging that is sensitive to the information in the structured reports and to the background knowledge in the knowledgebase.

To support the use of a knowledgebase, we use fusion rules to manage the semi-structured information that is input for merging. Fusion rules are a form of logic-based scripting language that defines how structured reports should be merged. The antecedent of a fusion rule is a conjunction of conditions, where each condition is ground with information from the input structured reports and then handled as a query to the knowledgebase, and the consequent of a fusion rule is a formula specifying an action to be undertaken to form a merged report. So if all the conditions in the antecedent of a fusion rule succeed as queries to the knowledgebase, then the action in the consequent is executed.

The key aim of this paper is to show how formalisms for knowledgebased systems (logic-based knowledge representation and reasoning formalisms) can provide the core of viable and valuable merging of heterogeneous and conflicting information. The subsidiary aim of this paper is to explain how our approach to knowledgebased merging, with associated fusion rules, is a good and useful example of the general approach to knowledgebased merging[1].

In other papers, we have (1) presented an outline of using for fusion rules for knowledgebased merging of structured reports [Hun02a]; (2) presented a range of aggregation functions for use in fusion rules [HS04a]; (3) presented a framework for using temporal logic in knowledgebased merging [Hun02c, HS05]; (4) explored properties of a restricted form of fusion rules [HS03b]; (5) developed a framework for measuring degree and significance of inconsistencies in information in order to choose how to act on inconsistency with actions including ignore, resolve and reject [Hun03, Hun05]; and (6) developed aggregation predicates for merging uncertain information [HL05a, HL05b, HL05c, HL05d].

This paper extends the previous papers by providing a more general framework for knowledgebase merging and by providing comprehensive experiential insights into practical aspects of developing knowledgebased merging using fusion rules. In Section 2, we review the background to merging information, and argue how knowledgebased merging goes beyond other approaches to merging. Then in Section 3, we present our approach to knowledgebased merging describing the nature of a knowledgebase for merging together with associated fusion rules.

2 Background to merging information

There are of course many technologies that attempt to deal with merging or integrating information. But terms such as "merging", "fusion", and "integration" can be used in a variety of ways. It is useful, therefore, briefly to survey some of these existing approaches to merging or integrating information in order to see how they compare with our approach. It will be our contention that they are all in fact aiming to do something rather different from what we wish to accomplish.

2.1 The semantic web and associated technologies

Chief amongst technologies whose aim could be described as integrating information are those associated with the semantic web (see [BL99, BLHL01]), such as the Resource Description Framework (RDF) and RDF schema (RDFS), the Ontology Inference Layer (OIL, also used as an acronym for "Ontology Interchange Language"), and the DARPA Agent Mark-up Language (DAML) (see [FHvH$^+$00, FvHH$^+$02, Fen00]). The combination of these technologies is intended to make the information on the web not just machine *readable*, as at present, but machine *understandable*, thus increasing the "semantic interoperability" of the web. If this is to be done, what we need is semantic metadata — data that tells machines what the

[1] Further information on our approach to knowledgebased merging, including details on software prototypes, case studies, examples of knowledgebases, examples of sets of fusion rules, examples of structured reports used as input for merging, and examples of structured news reports obtained as output from merging, are given at www.cs.ucl.ac.uk/staff/a.hunter/frt.

content on the web means — rather than metadata that simply tells machines how to display web content (HTML), or what syntactic structure it has (XML).

The consensus is that such semantic metadata is best provided via ontologies, which offer a source of shared and defined terms standing in clear relationships to one another. These ontologies potentially provide the vocabulary to mark up or annotate the semantics of information on the web. DAML+OIL is an ontology representation language allowing for the specification and exchange of ontologies that is designed to provide this intelligent access for machines to the heterogeneous and distributed information found on the web. In particular, three features suit it to this purpose: (1) The use of a frame or class based modelling framework makes it easier for humans to construct ontologies; (2) The choice of XML and RDF web languages as a syntax makes DAML+OIL interoperable with existing web software; (3) The decision to design the language so that its semantics can be mapped to a description logic allows for reasoning in the construction and maintenance of ontologies, something that is very useful when checking the consistency of large ontologies.

Given this account of the purpose of these technologies, they can indeed be said to integrate information in the sense that they enable machines to determine the content or meaning of that information, and thus determine, for example, whether two sources contain information about the same thing. But it is important to note that they do this by acting on or exploiting this new semantic metadata; they do not concern themselves with the data — the actual web content — itself. The significance of this is that although they can be said to integrate diverse, heterogeneous data, perhaps it would be more accurate to say that they put us in a position of being able to integrate such data. For, knowing what information on a web site is about is one thing, knowing what to do with it is another. And as the developers of DAML+OIL have acknowledged, "Defining languages for the semantic web is just the first step. Developing new tools, architectures, and applications is the real challenge that will follow." (see [Fen00], p67.)

With respect to our interest in merging information, the relevance of these technologies is that they enable us to establish that the information to be merged is (or is not) about the same subject matter. But having done that, in themselves they say nothing about what to do with that data itself — what we should do if it does not conflict, what we should do if it does. Thanks to the semantic web, software agents will be able to tell us that one website on Claret contains information about the same thing as another website containing information on the wines of Bordeaux. But suppose these sites also contain differing opinions on the merits of the 2003 vintage. How should we deal with this conflict? In themselves, the languages developed to enable the semantic web say nothing about what to do here, but this is exactly the kind of task that we have in mind when we speak of merging information. Merging or integrating information, as we are using those terms, is one of those further applications that needs to be developed on top of the semantic web, along with applications such as intelligent search engines and other envisaged applications in e-commerce and knowledge management. Our view of the semantic web, then, is that it is an enabling technology for information integration. The type of fusion system that we advocate could be considered an information integration agent that could make use of semantic web technology.

The different roles of semantic web technology, on the one hand, and information merging (or knowledge fusion) systems as we are envisaging them, on the other hand, are nicely illustrated in the case of bioinformatics. Ontologies are playing an important role in organizing the vast amount of information generated by research in molecular biology. For example, DAML+OIL has been used to annotate the Gene Ontology (GO) data source[2] which in turn is used by a wider project called TAMBIS — Transparent Access to Multiple Bioinformatics Information Sources (see [BBB$^+$98, SGHB01b, SGHB01a]). The metadata provided by this project allows a single, uniform query interface to diverse bioinformatics sources. So, as its developers claim, here we have "an exemplar of using an ontology to facilitate the interoperation and fusion of bioinformatics sources" ([SGHB01b], p2).

Once again, we may note that what this technology facilitates is better for user access to existing information. But in bioinformatics there is also a need for merging or fusing information in the way that we

[2]The Gene Ontology is available from www.geneontology.org

mean. One example is the need to merge information about the functional annotations of individual protein domains. In some cases different databases assign different GO annotations to the same domain, and one way to resolve such conflicts is by using preferences over sources. Here, this is not simply a matter of extracting information from existing sources; rather, new information needs to be created by, for example, reasoning using a knowledgebase.

2.2 Information mediators and information integration on the web

The aim of merging, as we choose to understand it, is to take potentially conflicting information from different sources and, by reasoning about those conflicts, to create a distinct, novel output report that summarizes, or in other ways combines, that information. The upshot of merging in this sense is that a new source of information is created, something that did not previously exist.

This aim seems to be substantially different from most extant approaches to merging or integrating information, where the emphasis often seems to be more on extracting information from different sources and presenting it in such a way as to make it easier for the user to find the information she wants. The upshot of merging or integration in this sense is the user finds a pre-existing source of information; no new information is created (although, of course, the information will be new to the user — that is why she's looking for it), rather, existing information is made more useable.

The difference between these two projects can perhaps best be brought out in terms of the familiar entity-attribute model. Our primary interest in merging is to take information from different sources about the same entity, where that information concerns the *same* attributes, and then combine it. If the values of those attributes are the same (or similar), not much reasoning will be involved. But where the values of those attributes conflict, a variety of kinds of reasoning using a knowledgebase can be used to resolve the conflict. By contrast, most other projects of integrating information on the web seem concerned with a different task: they start from the same place, with different sources of information about the same entity, but those sources typically contain information about *different* attributes. Thus these different data sources are conceived of as containing *complementary* information, not *conflicting* information. As a result, merging is conceived of, not as the process of resolving a conflict, but as the process of combining the information about the different attributes of a given entity, thus making it easier for the user to find the information she wants[3].

A couple of examples should make this clearer. One example is the TheatreLoc application ([BKY+99]): here the user chooses to search for restaurants and/or cinemas in a specified city, and is presented with a list of them, together with their locations displayed on an interactive map. The user can then navigate via the list or the map to detailed information on individual theatres or restaurants, including watching previews of the films. In this application, information about a theatre's films is brought together with video previews, and information about the street address of the theatre is used to find the theatre's grid reference, allowing it to be placed on a street map. The end product is a user interface making it easier for the user to find what she wants; a single website now allows the user both to find what is showing, and how to get there. In that sense we can speak of information integration.

A second example is an application that seeks to integrate the information contained in restaurant review websites such as Fodor's and Zagat's, with information on the health or sanitation status of those restaurants contained on a local government website [KM98]. Combining this information would allow users to answer such queries as "Find all the Japanese restaurants in Santa Monica with a grade A health rating." Once again, the purpose of integration here is not to reason about conflicting data, but to make it easier for the user to find information which already exists. The end result is that the user is looking at the same web pages that she could have found manually, but she got their a lot more easily.

Both applications work by using the Ariadne information mediator (see [KMA+98, KMA+99]) (which is

[3]Whilst our primary interest in knowledgebase merging is in addressing conflicts, knowledgebase merging can be used to merge complementary information

itself based largely on the SIMS mediator architecture — see [AKS96, Kno95]) forming an intermediate layer between the user and the various heterogeneous information sources. The purpose of the mediator is to provide a uniform query interface, abstracting away from the heterogeneous formats of the different sources. The mediator also plans how the sources should be queried and how the data that is retrieved is to be integrated. The extraction itself is done by wrappers. It should be acknowledged that this sort of information integration does involve some reasoning concerning conflicts (see [TKM98]). But this is confined to determining if information from two sources is in fact about the same entity, say the same film or restaurant. For example, we need to be able to tell that information from a cinema Website about a film called "A bug's Life" is information about the same film that is listed as "Bug's Life, A" on a trailer Website, so that links to the two can be combined. Or, we need to know whether a restaurant listed as "Art's Delicatessen" on one site is the same as one listed as "Art's Deli" on another. The role of reasoning about conflicts thus seems to be restricted to providing robustness to the semantic heterogeneity of different information sources in their description of the entities they are about. These applications do not seem to provide a role for reasoning about conflicts between the values of the attributes of those entities.

There is no question that the sort of issues in integration addressed by these applications are pressing, but the end result of this approach is that it is easier for the user to find one or more of the pre-existing input sources that she is interested in. Some limited resolution of conflicts between sources may be required in order to do this, but it remains a subsidiary goal. By contrast, the objective of our approach is that the user gets to see a wholly different, merged or summarized, output report. The focus of our approach is in developing ways of merging or integration that can deal with a whole array of problems that arise with conflicting information of which the examples just mentioned in the case of these web integration applications are only one.

2.3 Database integration

The goal of database integration is to provide uniform access to multiple, heterogeneous databases that each has its own associated local schema (see [Lev00]). Logic-based techniques in data integration, such as global-as-view and local-as-view, offer some ability to relate sources using restricted forms of first-order logic, and so can be considered as special cases of knowledgebased merging. However, the formats of the clauses used are largely limited to defining virtual tables directly in terms of existing tables. We could describe this process of integration as providing a mapping of one schema onto another, so that, for example, data in a column headed *location* in one table is mapped onto a column headed *address* in another table. So, "fusion" or "integration" in this context refers to the "combining" of several database tables into one. These "mappings" can be regarded as a form of ontological knowledge.

If this is how the goal of database integration should be understood, then we can see, once again, that this goal is substantially different from the primary goal of knowledgebase merging that we are advocating. Data integration in the former sense is the combining of information previously available in different applications, and making that same information available to a single application so as to enable easier access and querying. But having brought the information together, this approach in itself has nothing to say about what should be done with it. By contrast, "integration" as we mean to use the term stands for the use of logical inference to create a new piece of information that was not necessarily previously available. As was the case with the semantic web, far from being a competitor, we can view database integration as a useful enabling technology that provides the kind of input that could subsequently be merged in the ways that we envisage.

Dealing with conflicts is emerging as an important aspect of data integration, and there is increasing interest in querying multiple relational databases that are in conflict [ABC99, BC03]. For example, answers that are consistent with global integrity constraints may be required in a local-as-view paradigm from an incomplete (open) database [BB03]. Techniques being developed for these problems do directly address conflicts as they arise in the input databases, but in so doing they suppress information. These techniques do not allow

the representation of inconsistency in the "merged" information. Furthermore, these approaches are not context-dependent: Each technique offers a single mechanism for aggregation irrespective of the nature of the data, and they cannot take advantage of background knowledge in the process. Furthermore, they do not provide support for analysing the significance of inconsistencies, they do not merge uncertain information, and they do not resolve inconsistencies by finding the most-up-to date pieces of information based on temporal or event-based reasoning. Finally, these approach do not "create" knowledge, they only take a subset of the existing information.

2.4 Summary of background to merging information

In one way or another the various technologies we have briefly examined in this section are concerned with giving the user access to an *existing* piece of information — a piece of information that is already there on the web, or in a database, but that is difficult to access, perhaps because of the vast amount of information available, or because it is spread across more than one database. By contrast, what is different about our view of knowledgebase merging is that it is designed to create a *new* piece of information — the merged output report — from existing sources, by reasoning about conflicts using a knowledgebase.

The second main difference between our approach, and the other approaches to merging information that we have considered here, is that in our approach we have much more comprehensive and complex background knowledge for reasoning with the information to be merged, and thereby we support more context-sensitive merging. However, we should stress that our approach is only intended for focussed domains, and this is in contrast to some of the other techniques which are being developed for much wider and dynamic domains.

3 Knowledgebased merging

For our approach to merging, we have assumed that information to be merged is in the form of a set of structured reports. In particular, each structured report is represented by an XML document. This implicit tree structure for each structured report means each one is isomorphic with a ground term (of classical logic) where each tagname is a function symbol and each textentry is a constant symbol. This in turn means, we can reason with a set of structured reports directly in a knowledgebase by representing them as a list of terms. Furthermore, we can reason with information from structured reports in the form of subterms, and if we want to reason with just textentries from a set of structured reports, then we can represent these textentries by a list of constant symbols in the logic. Fusion rules, which we will explain fully in Section 3.5, are defined to extract the required information from the structured reports prior to reasoning in the knowledgebase.

In this section, we discuss the following issues: (1) The role of the knowledgebase in merging; (2) What kind of knowledge is used in a knowledgebase? (3) A role for description logics? (4) How easy is it to construct a knowledgebase? and (5) Controlling knowledgebased merging using fusion rules.

3.1 The role of the knowledgebase in merging

In order to merge information in the kinds of ways we have indicated (as in, say, Examples 1.1 and 1.2) we use both domain specific and more general knowledge in a knowledgebase. The knowledge in the knowledgebase can be viewed as evaluating one of the following two types of query.

Instrument queries These are queries which test aspects of the input information such as testing whether

or not particular parts of the information from the various input reports are similar in some particular way. So, for example, in merging weather reports, the conditions in a knowledgebase would typically test such things as whether the temperature, humidity, and pressure, etc., as specified by the various input reports were or were not similar, where what counts as being similar is determined by the domain specialist. To illustrate, consider the two input structured reports given in Example 1.1. The textentries for `date` are `19/3/04` and `20 March 2004` respectively, and so an instrument query involving these textentries might be `SameDate`([`19/3/04`, `20 March 2004`]) where `SameDate`(`X`) would be defined to hold if the list `X` contains one or more dates that are the same. In this case, the first of the following does not hold, whereas the second does hold.

`SameDate`([`19/3/04`, `20 March 2004`])
`SameDate`([`19/3/04`, `19 March 2004`, `19/03/2004`])

Further instrument queries may involve a variable that needs to be grounded. Consider the following instrument query `PreferredTerms`(`List`, `X`), where for the list of terms in `List`, the variable `X` is ground by the knowledgebase with a list that contains the preferred synonym for each term in `List`. For example, `PreferredTerms`([`mostlysunny`, `sunny`, `sunshine`, `cloudy`], `X`), if according to the knowledgebase `sun` is the preferred term for sunny weather, and `cloud` is the preferred term for cloudy weather, then `X` is ground by the knowledgebase with the list [`sun`, `sun`, `sun`, `cloud`].

Aggregation queries These queries are used to aggregate a tuple of values. Their general form is a predicate, $P(X_1, .., X_n, Y)$, where P is the name of some aggregation predicate, $X_1, .., X_n$ are the arguments containing the terms to be merged from the input reports, plus terms providing contextual information, and Y is the output of aggregating those textentries (see Example 3.1). So for example, if the majority of the input reports agree on today's weather, then majority aggregation selects the term used by the majority of the input reports, and uses that in the output report. Or, in the case where the input information is a numerical value, such as temperature, pressure or humidity, if the input values are all within an allowable range, then the interval from the minimum to the maximum of the input values might be selected for inclusion in the output report. Here the aggregation predicate, might look like this: `TempInterval`([`15C`, `13C`, `58F`, `13C`, `12C`], `X`), where X is ground to $12 - 15$, assuming that the knowledge engineer has determined that a 3C variation is within the acceptable range. Matters are more interesting, however, if the input reports are not deemed to be similar by the knowledge engineer, and so some similarity condition in the knowledgebase fails. In such cases a much richer variety of aggregation functions is available. Conflicts can be dealt with by aggregation functions that implement various sorts of voting functions, preferences over sources, or if the conflicting terms fall into a class hierarchy, the most specific term that is general enough to include them all could be chosen. Whichever method of aggregation is selected, the choice is encoded in the knowledgebase by the knowledge engineer or user according to the needs of the domain in question.

To answer instrument queries and aggregation queries, the knowledgebase requires the definition of clauses that with an inference mechanism can be reasoned with to determine whether the query holds or not, and if there is a variable in the query, finds groundings for the variable in the query. For our approach, we can use Prolog for the language of the knowledgebase and for the inference mechanism. The choice of Prolog was made because of the availability of high performance implementations, associated software, the generality of the language, and the well-developed theoretical foundations. See [Bra01] for a comprehensive introduction to Prolog. However, it would be straightforward to adapt our approach for a variety of other approaches to representing and reasoning with knowledge.

Example 3.1 *The following two examples show how instrument and aggregation predicates work together to merge information from the input reports. In the first example the instrument and aggregation predicates could be used in merging the reports from Example 1.1:*

`SameParty`(`Parties`)

`Mean`(`Values`, `Mean`)

Where Parties *is a list of parties, one from each report;* Values *is a list of poll results, again one from each report; and* Mean *is the mean of all the results in list* Values*. These two predicates would be applied to each of the three parties occurring in the input reports.* SameParty *is a query that is called first to check that the parties named in the list* Parties *are all in fact the same party, to ensure that* Mean *would be the mean of the polls for a single party. Note that as political parties sometimes have different names ("Conservative" and "Tory", "Republican" and "GOP", etc.), or because their names are sometimes abbreviated ("Labour" and "Lab", "Conservative" and "Cons", etc.), this condition has to do more than simply check that each element in the list* Parties *is the same: It would also require several domain-specific facts to be checked in turn. Like many instrument queries, it simply succeeds or fails. If it succeeds,* Mean *is a utility aggregation predicate that provides the mean. The* Mean *predicate is an example of a predicate that could of course be reused in other applications.*

With their arguments grounded with textentries from the two input reports, the two predicates would look like the following, and X *would be ground to* 39.5%*.*

$$\texttt{SameParty}([\texttt{Labour}, \texttt{Lab}])$$

$$\texttt{Mean}([41\%, 38\%], \texttt{X})$$

Example 3.2 *The second example consists of an instrument and an aggregation predicate that could be used in merging the reports from Example 1.2:*

$$\texttt{ConflictingRecommendations}(\texttt{Recommendations})$$

$$\texttt{UsePreferredSource}(\texttt{Sources}, \texttt{Recommendations}, \texttt{PreferredRecommendation})$$

Where Recommendations *is a list of recommendations, one from each report;* Sources *is a list of the sources of the recommendations, again one from each report; and* PreferredRecommendation *is the overall recommendation selected on the basis of a preference over the sources. The first predicate is the condition predicate that checks that the recommendations do in fact conflict. If they do, the second predicate then selects the recommendation from the most preferred source.*

With their arguments grounded with textentries from the four input reports, the two predicates would look like the following, and X *would be ground to* sell*.*

$$\texttt{ConflictingRecommendations}([\texttt{buy}, \texttt{sell}, \texttt{buy}, \texttt{buy}])$$

$$\texttt{UsePreferredSource}([\texttt{FinancialTimes}, \texttt{HSBC}, \texttt{Waterhouse}, \texttt{MorganStanley}], [\texttt{buy}, \texttt{sell}, \texttt{buy}, \texttt{buy}], \texttt{X})$$

If, instead, the overall recommendation was made on the basis of some form of voting, the sources could be used as a basis of weighting the votes (see below), or if weighting was not to play a part, the first argument could be omitted.

$$\texttt{WeightedMajority}(\texttt{Sources}, \texttt{Candidates}, \texttt{Winner})$$

WeightedMajority *is an aggregation predicate that selects the element from list* Candidates *that has the most votes according to some weighting scheme defined over the sources in* Sources*, provided that the winning element has more than 50% of the votes cast. If no candidate meets this requirement, the output of the aggregation predicate,* Winner*, is the string* NoMajority*, which occurs in the merged report as the text entry for the tag* overallrecommendation*. Note that, once again, in both cases the aggregation functions are generic and so could be reused in other applications, though each does require domain-specific facts in order to select the preferred source, or compute the weighted vote totals.*

Queries to the knowledgebase reflect both the tagnames in the structured input reports and, of course, the interests we have in merging those reports. Thus in the case of merging weather reports, the queries will likely involve predicates for similar temperatures, similar pressures, and so forth. Other queries will reflect the kinds of aggregation functions used.

But the knowledgebase will contain much else besides these top-level predicates. In addition, there will need to be facts about, for example, which sources are acceptable, and perhaps a preference ordering over them as well, if that is to be used as a method of aggregation in the case of conflicts. Other facts might include equivalence classes of expressions, say for today's weather (for example, {rain, wet, inclement, downpours, heavy rain, prolonged rain}), preferred terms from amongst these equivalence classes, and class hierarchies where they are appropriate. Besides facts such as these, a number of rules will also be required in order to carry out the subsidiary tasks involved in implementing the top-level predicates. Many of these rules will implement utility functions such as constructing conjunctions or disjunctions of input textentries. Others will remove duplicates from the lists of such textentries, sum the values in a list, calculate their mean (see Example 3.1), or determine the maximum and minimum values in such lists.

In many cases, before any similarity comparisons can be made, or before any aggregation functions can be calculated, numerical textentries will need to have their unit suffixes removed ("C", "F", "mph", "mb", "%", etc.), and unit conversions made, if required (kph to mph, Fahrenheit to Centigrade, etc.); so, much of the subsidiary computation in the knowledgebase is concerned with the string manipulation required to do this.

3.2 What kind of knowledge is used in a knowledgebase?

The knowledge in the knowledgebase can be seen as falling into one of two categories: domain (or specific or domain-specific) knowledge and generic (or general) knowledge.[4]

Domain knowledge includes: (1) Ontological knowledge. For example, knowledge specifying which terms are synonyms or can be seen as belonging to the same equivalence class ("drizzle" and "light rain", "Bordeaux" and "Claret", etc.); and hierarchical information about more general and less general terms ("rain" is less general, or is a subclass of "precipitation", etc.). (2) Relational knowledge. For example, extra data about the input information that can be used to help resolve conflicts or extra information that can be added to fill gaps in the information to be output (such as "Paris is the capital of France", and "Shell is listed on the London Stock Exchange"). This data can often be sub-contracted out to relational databases. (3) Metalevel knowledge. For example, preferences over sources, or information about the reliability of sources. (4) Coherence knowledge. For example, terms which are consistent ("rain" and "wind") and terms which are not ("overcast" and "sunny"). (5) Knowledge about allowable ranges of various values. For example, what variations in temperature, windspeed and humidity are compatible with a set of input reports being seen as not conflicting (such as the midday temperature in London being 5-10C is an acceptable range, whereas it being 5-25C is not acceptable since it is too vague).

Generic knowledge, on the other hand, is knowledge that can be used in multiple applications. We can draw on extensive research in knowledge representation and reasoning to define generic knowledge with well-understood behaviours.

Drawing on the literature in aggregation functions, we have implemented a whole range of aggregation predicates (see [HS04a]). These include simple functions such as disjunction and conjunction, but also more complex ones such as semantic generalization, that is, finding the most specific term that is general

[4]This distinction should not be confused with the distinction, familiar from work in description logics, between intensional and extensional knowledge. The former is often described as general knowledge, but this is because it is about *concepts* in the knowledgebase rather than individuals, which are the subject of extensional knowledge. However, intensional knowledge is still knowledge that applies to a particular domain, and so could not be reused across different applications about different subjects. Hence, it does not qualify as general knowledge as we understand it.

enough to subsume the input terms. There are also many voting functions for resolving conflicts that can be included in the knowledgebase, and these can all take unweighted and weighted forms [CG04]. Various kinds of preference functions defined over the sources of the input reports can also be used to resolve conflicts when merging. In the case of many of these functions the rules are generic, but they will also rely upon facts that are themselves domain-specific. So for example, both the facts embodying the concept hierarchies used in finding the semantic generalization of different input textentries, and the facts specifying the order of preferences for the sources of reports will be domain-specific. But the rules themselves are generic, and so using functions such as these it is straightforward to compile a library that can be used for resolving conflicts in a variety of applications.

We have also investigated the following types of generic knowledge: (1) Reasoning about time and events (Section 3.2.1) and using this for defining an event-based form of aggregation; (2) Reasoning about uncertainty (Section 3.2.2) and using this for defining a class of aggregation predicates that take uncertainty into account; and (3) Reasoning about inconsistency (Section 3.2.3) and using this for context-sensitive selection of an aggregation function.

3.2.1 Reasoning about time and events

Many kinds of information involve information about time and events. So when merging such information, there is often a need to obtain a merged view of the temporal and event information. Furthermore, time and events can be used to resolve some kinds of conflict. For example, a general heuristic is that more recent information is more reliable and so should be chosen in case of conflict.

Many kinds of news report provide information about events. For example, business news reports in the area of mergers and acquisitions, provide information about events such as "company X makes a bid for company Y", or "takeover of company Y by company X is rejected by the anti-trust authorities". In order to merge heterogeneous news reports that describe events, we need to identify and reason about the events being described prior to merging them.

Clearly, news reports do not normally exist in isolation. They are usually part of "narratives" which relate them to other articles that deal with the same story. For example, in the mergers and acquisitions domain, we may find a news report announcing a takeover bid followed by a news report of the bid being accepted by the board, followed by a report on the shareholders voting whether to accept the bid, and so on. All news reports belong to at least one narrative and each narrative involves one or more reports.

Example 3.3 *Consider the following two conflicting (fictitious) business reports. The left report states that the value of the bid is* `$35Billion` *and the capital of the target is* `$25Billion`*, and the right report states that the value of the bid is* `$47Billion` *and the capital of the target is* `$50Billion`*.*

```
⟨businessreport⟩
        ⟨source⟩ Reuters ⟨/source⟩
        ⟨reportdate⟩ 11/01/02 ⟨/reportdate⟩
        ⟨action⟩ unexpected takeover offer ⟨/action⟩
        ⟨bidvalue⟩ $35Billion ⟨/bidvalue⟩
        ⟨buyer⟩ Shell ⟨/buyer⟩
        ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩
                ⟨capital⟩ $28Billion ⟨/capital⟩
        ⟨/target⟩
⟨/businessreport⟩
```

```
⟨businessreport⟩
        ⟨source⟩ Reuters ⟨/source⟩
        ⟨reportdate⟩ 13 March 2002 ⟨/reportdate⟩
        ⟨action⟩ revised takeover offer ⟨/action⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨buyer⟩ Shell ⟨/buyer⟩
        ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩
                ⟨capital⟩ $34Billion ⟨/capital⟩
        ⟨/target⟩
⟨/businessreport⟩
```

We can merge them to resolve these apparent conflicts. A simple approach is to take the most recent report. A better approach that is easier to justify is to consider the narrative of the news reports. In this case, the right report involves a more recent event `revised takeover offer`*, and so describes a more up to*

date state. In this example, both approaches lead to the same conclusion. But this is not always the case, particularly in situations where there is a time delay in reporting the news.

```
⟨businessreport⟩
        ⟨source⟩ Reuters ⟨/source⟩
        ⟨reportdate⟩ 13/3/02 ⟨/reportdate⟩
        ⟨action⟩ revised takeover offer ⟨/action⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨buyer⟩ Shell ⟨/buyer⟩
        ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩
                ⟨capital⟩ $34Billion ⟨/capital⟩
        ⟨/target⟩
⟨/businessreport⟩
```

Example 3.4 *Consider the following two conflicting (fictitious) business reports. The left report states that the action is* `takeover offer` *and the right report states that the action is* `bid accepted`*.*

```
⟨businessreport⟩
        ⟨source⟩ Reuters ⟨/source⟩
        ⟨reportdate⟩ 13/03/02 ⟨/reportdate⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨action⟩ takeover offer ⟨/action⟩
        ⟨buyer⟩ Shell ⟨/buyer⟩
        ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩
                ⟨sector⟩ oil ⟨/sector⟩
        ⟨/target⟩
⟨/businessreport⟩
```

```
⟨businessreport⟩
        ⟨source⟩ DowJones ⟨/source⟩
        ⟨reportdate⟩ 13 March 2002 ⟨/reportdate⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨action⟩ bid accepted ⟨/action⟩
        ⟨target⟩ Texaco ⟨/target⟩
        ⟨buyer⟩
                ⟨company⟩ Shell ⟨/company⟩
                ⟨sector⟩ oil ⟨/sector⟩
        ⟨/buyer⟩
⟨/businessreport⟩
```

We can merge them so the more recent event is used. But we can also take the information from the previous report that is not invalidated by the newer report.

```
⟨businessreport⟩
        ⟨source⟩ Reuters and DowJones ⟨/source⟩
        ⟨reportdate⟩ 19/3/02 ⟨/reportdate⟩
        ⟨action⟩ bid accepted ⟨/action⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨buyer⟩
                ⟨company⟩ Shell ⟨/company⟩
                ⟨sector⟩ oil ⟨/sector⟩
        ⟨/buyer⟩
        ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩
                ⟨sector⟩ oil ⟨/sector⟩
        ⟨/target⟩
⟨/businessreport⟩
```

In order to represent and reason with narratives, we need to model states and changes of state. To address this need, we can use an event reasoning system based on a variant of the event calculus to represent and reason with such information. Event calculus was originally proposed by Kowalski and Sergot [KS86]. Since then a number of variants of event calculus have been proposed. For a review see [Sha99, MS99]. These variants of event calculus have been presented in either classical logic or in Prolog.

A framework for taking a list of structured reports as input and producing a merged structured report as output has been presented based on knowledgebased merging [HS05]. The framework includes a format for axioms for extracting information about events from the set of input reports, a form of transition model as

a way of representing key inter-dependencies between events, a new variant of event calculus for reasoning about events in news reports, and a definition for an aggregation predicate for merging a set of structured reports based on the underlying model.

3.2.2 Reasoning about uncertainty

A number of mechanisms for reasoning under uncertainty have been proposed and studied over the past 20 years. Probability theory, Dempster-Shafer theory of evidence (DS theory) [Dem67, Sha76], and possibility theory [DP88] are the three popular ones that have been widely applied in many different domains. Probability theory, the most traditional method, is one of the first to be used to represent uncertain information in databases (e.g., [CP87, BGP92, NS94]). The restriction of assigning probabilities only to singleton subsets of the sample space led to the investigation of deploying DS theory (e.g., [Lee92]) where uncertainty can be associated with a set of possible events. A mass function m assigns each unit of an agent's belief to a distinct subset of a set of all possible outcomes, with the total sum of all assignments being 1. When all the distinct subsets are singletons, a mass function is reduced to a probability distribution. It is in this sense that DS theory can be regarded as a generalization of probability theory. Possibility theory is another option to easily express uncertainty in information. A possibility measure Π on a subset A of a set of possible events gives a value in $[0, 1]$ which estimates to what extent A contains the true event. The dual function, necessity measure N with $N(A) = 1 - \Pi(\bar{A})$ where $\bar{A}$ stands for the complementary set of A, evaluates the degree of necessity that A is true.

Using these approaches to representing and reasoning with uncertainty, structured reports can be annotated with information about the uncertainty associated with particular textentries or subtrees. Furthermore, these mechanisms can be used in knowledgebases to support the derivation of inferences with associated information about the uncertainty. In [HL05a] a framework for merging structured reports with uncertainty has been proposed. This has involved the modelling and merging of uncertain information associated with textentries in XML documents. Multiple pieces of uncertain information concerning the same issue are assumed to be specified on the same set of possible values. For example, in weather reports, an uncertainty distribution (such as probability distribution) can be given for `outlook` so that $\texttt{P}(\texttt{sunny}) = 0.3$, $\texttt{P}(\texttt{overcast}) = 0.6$, and $\texttt{P}(\texttt{rainy}) = 0.1$

However, [HL05a] does not consider situations where one piece of information uses more specific values than another nor the situation where one piece of information is described on one set of values and another is on a different set of values where these two sets of values are inter-connected. To elaborate this issue further, let us consider a very simple example about clinical records. Assume that for each patient, we only conclude whether the patient has cancer or not, regardless of what type of cancer it could be. So we use a set of values $\{\texttt{cancer}, \texttt{noCancer}\}$ to bare any information we have about the patient. However, we could make this information more specific by giving different types of cancer, such as, *skin cancer, breast cancer, bone cancer, etc*. Therefore some uncertain information can be described on the set of values $\{\texttt{noCancer}, \texttt{skinCancer}, \texttt{breastCancer}, \texttt{boneCancer}, \texttt{etc}\}$. This latter set of values has a finer granularity than the former one. Furthermore, since cancer diagnosis is often done through some tests other than being observed directly, test results will directly influence the conclusion $\{\texttt{cancer}, \texttt{noCancer}\}$. For instance, it is commonly known that a high level of prostate specific antigen (PSA) can indicate that a patient has prostate cancer[5]. Assume the level of prostate specific antigen lies in $[0, 15]$, and a patient's PSA level is measured as 11, then we may conclude that this patient has prostate cancer with probability 0.65. In this situation, the information is given on one set of values (as a PSA level between $[0, 15]$) and the conclusion is on another set $\{\texttt{cancer}, \texttt{noCancer}\}$. The information from the given set of values should be propagated to the destination set of values as a new distribution of beliefs.

To deal with these situations, in [HL05b, HL05c, HL05d], we have further extended the approach to merging multiple pieces of uncertain information to situations where

[5]For details on PSA, see medic.med.uth.tmc.edu/ptnt/00000390.htm.

- evidence is specified at different levels of granularity on the same concept as textentries. We refer to two pieces of this type of evidence as *semantically homogeneous*. In this case, a value in a coarse set can be replaced by a set of values in a finer set.
- evidence is specified on inter-related concepts as textentries. We refer to two pieces of this type of evidence as *semantically heterogeneous*. The PSA example above belongs to this category.
- evidence is assigned to heterogeneous subtrees involving multiple concepts. We refer to two pieces of this type of evidence as *semantically heterogeneous*. For instance, if we have a set of values measuring PSAs and another set measuring blood pressures, then the joint set from these two sets says what PSA level and what blood pressure a patient has.

Using Dempster-Shafer theory, we can harness Dempster's rule of combination that allows for the combination of two or more uncertainty distributions [HL05a, HL05b, HL05c, HL05d]. This forms the basis of a useful class of aggregation predicates with intuitive properties.

3.2.3 Reasoning about inconsistency

Comparing heterogeneous sources often involves comparing conflicts. Suppose we are dealing with a group of clinicians advising on some patient, a group of witnesses of some incident, or a set of newspaper reports covering some event. These are all situations where we expect some degree of inconsistency in the information. Suppose that the information by each source i is represented by the set Φ_i. Each source may provide information that conflicts with the domain knowledge Ψ. Let us represent $\Phi_i \cup \Psi$ by Δ_i for each source i. Now, we may want to know whether one source is more inconsistent than another — so whether Δ_i is more inconsistent that Δ_j — and in particular determine which is the least inconsistent of the sources and so identify a minimal Δ_i in this inconsistency ordering. We may then view this minimal knowledgebase as the least problematical or most reliable source of information.

When an intelligent agent works with a set of information, beliefs, knowledge, preferences, ... expressed in a logical form, the notion of informational content of a piece of information and the notion of amount of contradiction are of crucial interest. Effectively, in many high-level reasoning tasks one needs to know what is the amount of information conveyed by a piece of information and/or what is the amount of contradiction involved with this piece of information. This is particularly important in complex information about the real world where inconsistencies are hard to avoid.

While information measures enable us to say how "valuable" a piece of information is by showing how precise it is, contradiction measures enable us to say how "unvaluable" a piece of information is by showing how conflicting it is. As joint/conditional information measures are useful to define a notion of pertinence of a new piece of information with respect to an old one (or more generally for a set of information), joint/conditional contradiction measures can be useful to define a notion of conflict between pieces of information, that can be useful for many applications. These two measures are to a large extent independent of one another, but needed in numerous applications.

Five key approaches to measuring inconsistent information are: Consistency-based analysis that focuses on the consistent and inconsistent subsets of a knowledgebase [Hun04]; Information theoretic analysis that is an adaptation of Shannon's information measure [Loz94]; Probabilistic semantic analysis that assumes a probability distribution over a set of formulae [Kni01, Kni03]; Epistemic actions analysis that measures the degree of information in a knowledgebase in terms of the number of actions required to identify the truth value of each atomic proposition and the degree of contradiction in a knowledgebase in terms of the number of actions needed to render the knowledgebase consistent [KLM03]; and model-theoretic analyses that are based on evaluating a knowledgebase in terms of three or four valued models that permit an "inconsistent" truth value [Hun02b, Hun03]. Whilst inter-relationships between these approaches are yet to be fully

established, it is clear they offer a range of formalisms that can be harnessed via a knowledgebase for evaluating sources of information prior to merging.

Having some understanding of the "degree of inconsistency" of a structured report can help in deciding how to act on it. Moreover, inconsistencies between information in a structured report and domain knowledge can tell us important things about the structured report. For this we use a significance function to give a value for each possible inconsistency that can arise in a structured report in a given domain [Hun05]. We may also use significance in the following ways: (1) to reject reports that are too inconsistent; (2) to highlight unexpected information; (3) to focus on repairing significant inconsistencies; and (4) to monitor sources of information to identify sources that are unreliable.

3.3 A role for description logics?

As we have already seen, description logics have been used both to provide a formal semantics for the kind of knowledge captured in hierarchies of classes and individuals, and to enable reasoning over those hierarchies. Because these hierarchies are themselves knowledgebases it is natural to ask whether description logics have a role to play in the kind of knowledgebased approach to merging that we are discussing. The answer is: yes, but at least for most applications, it only provides a fraction of the knowledge needed. This becomes apparent when we consider the kinds of reasoning that description logics facilitate. The main kinds of reasoning fall under the following headings.

Classification This is the positioning of a new concept in the correct place in the hierarchy, that is, below the most specific concept that subsumes the new concept, and above the most general concept that the new concept subsumes.

Instance checking This is the verification that a given individual is an instance of a specified concept.

Knowledgebase consistency The verification that every concept in the knowledgebase has at least one individual.

Realization The determining of the most specific concept that a given individual is an instance of.

Least Common Subsumer The least common subsumer of a set of concepts is the most specific or least general concept in the class hierarchy that subsumes all of them.

It may be, then, that since the hierarchies associated with description logics *are* knowledgebases, and description logics can be used to reason about those hierarchies, description logics could play some role in the kind of merging we envisage. But given the kind of reasoning that we anticipate will be exploited in merging information (preferences over sources, voting, allowable ranges, etc.), that role will inevitably be limited. In order to implement these kinds of reasoning, a knowledgebase for merging will have to contain far more than the class hierarchies used with description logics.

The most obvious example of reasoning involved in merging where a description logic would be applicable is that involved in finding what we have termed the *least upper bound* of a number of concepts, and what is often referred to as finding the least common subsumer; that is, the most specific concept that is general enough to subsume the diverging concepts in the input reports. This kind of reasoning would seem to offer a reasonable approach to merging conflicting concepts that fell under a class hierarchy. However, even in this case there are many other ways of resolving such conflicts and a knowledge engineer may well deem another approach more appropriate for a particular application. And moreover, there are many other kinds of conflicts that cannot be resolved in this way (e.g. conflicts concerning numerical values). In conclusion, an ontology is only one kind of knowledgebase. By itself it is unlikely to be sufficient to resolve all of the conflicts encountered in merging information.

3.4 How easy is it to construct a knowledgebase?

As a general remark, the application-specific, or domain-specific, nature of the approach to merging that we advocate avoids the well-known obstacles to constructing broader knowledgebased systems for tasks such as decision-support or planning. More specifically, there are two further reasons that help to mitigate the problems involved in constructing a knowledgebase.

First, for many applications much of the domain knowledge required is already readily available in a usable, or near usable, form. For example, in an application in bioinformatics (see Examples 3.9 and 3.10) groups of protein domains were to be grouped together by finding the most specific functional annotation that was general enough to include all their individual molecular functions. The hierarchical information required to do this already existed in the Gene Ontology (GO) database. A very simple Prolog program transformed the relations into Prolog facts,[6] creating around twenty thousand facts, including both parent-child relationships using the GO accession numbers, and further facts for the actual functional annotation associated with each number. In addition, it required a few simple rules to extract from those facts the ancestor terms (their "lineages") for each term in the hierarchy, and to use those lineages to find the first common ancestor for a group of protein domains (that is, the common ancestor with the greatest depth, or least height, in the hierarchy).

Second, as we have seen, much of the information in the knowledgebase can be seen as generic, and obviously the generic knowledge is simply reusable across applications. For these reasons, then, constructing a knowledgebase for use with an application does not necessarily involve a large scale exercise in knowledge engineering. As we have indicated in Section 3.2, we can also draw on a large literature in knowledge representation and reasoning for defining generic knowledge. We have harnessed this for reasoning with time and events and for reasoning with uncertainty. There are many further types of knowledge we can consider, such as for spatial reasoning and for legal reasoning.

If merging information is to be achieved by using background information in a knowledgebase, the question arises how the information to be merged is going to be used to query the knowledgebase. We propose the use of fusion rules, and it is to a discussion of this technology that we now turn.

3.5 Controlling knowledgebased merging using fusion rules

So far with our presentation of knowledgebased merging, we have described how a knowledgebase can be used to answer queries that are relevant to merging structured reports. However, we need to consider an overarching mechanism for controlling knowledgebased merging. This mechanism needs to take information from the input information (in the form of a set of structured reports), and produce the output information (in the form of a structured report).

In our approach to merging structured reports, we draw on domain knowledge to help produce merged reports. The approach is based on fusion rules[7] (using FusionRuleML [HS03a]). These rules are of the form $\alpha \Rightarrow \beta$, where if α holds in the knowledgebase (i.e. it is a conjunction of queries that succeed in the knowledgebase), then β is an instruction that needs to be undertaken in the process of building a merged news report.

Example 3.5 *Consider the following fusion rule which we will use with the top two structured reports given in Example 1.1.*

$$\texttt{Conjunction}(\texttt{//opinionpoll/source}, \texttt{X}) \Rightarrow \texttt{AddText}(\texttt{X}, \texttt{pollofpolls/sources})$$

[6] Alternatively, the Prolog knowledgebase could simply query the SQL database as required. On this architecture, the "low level" facts would remain in the database, while the Prolog knowledgebase would be restricted to the "higher level" reasoning.

[7] Examples of fusion rules together with XML representation are available at www.cs.ucl.ac.uk/staff/a.hunter/frt

The notation //opinionpoll/source *is a schema variable that is ground by a list of textentries. Each of these textentries is obtained from the* opinionpoll/source *branch of each of the structured reports to be merged. In this example, the list is* [Mori, Gallup]. *So the ground version of the fusion rule is as follows.*

$$\texttt{Conjunction}([\texttt{Mori}, \texttt{Gallup}], \texttt{X}) \Rightarrow \texttt{AddText}(\texttt{X}, \texttt{pollofpolls/sources})$$

Assuming that the knowledgebase incorporates appropriate clauses, the Conjunction([Mori, Gallup], X) *query succeeds with* X *being ground to* Mori and Gallup. *Since all the conditions of the fusion rule hold, the following action* AddText(Mori and Gallup, pollofpolls/sources) *needs to be undertaken as part of the process of building the merged report. This action is of the form* AddText(Textentry, Branch) *and it requires that* Textentry *is the textentry added as the leaf on* Branch. *We call* Branch *the target constant.*

Fusion rules are thus comprised of condition literals (the antecedent of the rule) and action literals (the consequent). A single rule may have zero or more conditions, and one or more actions. Both conditions and actions are themselves comprised of schema variables, logical variables, and constants. Schema variables specify both what information is to be extracted from an input report, and from which report it is to be extracted. This information is then used to instantiate the fusion rules prior to querying the knowledgebase. Because either individual reports or the whole set of input reports may be specified as the source, schema variables are of two types: set variables and singleton variables. Constants take constant values which either denote targets in the merged report to which text entries and subtrees are to be added, or numerical values specifying, for example, voting thresholds or the number of nodes to be added to the output report, etc. Finally, logical variables indicate where the knowledgebase is expected to provide a binding that may be used either as input to a subsequent condition, or as output for the merged report.

To merge a set of structured reports, we start with the knowledgebase and the information in the news reports to be merged, and apply the fusion rules to this information. For a set of structured reports and a set of fusion rules, we attempt to ground each fusion rule with textentries or subtrees from the structured reports, and then check whether all the conditions of each ground fusion rule are implied by the background knowledge, and if they are, then the ground actions of the rule are added to the actionlist (a list of actions that specify how the merged report should be constructed).

The FusionEngine is the software that is responsible for taking each fusion rule in turn, grounding the schema variables with the required information from the input reports, and posting the resulting queries to the knowledgebase, and if all the conditions succeed, then posting the resulting actions to the ActionEngine. The ActionEngine is the software responsible for taking the actions and building the merged reports. The ActionEngine is implemented to handle a range of actions including the following.

1. Initialize($\phi(\psi_1, .., \psi_n)$) where $\phi(\psi_1, .., \psi_n)$ is called a skeleton constant. The intended action is to start the construction of the merged structured news report with $\phi(\psi_1, .., \psi_n)$ defining the basic structure. So the root of the merged report is ϕ, and by recursion ψ_1 and .. and ψ_n define the subtrees of the merged structured report.

2. AddText(T, $\phi_1/../\phi_n$) where T is a textentry, and $\phi_1/../\phi_n$ is a branch constant. The intended action is to add the textentry T as the child to the tagname ϕ_n in the merged report on the branch $\phi_1/../\phi_n$.

3. AddNode(N, $\phi_1/../\phi_n$) where N is a tagname for a node, and $\phi_1/../\phi_n$ is a branch constant. The intended action is to add N as the child to the tagname ϕ_n in the merged report on the branch $\phi_1/../\phi_n$.

A fusion system for merging structured reports for a particular domain incorporates software modules including a FusionEngine and an ActionEngine. It also incorporates a set of fusion rules that has been defined for the application domain together with an appropriate knowledgebase. The basic architecture for a fusion system is given in Figure 1.

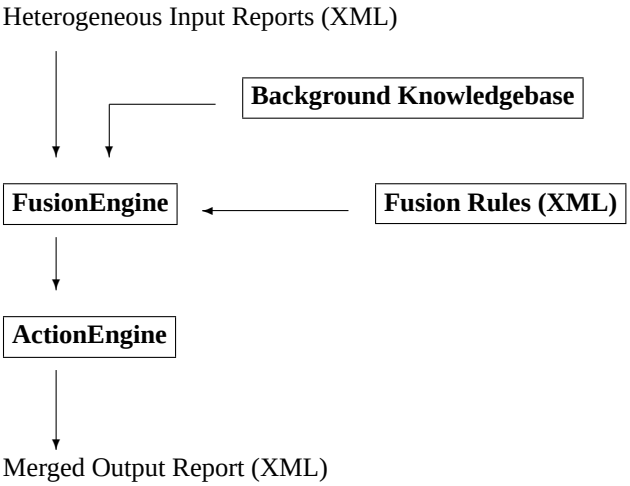

Figure 1: The basic architecture for a fusion system that merges information in the heterogeneous input reports using a knowledgebase and fusion rules. The FusionEngine and ActionEngine modules are implemented in Java. The fusion engine executes a set of fusion rules (an XML file containing the fusion rules marked up in FusionRuleML [HS03a]), by grounding the fusion rules with textentries from the structured reports, and then checks whether all the conditions of each ground fusion rule are implied by the knowledgebase and, if they are, then the ground actions of the rule are added to the actionlist (a list of actions that specify how the merged report should be constructed). The ActionEngine executes the actionlist to build a merged report.

A comprehensive introduction to fusion rules together with a number of worked examples of their execution in a fusion system is given in [HS04a]. In the following, we consider some of the features of a well-engineered set of fusion rules. In particular, we address the question: how difficult is it to engineer a correct set of fusion rules for an application? The aim of the following is to address some practical issues regarding the use of fusion rules. We focus on these questions in order to consolidate our argument that knowledgebase merging using fusion rules is a viable and useful approach to merging heterogeneous structured reports.

3.5.1 Foundational rules

We turn now to a discussion of what constitutes a well-engineered set of rules and how easy is it to produce such a set. First, we discuss why some rules ought to have a privileged status. We classify rules with this status as *foundational*. Their privileged status consists in the fact that if any foundational rule fails to fire, then no merged report is to be constructed. We may say, then, that foundational rules have a veto over the production of a merged report. There is, however, one exception to this. If a fusion rule is deemed foundational because it contains an Initialize action (see below), the first such rule to fire causes any subsequent foundational rule with Initialize actions to be disabled.

There are two reasons why a rule is designated as foundational: (1) It contains an action whose role is to initialize the construction of the merged report. At the very least this will specify the root node of the report. We will call such actions Initialize actions. (2) It contains some condition whose success is essential to the integrity of the merged report. In the first case, if a rule containing an Initialize action were to fail, then we would not want any other rules to execute because without this initial skeleton there would be no merged report to which other rules could add text, nodes, etc.

In the second case, the reason why rules involving certain conditions may be given foundational status is that the failure of those conditions is deemed (by the rule engineer) to fatally threaten the integrity of the merged report. In the case of a set of rules for merging weather reports, for example, if the input reports were not for the same geographical location (city, region, country), or were not for the same time period, or were not from accepted sources, then (in most cases at any rate) there is simply no point in trying to construct a merged report (what point is there in merging today's weather in New York and today's weather in Tokyo, or yesterday's weather in London and tomorrow's weather in Rome?). By contrast, if the windspeeds, temperatures, humidities, etc., fail to be similar enough to be included as is in a merged report, then typically the conflict will be resolved by applying some aggregation function (e.g. voting, preference over sources, etc.) and the result will be added to the merged report. And even if it was not thought useful to do this, that should not prevent a merged report from being constructed that included other information on, for example, air pollution or visibility. Hence rules involving these input report elements would usually be considered to be optional, not foundational.

A well engineered set of rules must then contains at least one foundational rule. Because the success of these rules is in one way or another essential to the construction of a merged report, all foundational rules must *precede* all non-foundational rules.

The requirement for some rules to have foundational status results, in part, from the nature of the information being merged. An input report can be viewed as being concerned with an entity in some loose sense (a weather report, a protein domain, an opinion poll, etc.) and a number of attributes of that entity. Some elements in the report serve to identify the entity in question, whilst other elements serve to express the attributes of that entity. A weather report, for example, might have elements for source, date, and location, serving to individuate that report, whilst it might also have elements with textentries for temperature, humidity, and so on, specifying the attributes of that weather report.

The significance of this distinction is that rules that are concerned with these two types of report elements can be viewed as having different functions. Rules that deal with elements that individuate reports serve

to determine if merging should take place; if we do not have the right kind of entities, a pre-condition for merging does not obtain. Hence, conditions dealing with such elements tend to be placed in foundational rules. By contrast, rules that deal with the attributes of the entities being merged serve to determine the substantive content of the merged report. If these attributes are in conflict (conflicting reports for today's weather, etc.) then some rules will fail. But that is not a reason for not merging; rather, that is an important fact that must be dealt with and included in the merged report. Hence, rules dealing with this second type of report element tend to be given optional status.

It should be noted that sometimes it is not necessary to formulate rules for the individuating or identifying elements of input reports. This is because some form of preprocessing or preselecting of the input data has taken place. In the case of merging protein domains so as to find functionally similar subgroups, for example, the set of input data all come from the same superfamily, so the rules can simply go ahead and attempt to merge the domains. Similarly, if we wished to rely on the user to make sure that all the input weather reports were for the same time and place, and were all from accepted sources, then no rule would be required to check this. This is not to say, however, that the idea of a foundational rule goes by the board; this special status for a rule is still required because of the need of every rule set to contain a rule with an Initialize action.

3.5.2 Rules with Initialize actions

There is no reason why a set of rules should not contain more than one rule with an Initialize action, but for obvious reasons — we cannot build a merged report with more than one root node — only one such action should be executed for any set of input reports.

This imposes the following five constraints on the use and occurrence of Initialize actions: (1) Every set of rules must contain at least one rule with an Initialize action. (2) A single rule may contain at most only one Initialize action. (3) Such actions must be the *first* action in a rule. Quite often, for example, a single rule contains an Initialize action followed by several actions adding text or nodes to the skeleton thus created. Clearly, if the order of these actions was reversed, then some actions would be seeking to add text and nodes to a skeleton that does not yet exist. (4) For a similar reason, rules with Initialize actions should precede all other rules. (5) For any given set of input reports, at most only one rule with an Initialize action should fire. The conditions of these rules should thus be engineered accordingly.

3.5.3 Grouping rules into subsets

We turn now to a discussion of how a set of rules should be constructed if they are to have the correct logical behaviour with respect to a set of input reports. The fact that for a given set of rules and for any set of input reports, at most one rule with an Initialize action should fire brings it out that, although rules are individually numbered, the rules in a well-engineered rule set should not act in isolation from each other. Certain rules bear close relations to some rules that they do not bear to others. In particular, we formulate the following condition:

Condition 1: Rules whose actions are concerned with the same merged report element should be such that, for any given set of input reports, *at most one such rule should be fired*.

The reason for this is obvious enough; we cannot have two rules acting on the same merged report element since the second would simply overwrite the result of the first.

However, whether or not this condition is violated by a group of rules whose actions all deal with the same output report element cannot be determined simply by inspecting the logical form of their conditions. For example, two rules (whose actions deal with the same merged report element) with conditions of the form (1) p and q, and (2) p and r, may seem to violate this condition in that both sets of conditions could be

true, given a suitable set of input reports. However, it may be that, given the background knowledge in the knowledgebase, whenever q is true r is false, and vice versa, so that in fact there is no possibility of both rules firing.

It should be noted, however, that it is not the case that in such a grouping of rules (whose actions all concern the same output report element), that for any set of input reports, *at least one rule should be fired*. Rather, the conditions of a group of rules that belong together are formulated so that they can all *fail*. Intuitively, this is desirable behaviour. If none of the input reports has a textentry for the report element with which the conditions of the rules in the group are concerned (in this case the schema variable, if there is one, would be ground by a list of `null`s), we want no action to be taken, and so all the rules in the group should fail. For this reason negation-as-failure is in general not employed in formulating conditions for groups of rules (though of course it is frequently used with predicates in the Prolog knowledgebase), purpose-built predicates being defined instead. As an instance of this consider Example 3.6: if the set variable that is the argument to `EquivalentTerms` in rule 12 is ground with a list of textentries that are all `null`, then the predicate will fail. This means, of course, that were `NOT EquivalentTerms` to be used with the same set variable as an argument in rule 13, it would succeed, leading to the risk that if other conditions in the rule were also to succeed, a `null` textentry would be added to the merged report. As this is deemed undesirable, a new predicate, `NotEquivalentTerms` is defined that also fails if the set variable is ground with a list of textentries that are all `null`. Hence, if no input report contains a tag for `today` neither will the output report.

Example 3.6 *A pair of rules for merging weather reports dealing with the entry for today's weather. The rule code indicates that these have been registered as the twelfth and thirteenth rules in the rulebase. The status of both is optional (i.e. not foundational). Rule 12 says that if all the textentries for today's weather are from the same equivalence class, the preferred term from that class is selected and that text is added to the* `today` *tag in the merged report. If the first condition of rule 12 fails, then, provided that the values of the textentry variables for the report element* `today` *are not all* `null`*, the first condition of rule 13 will succeed. The second condition simply constructs the disjunction of each textentry for today's weather from the set of input reports (duplicates and nulls being removed), and the action adds the disjunction as the textentry for the branch* `weatherreport/today`.

`Rulecode is 12` $(\texttt{Status} = \texttt{optional})$

$\texttt{EquivalentTerms}(\texttt{//weatherreport/today})$
$\texttt{AND PreferredTerm}(\texttt{//weatherreport/today}, \texttt{X})$
$\texttt{IMPLIES AddText}(\texttt{X}, \texttt{weatherreport/today})$

`Rulecode is 13` $(\texttt{Status} = \texttt{optional})$

$\texttt{NotEquivalentTerms}(\texttt{//weatherreport/today})$
$\texttt{AND Disjunction}(\texttt{//weatherreport/today}, \texttt{X})$
$\texttt{IMPLIES AddText}(\texttt{X}, \texttt{weatherreport/today})$

Of course it is possible to group rules into subgroups in other ways. In particular, it may seem natural to consider some rules as belonging to a single subgroup if they share some of their conditions, or if they have conditions which stand in the relation of negation. One might want to group rules in this way because there is, as far as we can see, no reason why the actions of rules in such subgroups should have to be concerned with the same output report element.

In the simplest case, for example, there is no reason why we might not have two rules of the form: (1) If p succeeds then execute action A_1; (2) If $not\ p$ succeeds then execute action A_2, where A_1 and A_2 concern different output report elements. Of course, in these sort of rule groupings, if more than one rule fires for any given set of input reports, there is no danger of later rules in the grouping overwriting the same output

report element. However, it is presumably still the intention of the rule engineer that at most one rule in such groupings should succeed, and that if both actions were to be executed for a given input, the logical integrity of the merged report would be violated.

A problem in determining which rules belong to this kind of subgroup is deciding how many conditions the rules should share. Clearly they need not share all of them. The following seems a reasonable subgrouping: (1) If p succeeds and q succeeds, then execute action A_1; (2) If p succeeds and q fails, then execute action A_2; (3) If p fails and r succeeds, then execute action A_3; and (4) If p fails and r fails, then execute action A_4. However it is also clear that merely sharing one condition is not a sufficient condition for us to group some fusion rules together, for it is clear that in some cases such a condition is too weak. This is because of the use of logical variables in the formulation of rules. The same symbol, for example X, might occur in many different rules but with different denotations, with the result that, at least syntactically, the same condition can appear in rules that clearly do not belong together. For example, in a set of rules that merge conflicting textentries in weather reports by using weighted voting, the condition `weightedpopularsharing`$(\texttt{//weatherreport/source}, \texttt{X}, \texttt{Y})$ occurs in rules dealing with today's weather, pressure (direction of change), air visibility, and sunindex. It may be that such groupings can be delimited, instead, by taking into account the presence of schema variables indicating with which input report element the rules are concerned, though the widespread occurrence of variables denoting the source of a report (as in the condition just mentioned) makes this difficult.

3.5.4 Completeness of rules

Whichever way we choose to group rules into subgroups there is a further problem that needs to be avoided. Suppose we have two rules in a subgroup with conditions of the form (1) p and q, and (2) $not\ p$. In this case the first condition succeeds and at most one of the pair can fire, since if the first rule's conditions succeed the second rule's must fail, and vice versa. But if p succeeds and q fails neither rule will fire and the merged report may well be lacking information it ought to contain. For example, there may be a node in the merged report with no textentry. However, whether another rule should be added (with conditions p and $not\ q$) depends, once again, on the knowledgebase. If the knowledgebase is such that if p succeeds q must also succeed then the rules are satisfactory as they stand. It seems then, that there is a further condition on an adequately engineered set of rules:

Condition 2: For any input set of reports (excepting the possibility that all the schema variables are ground to null by the input reports), and for each subgroup of rules, *at least one rule should fire*.

However, whether or not this condition is violated by a group of rules cannot be determined simply by inspecting the logical form of their conditions because, as already mentioned, the knowledgebase also plays a crucial role.

3.5.5 How many conditions should a rule have?

In this subsection, and the two following, we consider some further factors that determine whether a set of rules is well-engineered, namely, the number of conditions in a rule and the number of rules in a rule set, and the order of actions within a rule and the order of rules in a rule set.

It is instructive to return to the case of the subgroup with two rules whose conditions were of the form: (1) p and q, and (2) $not\ p$ and r. This case might give rise to the question: if the knowledgebase makes it clear that whenever p succeeds q must also succeed, then why bother to add q as a further condition to the first rule? There is of course a considerable amount of leeway as to what gets put into the rules and what gets put into the knowledgebase. In Example 3.7 the predicate `PreferredTerm` selects the preferred term from amongst a list of input terms that have already been determined by the first condition to belong to the

same equivalence class. Provided that the equivalence class is properly represented in the knowledgebase, and provided that the textentries are not all null, it cannot fail. But it is clear that the computational work involved in selecting the preferred term could have been incorporated into the work done in determining whether or not the input textentries belong to the same equivalence class. If so, then the rules would have had the simpler form as in Example 3.8.

Example 3.7 *A pair of rules for merging weather reports dealing with the entry for today's weather. Rule 12 says that if all the textentries for today's weather are from the same equivalence class, the preferred term from that class is selected and that textentry is added to the* `today` *tag in the merged report. If the first condition of rule 12 fails, then, provided that the values of the textentry variables for the report element* `today` *are not all* `null`*, the condition of rule 13 will succeed. The action of rule 13 uses a function to add the disjunction of the input textentries as the textentry for the branch* `weatherreport/today` *in the merged report.*

```
Rulecode is 12 (Status = optional)

EquivalentTerms(//weatherreport/today)
AND PreferredTerm(//weatherreport/today, X)
IMPLIES AddText(X, weatherreport/today)

Rulecode is 13 (Status = optional)

NotEquivalentTerms(//weatherreport/today)
AND Disjunction(//weatherreport/today, X)
IMPLIES AddText(X, weatherreport/today)
```

As far as a rule's conditions are concerned, then, the rule engineer faces a choice between, on the one hand, making them as simple as possible and, on the other hand, making them more perspicuous or revealing of their function. In the case of Examples 3.7 and 3.8, although the latter is simpler (and perhaps makes clearer the logical relationship between the rules), the former surely makes it clearer how the information is to be merged.

Example 3.8 *A simplified version of the pair of rules from Example 3.7.*

```
Rulecode is 12 (Status = optional)

EquivalentTerms(//weatherreport/today, X)
IMPLIES AddText(X, weatherreport/today)

Rulecode is 13 (Status = optional)

NotEquivalentTerms(//weatherreport/today, X)
IMPLIES AddText(X, weatherreport/today)
```

3.5.6 How many rules should a rule set have?

As a limiting case, it would of course be possible to deal with any application by formulating a single rule, with a predicate name of, say, `merge`, taking as its arguments a set variable for each element in the input reports. The knowledgebase would then have to be marginally more complex, having the extra `merge` predicate in addition to those predicates dealing with individual input report elements. But there are serious disadvantages with this approach.

First, it would not be possible to tell, simply by inspecting the rule, how the different report elements were to be merged; that is, the rule itself would not specify how conflicts should be resolved — should conflicts between textentries for today's weather be settled by some form of voting, or by preferences over sources, etc. Instead, the reader would have to determine this by inspecting the knowledgebase.

Second, the rule itself would likely be difficult to understand because an action must be included for each output report element. Depending on the size of the ouput report, then, the single rule could be quite unwieldy.

Third, and most seriously, it is much easier to change the aggregation predicate involved in merging the textentries for a particular report element by editing a fusion rule than it is to recode part of the knowledgebase. Suppose, for example, that we wish to resolve conflicts for today's weather by using majority voting rather than first-past-the-post voting. Using a normal set of fusion rules all this requires, assuming the definitions for the predicates already exists in the knowledgebase, is the substitution of one predicate name for another in the appropriate fusion rule.[8] By contrast, even if the code for all the required aggregation predicates exists in the knowledgebase, if there were only one rule, the subsidiary predicate dealing with today's weather would have to be recoded so that it called a different helper aggregation predicate. Although such recoding is not difficult, editing a fusion rule is clearly much simpler.

What emerges from this are the advantages, where possible, of using more rather than less fusion rules to input data into a knowledgebase. First, and as was the case with the number of conditions in a rule, it makes it clearer than it would otherwise be what aggregation predicates are being used to merge conflicting information. (Though to be sure, inspection of the knowledgebase is still required to appreciate the exact definition being used.) And, second, given that the knowledgebase already contains a library of aggregation predicates, changes as to which predicates are to be used are more easily effected by editing fusion rules than by editing the knowledgebase itself.

Another factor that may determine the number of fusion rules is computational efficiency. Example 3.9 is from a case study in bioinformatics where the objective is to find subsets of a set of protein domains that exhibit varying degrees of functional similarity. This similarity is to be determined on the basis of an ontology of molecular functions that had been incorporated into the knowledgebase. Because this ontology is comprised of a class hierarchy of molecular functions, finding functionally similar subgroups is achieved on the basis of locating subgroups whose least upper bound (that function in the hierarchy that is general enough to include all the functions in a subgroup, but that is no more general than necessary) is deemed sufficiently specific. In addition to finding these functional subgroups, the rules were also to produce a list of the key words and phrases belonging to the functional annotations true of all of the proteins in the subgroups. There were, then, two distinct tasks for the rules, and so it would be natural to assign each task to a separate rule. However, because finding the keywords and phrases for a subgroup is a matter of finding the keywords and phrases that belong to the annotations for the least upper bound, and to all of its superclass molecular functions, producing this information in a separate rule would have meant in large part repeating the computations required by the first rule. Hence, for reasons of efficiency at runtime, the two tasks were combined into a single rule.

Looking at the matter in general terms, then, there seem to be competing considerations as to how many rules a rule set would require. But given a particular application — a particular set of input reports — can we say with any confidence how many rules are required of a well-engineered set? A crude way of estimating this would be to make the following three assumptions: (1) That we are interested in every element in the input reports (that is, none of the information is irrelevant to our interests). (2) That in determining the structured report to be output, the information contained in each input report element is to be treated independently of the information contained in the other input elements. And (3), that we will need a pair of rules to deal with each input report element, one specifying what action to take if a certain condition holds, the other specifying what action to take if the condition fails. If these three

[8] We have implemented in Java a fusion rule editing tool, with a graphical user interface, that would allow changes such as this to be made simply by selecting a different predicate name from a dialog combo box. See www.cs.ucl.ac.uk/staff/a.hunter/frt for details.

assumptions hold, then we can say that the total number of rules will be roughly twice the number of input report elements. And, indeed, in both the bioinformatics and the weather reports case studies that we have undertaken, this is what we find. In the former case mentioned in Example 3.9, the input reports concerning individual protein domains had in effect only one attribute, the functional annotation accession number for that domain (see Example 3.10); as a result, the rule set for merging a group of protein domain reports contained only two rules: one to deal with the case where the domains were similar enough to be merged, and the second to deal with the case where they were not.

Example 3.9 *A rule from a bioinformatics case study that selects from a family of protein domains subgroups that are functionally similar. The first condition finds a list of the subgroups that are functionally similar, and uses it to bind the logical variable* X*. It also counts the number of such subgroups (binding the variable* N*), so that the first action knows how many* functionalgroup *elements to add to the merged report. The second condition finds the least upper bound for each subgroup (this is the function that is general enough to include all of the functions of the protein domains in the subgroup, but that is no more general than necessary) and uses the list of these common functions to bind the variable* Y*. The third condition exploits this list of common functions to find the key words and phrases from amongst those functional annotations that apply to all protein domains in the subgroup — i.e. from amongst the annotations of the least upper bound and its superclasses in the Gene Ontology class hierarchy.*

Rulecode is 1 (Status = foundational)

SelectFunctionalGroups(//protein/function, //protein/name, X, N)
AND GetLeastUpperBounds(X, Y)
AND GetKeywords(Y, Z)
IMPLIES Initialize(biofusionanalysis)
AND RepeatAddNode(functionalgroup, N, biofusionanalysis)
AND RepeatAddNode(selectedproteins, 1, biofusionanalysis/functionalgroup)
AND RepeatAddText(X, biofusionanalysis/functionalgroup/selectedproteins)
AND RepeatAddNode(commonfunction, 1, biofusionanalysis/functionalgroup)
AND RepeatAddText(Y, biofusionanalysis/functionalgroup/commonfunction)
AND RepeatAddNode(keywords, 1, biofusionanalysis/functionalgroup)
AND RepeatAddAtomicTrees(Z, keyword, biofusionanalysis/functionalgroup/keywords)

The RepeatAddNode(Tagname, Number, Branch) *predicate specifies the action that the branch given by* Branch *is extended by a number of children, each of which has the tagname given by* Tagname*, and the number of these children is given by* Number*. The* RepeatAddText(List, Branch) *predicate specifies the action that the branch given by* Branch *is extended by a number of children where there is one child per textentry in* List*. The* RepeatAddAtomicTree(ListofLists, Tagname, Branch) *predicate specifies the action that for the ith occurrence of the branch given by* Branch *is extended by the ith list of textentries in the list of lists of textentries in* ListofLists *as follows: For each occurrence of a branch of the form* Branch*, the jth extension of that occurrence of the branch is a child node with tagname given by* Tagname *followed by a leaf node and the leaf node is the jth textentry in the ith list of textentries (in* ListofLists*).*

Example 3.10 *An example of an individual protein report that could serve as part of the input for the rule in example 3.9. The* name *tag serves to identify the entity being merged, a protein domain. The* function *tag serves to express the single attribute of this entity, its molecular function.*

⟨protein⟩
⟨name⟩ 1c4tC0 ⟨/name⟩
⟨function⟩ GO : 0004149 ⟨/function⟩
⟨/protein⟩

In the weather reports case study, by contrast, the input reports contained eighteen separate elements (see Figure 2 and Example 3.11). The elements for source, location, and date in effect identify the individual

⟨!ELEMENT weatherreport(source, date, city, today, temp,
windspeed, relativehumidity, daylighthours,
pressure, visibility, visibilitydistance,
airpollutionindex, sunindex, sunindexrating,
dewpoint)⟩
⟨!ELEMENT temp(max, min)⟩
⟨!ELEMENT daylighthours(sunrise, sunset)⟩
⟨!ELEMENT pressure(absolutevalue, directionofchange)⟩
⟨!ELEMENT α(#PCDATA)⟩

Figure 2: A condensed version of the DTD for the weather reports where α is instantiated with any of source, date, city, today, max, min, windspeed, relativehumidity, sunrise, sunset, absolutevalue, directionofchange, visibility, airpollutionindex, sunindex, and dewpoint.

reports, and so were combined into a single, foundational rule. The remaining fifteen elements were merged independently of each other, and so required two rules each. Thus a total of thirty one rules were required for this application, and that number remained the same no matter which of around a dozen different aggregation functions were used.

Example 3.11 *An example of an individual weather report from the weather reports case study that conforms to the DTD in figure 2.*

⟨weatherreport⟩
⟨source⟩ BBCi ⟨/source⟩
⟨date⟩ 10/12/02 ⟨/date⟩
⟨city⟩ London ⟨/city⟩
⟨today⟩ cloudy ⟨/today⟩
⟨temp⟩
⟨max⟩ 3C ⟨/max⟩
⟨min⟩ − 1C ⟨/min⟩
⟨/temp⟩
⟨windspeed⟩ 11mph ⟨/windspeed⟩
⟨relativehumidity⟩ 63% ⟨/relativehumidity⟩
⟨daylighthours⟩
⟨sunrise⟩ 7.13 ⟨/sunrise⟩
⟨sunset⟩ 15.51 ⟨/sunset⟩
⟨/daylighthours⟩
⟨pressure⟩
⟨absolutevalue⟩ 1021mb ⟨/absolutevalue⟩
⟨directionofchange⟩ rising ⟨/directionochange⟩
⟨/pressure⟩
⟨visibility⟩ good ⟨/visibility⟩
⟨airpollutionindex⟩ 3 ⟨/airpollutionindex⟩
⟨sunindex⟩ 1 ⟨/sunindex⟩
⟨dewpoint⟩ 2C ⟨/dewpoint⟩
⟨/weatherreport⟩

But in some cases these assumptions do not hold. The fact is that what determines the number of rules for an application is simply what we want to *do* with the information in the input reports. There may be cases where, despite the fact that the input reports contain a lot of elements that describe the attributes of the report entity, the purpose of the application is to use all of those attributes together to do one thing. In cases such as this, the number of fusion rules required will be small.

An example of a case study with a small number of fusion rules is an application for merging information from individual reports containing clinical information about patients being treated for breast cancer (see Example 3.12). Besides a unique identifier, each report contained twenty six elements recording various

clinical attributes of each patient. Yet the set of rules to deal with these reports contained only two rules. This was because the purpose of "merging" a set of such reports was to test the hypothesis that three of these attributes could be used, when combined in various ways, as new indicators of a patient's prognosis. This test was to be done by using six of the other attributes to classify each patient's prognosis as "Best" or "Worst" according to recognized criteria, and then to use the three further attributes to arrive at an additional classification, so that the two classifications could be compared. Doing this required using all of this information from each patient to produce a single result — could three of the attributes be used to provide a reliable indicator of the patient's prognosis? Hence, only a single rule was required.

Example 3.12 *An example of an individual patient report of the type used in a clinical case study. Fictional data has been substituted.*

```
⟨patient⟩
        ⟨id⟩ B05 − 5566; ⟨/id⟩
        ⟨type⟩ Lob ⟨/type⟩
        ⟨size⟩ 28mm⟨/size⟩
        ⟨grade⟩ 2⟨/grade⟩
        ⟨LVI⟩ Yes⟨/LVI⟩
        ⟨Ln⟩ 0/14⟨/Ln⟩
        ⟨npi⟩ 5.8⟨/npi⟩
        ⟨ER⟩
                ⟨positive⟩ 3 ⟨/positive⟩
                ⟨total⟩ 105 ⟨/total⟩
                ⟨percent⟩ 2.86% ⟨/percent⟩
        ⟨/ER⟩
        ⟨PR⟩
                ⟨positive⟩ 78 ⟨/positive⟩
                ⟨total⟩ 152 ⟨/total⟩
                ⟨percent⟩ 51.31% ⟨/percent⟩
        ⟨/PR⟩
        ⟨Ki67⟩
                ⟨positive⟩ 410 ⟨/positive⟩
                ⟨total⟩ 480 ⟨/total⟩
                ⟨percent⟩ 85.41% ⟨/percent⟩
        ⟨/Ki67⟩
        ⟨Mcm2⟩
                ⟨positive⟩ 300 ⟨/positive⟩
                ⟨total⟩ 300 ⟨/total⟩
                ⟨percent⟩ 100% ⟨/percent⟩
        ⟨/Mcm2⟩
        ⟨G94⟩
                ⟨positive⟩ 38 ⟨/positive⟩
                ⟨total⟩ 294 ⟨/total⟩
                ⟨percent⟩ 12.92% ⟨/percent⟩
        ⟨/G94⟩
        ⟨G95⟩
                ⟨positive⟩ 93 ⟨/positive⟩
                ⟨total⟩ 510 ⟨/total⟩
                ⟨percent⟩ 18.23% ⟨/percent⟩
        ⟨/G95⟩
        ⟨HER2⟩ − ⟨/HER2⟩
        ⟨history⟩ chemo ⟨/history⟩
⟨/patient⟩
```

3.5.7 Rule order and action order

Rules are executed by the action engine in the order in which they occur in the rule set. The fact that the output report takes the form of a tree places some constraints on the order of the rules. In particular, all rules with Initialize actions must precede all rules lacking such actions so that there is a tree of some sort to which those other actions can add a text entry or a subtree.

But more generally, if a particular action adds a text entry or a subtree to a target node in the output report,

then the ordering of the rules must be such as to ensure that the target node is already present in the output report as so far constructed. Provided, however, that this condition is met, there is little significance to the ordering of the rules beyond the ordering of elements which the rule engineer seeks in the output report; if, for example, we want the entry for `temperature` in the output report to precede that for `humidity`, then the rules should occur in a similar order.

Moreover, within a particular subgroup of rules (that is, a group of rules all of whose actions deal with the same output report element) there also seems little significance to the order of the rules: for example, although it may be natural for a pair of rules whose condition literals are `similartemperatures` and `notsimilartemperatures` to occur in that order, there is no reason, as far as their conditions are concerned, why the order should not be reversed. What is important is not their order but that, for any set of input reports, the conditions and the knowledgebase are such that at most one rule in this pair should be fired. In fact, as far as the production of a correct output report is concerned, it is not even essential that all the rules in a particular subgroup occur *consecutively*, though it is obviously confusing to their human reader if the two rules concerned with temperatures are separated by a pair of rules dealing with humidity.

In general, then, there is a good deal of flexibility, both in the ordering *between* subgroups of rules in a rule set, and in the ordering *within* subgroups of rules.

The case is somewhat different for actions. In most cases, though not all, the order of actions in a single rule is significant. Where the order is significant this is for the same reason that the order of rules can be important: if an action adds a text entry or a subtree to a target node, then that target must already be in the output report tree. That is why, for example, if a rule contains an Initialize action, that action must be the first action of the rule. The reason why order is more often important for actions than for rules is simply that when a rule has more than one action, generally (though not always), that will be because one action will indeed be adding a new node to a report that will be the target node for other actions. A good example of this is the rule from the bioinformatics case study given in Example 3.9. In this rule, four `RepeatAddNode` actions add nodes to the output report that are targets for subsequent `RepeatAddNode`, `RepeatAddText` or `RepeatAddAtomicTrees` actions. Each `RepeatAddNode` action must thus precede the `RepeatAddNode`, `RepeatAddText` or `RepeatAddAtomicTrees` action of which it provides the target node or nodes. The order of these actions is not, however, absolute: after the `Initialize` action, all of the `RepeatAddNode` actions could have preceded all of the other actions.

3.5.8 Algorithm for grouping rules into subgroups

Having investigated some of the conditions required of a well-engineered set of rules, we turn to an investigation of what help in the form of automated support can be given to the rule engineer to check if these conditions are satisfied.

We have shown that if a set of rules is to have the correct logical behaviour with respect to a set of input reports then that set must first be decomposed into disjoint subsets — subsets whose rules have to work together such that they conform to Conditions 1 and 2.

The algorithm for separating a set of rules into subgroups is as follows (see Figure 3). While the set of rules contains more than one rule do the following: remove the first rule from that set and add it to a new subgroup. Extract the merged report node with which the last action of that rule is concerned to use as a test. Then, loop over the remaining rules in the rule set. Extract the node from the last action of each rule and compare that node with the node extracted from the rule in the new subgroup. If they are the same, then delete that rule from the rule set, and add to the new subgroup. When comparisons have been made with all of the remaining rules, check the size of the new subgroup; if it is greater than one, add it to the list of subgroups.

This algorithm delimits a set of rules into subgroups on the basis of the rules' actions being concerned

```
while |Rules| > 1
    let Rule ∈ Rules
    let Rules := Rules \ Rule
    let SGRules := {Rule}
    let actionNode := node from last action of Rule
    for i := 0, until i < |Rules|
        let nextNode := node from last action of Rules[i]
        if actionNode = nextNode
            let Rules := Rules \ Rules[i]
            let SGRules := SGRules ∪ {Rules[i]}
            i := i − 1
        i := i + 1
    if |SGRules| > 1
        let GRules := GRules ∪ {SGRules}
```

Figure 3: The algorithm for grouping a set of rules into subgroups, where those subgroups are rules that deal with the same output report element. Let Rules be the set of rules and let GRules be the set of subgroups of rules that deal with the same output report element; let SGRules be the new subgroup of rules dealing with a particular output report element; let Rule be a rule remaining in the list of rules, Rules; and let actionNode be the output report node with which the last action of a given rule is concerned. Note that how this node is extracted from a given action depends on the kind of action. For example with `AddText`, it will be the node in the merged report to which the text is to be added. With `AddAtomicTree` it will be the node that is to be added to the merged report.

with the same output report element. In determining, in turn, with which output report element a rule is concerned we must look at the rule's *actions*. But often rules have more than one action, and those actions, of course, are concerned with different output report elements. For example, one action might add a node to the merged report tree (say `daylighthours`), and then a second will add an atomic tree to that node (say, the node `sunrise`, together with a textentry specifying the time). This means that in determining whether a rule belongs to a particular subgroup a decision has to be made as to which action of a rule (if there is more than one) to consider. It seems reasonable to suppose that it is the *last* action of a rule that indicates with which report element it is concerned. This is because actions prior to the last action in a rule tend to be concerned with setting up the merged report tree so that the textentry or subtree can be added at the right point. In the case of the rules dealing with the hours of sunrise and sunset, for example, the nodes containing this information have to be added to the node `daylighthours`, but the action that adds that node should not be used in determining with which output report element these rules are concerned. However, checking only the last action in a rule in determining to which subgroup the rules belongs is a risky assumption.[9]

Having grouped the rules into subgroups in this way it is now possible to test if they conform to condition 1. This is done by making pairwise comparisons for inconsistency, something which is done by checking for the occurrence of the string "not", either as a separate word, or as a prefix, together with the presence of the same condition name. However, as we have already noted, instead of being indicated by the string "not", the inconsistency of two rules may be guaranteed, rather, by the structure of the knowledge itself in a particular domain; for example, if one rule has condition *p* and a second condition *q*, it may be that whenever *p* succeeds, *q* fails. For this reason, the automated approach given here may flag some subgroups as not being pairwise inconsistent (as failing Condition 1) when in fact they are pairwise inconsistent, and

[9] In fact, we have already seen a case where this assumption does not hold. (See example 3.9) Here, it is not the case that only the last action of the rule is concerned with adding the results of some merging operation to the output report; rather, several actions in this single rule are concerned with adding the results of different merging operations to different parts of the output report. However, this does not show that the grouping algorithm is useless; it merely shows that we have no need for it in cases where the number of rules in the rule set is very small (in this example there were only two).

```
for i := 0, until i < no. of elements in Elements
        Flags[i] := false
        ruleLoop : for j := 0, until j < no. of rules in Rules
                for k := 0, until k < no. of actions in Actions
                        for m := 0, until m < no. of constants in Constants
                                if last segment of Constants[m] = Elements[i]
                                        Flags[i] := true
                                        break ruleLoop
                                m := m + 1
                        k := k + 1
                j := j + 1
        i := i + 1

for all i, if Flags[i] = true, Rules are minimally adequate.
```

Figure 4: The algorithm for determining if a set of rules meets the minimum adequacy condition for coverage of output report elements. Rules is an array containing the set of rules being tested; Elements is the list of elements in the merged report that have textentries as their children; Flags is the array of Booleans flagging if each element in Elements is covered by at least one action; Actions is an array of the actions belonging to the rule being checked; and Constants is an array of the report branches which are the constants belonging to the action being checked. The list Elements is extracted from the DTD for the merged output report.

so do conform to the condition. That is, the check may result in some false negatives.

3.5.9 Rule coverage

Although we cannot fully automate the process of checking whether a set of rules meets the two conditions outlined above, we can formulate a much weaker, minimal, adequacy condition or lower bound on a set of rules covering an output report, a condition that can be checked automatically:

Condition 3: A set of rules is minimally adequate with respect to coverage if it contains at least one action that deals with each element in the *output* report.

It is coverage of output report elements that determines whether a set of rules is minimally adequate in this sense because: (1) in merging a set of reports not every input report element may be relevant to producing an output, merged report. It may be that for some applications there are some input report elements with which we are simply not interested. And (2) the element names of the output report may be different to those of the input reports.

The algorithm (see Figure 4) is used to check if a set of rules is minimally adequate with respect to coverage of output report elements: Using the DTD for the merged (output) report, extract a list of those report elements that have textentries as children. Create an array of Boolean flags, initialized to false, of the same size as this list. For each element in the list, check the rules in the rule set as follows; if the current rule has an action that contains a *constant* with a report branch whose last segment matches the current element in the list, set the corresponding Boolean flag to true, and proceed to check the next element in the list. If at the end of checking the list of elements, the array of flags are all true, then the set of rules is minimally adequate. If not, then in combination with the list, the flags which are false indicate which output report elements have not been covered by any rule.

Note that it is *action constants* that are used to check for rule coverage because branches in schema variables

in actions will be referring to the elements in the input reports from which textentries are to be extracted, rather than to the elements in output reports to which the results of merging are to be added. And in any case, even when the elements in the input reports do match those in the output report (which often they will do), in many cases schema variables are replaced in rules by logical variables from which, of course, it cannot be determined which report elements they are concerned with.

Example 3.13 *Consider the following simple DTD for an output report that is to be produced by merging weather reports.*

<!ELEMENT weatherreport (source, date, city, today, temp) >
<!ELEMENT temp (max, min) >
<!ELEMENT source (#PCDATA) >
<!ELEMENT date (#PCDATA) >
<!ELEMENT city (#PCDATA) >
<!ELEMENT today (#PCDATA) >
<!ELEMENT max (#PCDATA) >
<!ELEMENT min (#PCDATA) >

A set of rules that met the minimal adequacy condition for coverage of output report elements would have to contain at least six actions, since the DTD indicates that the output report must have six elements that have textentries: source, date, city, today, max, *and* min. *Each of the six actions would have to contain a constant that matched one of these elements.*

An alternative way of considering the coverage of a set of rules is with respect to the input reports rather than the output report. A set of rules could be considered to meet an alternative minimum adequacy condition for coverage if it contains at least one condition dealing with each element in the input reports. In this case, the test for meeting such an adequacy condition would be that for every complete branch in the input reports (i.e. a branch terminating in a textentry), the rule set contains at least one condition with a schema variable with a matching branch. Obviously, the greater the number of elements in the input reports, the greater the number of conditions required of a minimally adequate set of rules. And, given that we want to merge the elements in the input reports independently of one another (something which is not always true — see the breast cancer case study above), the greater the number of rules that will be required of an adequate rule set. It should be emphasized that this adequacy condition only applies on the assumption that we are in fact interested in all of the information contained in the input reports (again, something that is not true in the case of the breast cancer case study). But, provided that is so, then we could combine both of these conditions to produce a more stringent condition on rule coverage.

Condition 4: A set of fusion rules is minimally adequate with respect to coverage if for each leaf element in the input reports, it contains at least one condition that deals with that element, and for each leaf element in the output report it contains at least one action that deals with that element.

3.5.10 Checking the structure of the output report

A set of rules that are minimally adequate with respect to coverage of input and/or output report elements may well have serious shortcomings in other respects. One is that the actions in a set of rules may not in fact build an output report with a coherent tree structure. This could simply be because a target node, to which a new node or textentry is to be added, has not yet have been added to the tree because the rules are in the wrong order; or, more seriously, it might be because the target node is completely absent from the tree, indicating that a rule or an action is missing.

To check that the rules, as ordered, do in fact construct an output report with a correct tree structure, an algorithm can be used to reconstruct the tree, if any, that the rules would assemble if executed (see Figure

```
for i := 0, until i < no. of distinct Initialize atoms in Rules
    construct the initial tree, T, according to InitializeAtoms[i]
    for j := 0, until j < no. of rules without Initialize atoms in Rules
        for k := 0, until k < no. of actions in Actions
            extract targetBranch from the target constant in Actions[k]
            boolean match := false
            for m := 0, until m < no. of branches in T
                if targetBranch = Branches[m]
                    add any new nodes added by Actions[k] to T
                    boolean match := true
                    break
                m := m + 1
            if match = false
                generate appropriate error message
            k := k + 1
        j := j + 1
    i := i + 1
```

Figure 5: The algorithm for reconstructing the tree (or trees), if any, that a set of rules would construct if it were executed. Rules is a array containing the set of rules being tested; InitializeAtoms is an array of the distinct `Initialize` atoms in Rules; Actions is an array of the actions belonging to the rule being checked; Branches is an array of the branches in the tree, T, constructed by the rules in Rules checked so far; targetBranch is the branch constant belonging to the action being checked which represents the branch in the merged report to which are to be added any new nodes or textentries specified by the action; and the Boolean match flags if targetBranch matches a branch in T.

5). First, we must take account of the fact that a set of rules may contain more than one action with an `Initialize` atom, and those atoms may be distinct in that they would lead to the construction of merged reports with different tree structures. So, for each distinct `Initialize` atom in the rule set, construct the initial tree according to that atom. Then, check each action of each rule in the rule set, except rules with other Initialize atoms, by extracting the constant which specifies the target branch for that action. The target branch constant is the constant that specifies the branch in the merged report to which a textentry or node is to be added by the action (as opposed to a constant which specifies the node *being added*). If the target branch extracted matches a branch in the tree constructed so far, then add to the tree any new nodes added by this action. Otherwise, generate an error message.

Viewing the output of this algorithm will allow the rule engineer to see if her rules construct a coherent tree. But, by itself, this tree will not allow her to see if the rules will produce the tree that she *intends* them to construct. To see if they do that, a tree should also be constructed from the original merged report DTD, and the two trees compared.

3.6 The relationship between fusion rules and a knowledgebase

As has already been mentioned (Section 3.5), fusion rules are an overarching mechanism for controlling knowledgebased merging in that they are both a way of querying a knowledgebase and a way of compiling the results of those queries into a merged report. Looked at in this way fusion rules can be thought of as an interface between the input and output reports, on the one hand, and the knowledgebase itself, on the other. However, the relationship between a set of fusion rules and a knowledgebase is in fact more intimate than this way of considering fusion rules would suggest. This is because fusion rules can embody or contain knowledge about a domain of application, and in such cases they are thus themselves also a *part* of the

knowledge for merging. For example, a rule merging reports for today's weather may deal with conflict by using a voting function with a threshold, so that only if, say, at least 75% of input reports agree on a value will it be included in the output report. This threshold may be specified in the resource queried by the fusion rule, but it may instead be explicitly specified (as a constant) in the fusion rule itself. Or consider another example: a fusion rule merging textentries for temperature may deem them to be not conflicting if they are all within a certain acceptable range, say 3C. Again, in this case the knowledge as to what constitutes an acceptable range of temperatures could be specified in the resource being queried, but it could equally be explicitly specified in the fusion rule itself. (see example 3.14)

Example 3.14 *A rule for merging the textentries for the maximum temperature as specified by a set of weather reports. If the temperatures are deemed to be within an acceptable range then the logical variable,* X*, is bound by a string specifying the interval from the smallest to the largest values (or just a single value if they are all the same), and this interval is added as the textentry for the maximum temperature in the merged report. However, given this way of writing the rule, the user of this application would have to inspect the resource being queried in order to find out what that acceptable range is.*

```
Rulecode is 8 (Status = optional)

SimilarTemps(//weatherreport/temp/max, X)
IMPLIES AddText(X, weatherreport/temp/max)
```

The same rule rewritten so as explicitly to display the accepted range within which values for the maximum temperature are deemed not to conflict. This range, 3C, is specified in the rule by the addition of a constant. By checking the values in the input reports, the user of this rule can now easily see why the rule succeeds or fails when the rules are executed.

```
Rulecode is 8 (Status = optional)

SimilarTemps(//weatherreport/temp/max, 3C, X)
IMPLIES AddText(X, weatherreport/temp/max)
```

For a given application domain there is thus some leeway as to what knowledge is included in the resource queried and what is included in the fusion rules, and for this reason it is better to consider both as constituting the knowledge for that application. Moreover, because one of the virtues of fusion rules is that they make it clearer why the reports are being merged as they are, in general it is to be preferred that simple knowledge, such as acceptable ranges or voting thresholds, should be explicitly included in the fusion rules rather than buried in the resource being queried.

3.7 Viability of knowledgebased merging using fusion rules

How easy is it, then, to develop knowledgebase merging for an application? Obviously any approach to merging information using a knowledgebase requires getting to grips with the background knowledge, but no specific problems arise in creating a knowledgebase to be used in conjunction with fusion rules. Admittedly, in writing the fusion rules one must make sure that the order of the schema or logical variables in the conditions matches the order of the arguments in the corresponding knowledgebase predicates, but even if one were simply querying the knowledgebase in a standalone fashion one must get the order of the arguments correct anyway.

In practice, the hardest and most time-consuming task in developing an application is the writing of the clauses in the knowledgebase, and not the formulation of the fusion rules. The role of the fusion rules is in large part simply to connect the input reports to the output merged report, by selecting textentries or

subtrees from the former and adding the results of merging these to the latter. As a result, the trickiest part in formulating a set of fusion rules is simply making sure that they respect the structure of the input and output reports; specifically, making sure that the conditions (and occasionally the actions) extract information from the correct branches in the input reports, and that the actions add the merged results to the correct branches in the output reports. It is easy to specify the wrong branch as part of a schema variable or constant, and then things will go awry. But, as indicated in the previous sections, some automated support for this is possible, and indeed all of the support outlined above has been implemented in a rule engineering tool that we have developed.

There are some simple conditions that must be met if a set of fusion rules is to be well engineered, but whether they are met can only be determined by inspecting the knowledgebase as well.

A key advantage of knowledgebased merging with fusion rules is the robustness to heterogeneity in the information being merged. Heterogeneity comes in a variety of forms in the information to be merged including:

Ontological heterogeneity Different textentries are used by different sources for the same concept and/or the same textentry is used by different sources for different concepts.

Epistemic heterogeneity Different sources provide different beliefs, and these beliefs may conflict, and/or different sources may attach different uncertainty evaluations to beliefs.

Structural heterogeneity Different subtree structures are used by different sources to represent the same information and/or the same subtree structure may be used by different sources for different information.

A knowledgebased merging approach has significant advantages over other approaches to merging in that increasing relevant knowledge can increase robustness. In this case, relevant knowledge is knowledge that specifies when textentries/subtrees are equivalent or not equivalent. In general, such equivalence information can be more efficiently represented in quantified logical formulae than in other approaches. This relevant knowledge can be used to address ontological, epistemic, and structural, heterogeneity, though except in very limited applications, it is unlikely to address all cases of such heterogeneity. In addition to this relevant knowledge about equivalences, deeper knowledge about time/events, uncertainty, inconsistency, etc., as discussed in Section 3.2, can provide an important support role in deciding equivalence/non-equivalence.

What emerges from the discussion are the advantages of using fusion rules to input data into a knowledgebase. First, it makes it clearer than it would otherwise be what aggregation predicates are being used to merge conflicting information. (Though to be sure, inspection of the knowledgebase is still required to appreciate the exact nature of the concepts/definitions being used.) And, second, given that the knowledgebase already contains a library of aggregation predicates, changes as to which predicates are to be used are more easily effected by editing fusion rules than by editing the knowledgebase itself. Using a fusion rule system may not make it any easier to construct a knowledgebase in the first place, but it does make the knowledgebase easier to use. And, fusion rules themselves are not hard to write in general.

In order to demonstrate the viability of knowledgebased merging using fusion rules, we have provided extensive coverage of two case studies (in weather and in bioinformatics) on our website[10]. This includes a number of input and output reports, together with knowledgebases and sets of fusion rules. The website also has downloadable versions of the FusionTool, which incorporates the FusionEngine and RuleEngine, and the RuleTool, for engineering sets of fusion rules.

[10] Fusion Rule Technology Website: www.cs.ucl.ac.uk/staff/a.hunter/frt

4 Conclusions

We summarise the advantages of a knowledgebased approach to merging based on fusion rules as follows: (1) Structured reports to be merged can be treated as logical terms in the knowledgebase and so the information in the reports to be merged can be reasoned with in the knowledgebase; (2) Merged reports can be obtained by logical inference from the knowledgebase; (3) Merging based on fusion rules supports context-sensitive integration of information based on the contents of the structured reports to be merged and on the background knowledge; (4) The type of fusion sanctioned by the fusion rules can depend on the coherence of the input and the reliability and disposition of the sources; (5) The knowledgebase used for merging can incorporate specific and generic knowledge which can be based on rich theories for knowledge representation and reasoning for handling concepts such as time, events, space, and uncertainty; (6) The knowledgebase used for merging can incorporate ontological knowledgebases such as knowledgebases defined in a description logic; (7) Using the combination of a knowledgebase and fusion rules for merging offers some robustness to ontological heterogeneity, epistemic heterogeneity, and structural heterogeneity, in the structured reports to be merged; and (8) Fusion rules give the provenance of the merged information and so can be treated as meta-data or meta-knowledge associated with the merged structured report.

Our logic-based approach differs from other logic-based approaches for handling inconsistent information such as belief revision theory (e.g. [Gar88, DP98, KM91, LS98]) and knowledgebase merging (e.g. [KP98, BKMS92]). These proposals are too simplistic in certain respects for handling structured reports. Each of them has one or more of the following weaknesses: (1) One-dimensional preference ordering over sources of information — for structured reports we require finer-grained preference orderings; (2) Primacy of updates in belief revision — for structured reports, the newest reports are not necessarily the best reports; and (3) Weak merging based on a meet operator — this causes unnecessary loss of information. Furthermore, none of these proposals incorporate actions on inconsistency or context-dependent rules specifying the information that is to be incorporated in the merged information, nor do they offer a route for specifying how merged reports should be composed.

Other logic-based approaches to fusion of knowledge include the KRAFT system and the use of Belnap's four-valued logic. The KRAFT system uses constraints to check whether information from heterogeneous sources can be merged [PHG$^+$99, HG00]. If knowledge satisfies the constraints, then the knowledge can be used. Failure to satisfy a constraint can be viewed as an inconsistency, but there are no actions on inconsistency. In contrast, Belnap's four-valued logic uses the values "true", "false", "unknown" and "inconsistent" to label logical combinations of information (see for example [LSS00]). However, this approach does not provide actions in case of inconsistency.

Merging information is also an important topic in database systems. A number of proposals have been made for approaches based in schema integration (e.g. [PM98]), the use of global schema (e.g. [GM99]), and conceptual modelling for information integration based on description logics [CGL$^+$98b, CGL$^+$98a, FS99, PSB$^+$99, BCVB01]. These differ from our approach in that they do not seek an automated approach that uses domain knowledge for identifying and acting on inconsistencies. Heterogeneous and federated database systems are relevant, but they do not identify and act on inconsistency in a context-sensitive way [SL90, Mot96, CM01], though there is increasing interest in bringing domain knowledge into the process (e.g. [Cho98, SO99]). Also relevant is revision programming, a logic-based framework for describing and enforcing database constraints [MT98].

Our approach also goes beyond other technologies for handling heterogeneous information. The approach of wrappers offers a practical way of defining how heterogeneous information can be merged (see for example [HGNY97, Coh98, SA99]) and NLP/NLG offer some techniques for identifying information in input reports and generating a natural language summary [BME99]. However, in these approaches there is little consideration of problems of conflicts arising between sources. Our approach therefore goes beyond these in terms of formalizing reasoning with inconsistent information and using this to analyse the nature of the news report and for formalizing how we can act on inconsistency.

References

[ABC99] M Arenas, L Bertossi, and J Chomicki. Consistent query answers in inconsistent databases. In *In Proceedings ACM PODS*, pages 68–79. ACM Press, 1999.

[AKS96] Y Arens, C A Knoblock, and W M Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2):99–130, 1996.

[BB03] L Bravo and L Bertossi. Logic programs for consistently querying data integration systems. In *Proceedings of the International Joint Conference on AI (IJCAI'03)*, pages 10–15. 2003.

[BBB+98] P G Baker, A Brass, S Bechhofer, C Goble, N Paton, and R Stevens. Tambis: Transparent access to multiple bioinformatics information sources. an overview. In *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology (ISMB'98)*, pages 25–34. AAAI Press, 1998.

[BC03] L Bertossi and J Chomicki. Query answering in inconsistent databases. In G. Saake J. Chomicki and R. van der Meyden, editors, *Logics for Emerging Applications of Databases*. Springer, 2003.

[BCVB01] S Bergamaschi, S Castano, M Vincini, and D Beneventano. Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering*, 36:215–249, 2001.

[BGP92] D Barbara, H Garcia-Molina, and D Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4:487–502, 1992.

[BKMS92] C Baral, S Kraus, J Minker, and V Subrahmanian. Combining knowledgebases consisting of first-order theories. *Computational Intelligence*, 8:45–71, 1992.

[BKY+99] Greg Barish, Craig A Knoblock, Yi-Shin Chen, Steven Minton, Andrew Philpot, and Cyrus Shahabi. Theaterloc: A case study in information integration. In *Information Integration Workshop, Stockholm, Sweden (IJCAI'99)*, 1999.

[BL99] T Berners-Lee. *Weaving the Web*. Orion Business Books, 1999.

[BLHL01] T Berners-Lee, J Hendler, and O Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[BME99] R Barzilay, K McKeown, and M Elhadad. Information fusion in the context of multi-document summarization. In *Proceedings of 37th Annual Meeting of the Association of Computational Linguistics*, 1999.

[Bra01] I Bratko. *Prolog: Programming for Artificial Intelligence*. Addison Wesley, 2001.

[CG04] L Chovy and C Garion. Querying several inconsistent databases. *Journal of Applied Non-classical Logic*, 14(3):295-328, 2004.

[CGL+98a] D Calvanese, G De Giacomo, M Lenzerini, D Nardi, and R Rosati. Description logic framework for information integration. In *Proceedings of the 6th Conference on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13. Morgan Kaufmann, 1998.

[CGL+98b] D Calvanese, G De Giacomo, M Lenzerini, D Nardi, and R Rosati. Source integration in data warehousing. In *Proceedings of the 9th International Workshop on Database and Expert Systems (DEXA'98)*, pages 192–197. IEEE Computer Society Press, 1998.

[Cho98] L Cholvy. Reasoning with data provided by federated databases. *Journal of Intelligent Information Systems*, 10:49–80, 1998.

[CM01] L Cholvy and S Moral. Merging databases: Problems and examples. *International Journal of Intelligent Systems*, 10:1193–1221, 2001.

[Coh98] W Cohen. A web-based information system that reasons with structured collections of text. In *Proceedings of Autonomous Agents'98*, 1998.

[CP87] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proceedings of VLDB'87*, pages 71–81, 1987.

[Cru86] D Cruse. *Lexical Semantics*. Cambridge University Press, 1986.

[Dem67] A P Dempster. Upper and lower probabilities induced by a multivalued mapping. *Annuals of Mathematics and Statistics*, 38:325–339, 1967.

[DP88] D Dubois and H Prade. *Possibility theory: An approach to the computerized processing of uncertainty*. Plenum Press, 1988.

[DP98] D Dubois and H Prade, editors. *Handbook of Defeasible Resoning and Uncertainty Management Systems*, volume 3. Kluwer, 1998.

[Fen00] D Fensel. The semantic web and its languages. *IEEE Intelligent Systems*, 14:67, 2000.

[FHvH+00] D Fensel, I Horrocks, F van Harmelen, S Decker, M Erdmann, and M Klein. Oil in a nutshell. In R Dieng, editor, *Proc. of the 12th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW'00)*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer-Verlag, 2000.

[FS99] E Franconi and U Sattler. A data warehouse conceptual data model for multidimensional aggregation. In S Gatziu, M Jeusfeld, M Staudt, and Y Vassiliou, editors, *Proceedings of the Workshop in Design and Management of Data Warehouses*, 1999.

[FvHH+02] D Fensel, F van Harmelen, I Horrocks, D L McGuinness, and P F Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16:38–45, 2002.

[Gar88] P Gardenfors. *Knowledge in Flux*. MIT Press, 1988.

[GM99] G Grahne and A Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*, Lecture Notes in Computer Science. Springer, 1999.

[HG00] K Hui and P Gray. Developing finite domain constraints – a data model approach. In *Proceedings of Computation Logic 2000 Conference*, pages 448–462. Springer, 2000.

[HGNY97] J Hammer, H Garcia-Molina, S Nestorov, and R Yerneni. Template-based wrappers in the TSIMMIS system. In *Proceedings of ACM SIGMOD'97*. ACM, 1997.

[HL05a] A Hunter and W Liu. Fusion rules for merging uncertain information. *Information Fusion*, (in press), 2005.

[HL05b] A Hunter and W Liu. A logical reasoning framework for modelling and merging uncertain semi-structured information. In B Bouchon-Meunier, G Coletti and R Yager, editors, *Modern Information Processing: From Theory to Applications* Elsevier, (in press), 2005.

[HL05c] A Hunter and W Liu. Merging uncertain information with semantic heterogeneity in XML. *Knowledge and Information Systems* , (in press), 2005.

[HL05d] A Hunter and W Liu. Measuring the quality of uncertain information using possibilistic logic. In *Quantitative and Qualitative Approaches to Reasoning with Uncertainty*, LNCS Volume 3571, pages 415-426, Springer, 2005.

[HS03a] A Hunter and R Summerton. FusionRuleML: Representing and executing fusion rules. Technical report, UCL Department of Computer Science, 2003.

[HS03b] A Hunter and R Summerton. Propositional fusion rules. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 2711 of *Lecture Notes in Computer Science*, pages 502–514. Springer, 2003.

[HS04a] A Hunter and R Summerton. Fusion rules for context-dependent aggregation of structured news reports. *Journal of Applied Non-classical Logic*, 14(3):329-366, 2004.

[HS05] A Hunter and R Summerton. Merging news reports that describe events. *Data and Knowledge Engineering*, (in press), 2005.

[Hun02a] A Hunter. Logical fusion rules for merging structured news reports. *Data and Knowledge Engineering*, 42:23–56, 2002.

[Hun02b] A Hunter. Measuring inconsistency in knowledge via quasi-classical models. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'2002)*, pages 68–73. MIT Press, 2002. ISBN 0-262-51129-0.

[Hun02c] A Hunter. Merging structured text using temporal knowledge. *Data and Knowledge Engineering*, 41:29–66, 2002.

[Hun03] A Hunter. Evaluating the significance of inconsistency. In *Proceedings of the International Joint Conference on AI (IJCAI'03)*, pages 468–473, 2003.

[Hun04] A Hunter. Logical comparison of inconsistent perspectives using scoring functions. *Knowledge and Information Systems Journal*, 6(5):528-543, 2004.

[Hun05] A Hunter. How to act on inconsistent news: Ignore, resolve, or reject. *Data and Knowledge Engineering*, (in press), 2005.

[KLM03] S Konieczny, J Lang, and P Marquis. Quantifying information and contradiction in propositional logic through epistemic actions. In *Proceedings of the 18th International Joint Conference on Artificial Intellignce (IJCAI'03)*, pages 106–111, 2003.

[KM91] H Katsuno and A Mendelzon. On the difference between updating a knowledgebase and revising it. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (KR'91)*, pages 387–394. Morgan Kaufmann, 1991.

[KM98] C Knoblock and S Minton. The Ariadne approach to Web-based information integration. *IEEE Intelligent Systems*, 13(5):17–20, 1998.

[KMA+98] C Knoblock, S Minton, J Ambite, N Ashish, P Modi, I Muslea, A Philpot, and S Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Madison, WI, 1998.

[KMA+99] C Knoblock, S Minton, J Ambite, N Ashish, P Modi, I Muslea, A Philpot, and S Tejada. The Ariadne approach to information integration. *International Journal of Cooperative Information Systems*, 1999.

[Kni01] K Knight. Measuring inconsistency. *Journal of Philosophical Logic*, 31:77–98, 2001.

[Kni03] K Knight. Two information measures for inconsistent sets. *Journal of Logic, Language and Information*, 12:227–248, 2003.

[Kno95] C Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of IJCAI-95*. Montreal, Canada, 1995.

[KP98] S Konieczny and R Pino Perez. On the logic of merging. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 488–498. Morgan Kaufmann, 1998.

[KS86] R Kowalski and M Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[Lee92] S Lee. Imprecise and uncertain information in databases: An evidential approach. *Proceedings of Sixth International Conference on Data Engineering (ICDE)*, pages 614–621, 1992.

[Lev00] A Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Logic Based Artificial Intelligence*. Kluwer, 2000.

[Loz94] E Lozinskii. Resolving contradictions: A plausible semantics for inconsistent systems. *Journal of Automated Reasoning*, 12:1–31, 1994.

[LS98] P Liberatore and M Schaerf. Arbitration (or how to merge knowledgebases). *IEEE Transactions on Knowledge and Data Engineering*, 10:76–90, 1998.

[LSS00] Y Loyer, N Spyratos, and D Stamate. Integration of information in four-valued logics under non-uniform assumptions. In *Proceedings of 30th IEEE International Symposium on Multiple-Valued Logic (ISMVL2000)*. IEEE Press, 2000.

[Mot96] A Motro. Cooperative database systems. *International Journal of Intelligent Systems*, 11:717–732, 1996.

[MS99] R Miller and M Shanahan. The event calculus in classical logic: An alternative axiomatisations. *Linkoping Electronic Articles in Computer and Information Science*, 4(16), 1999.

[MT98] V Marek and M Truszczynski. Revision programming. *Theoretical Computer Science*, 190:241–277, 1998.

[NS94] R Ng and V Subrahmanian. Stable semantics for probabilistic deductive databases. *Information and Computation*, 110(1):42–83, 1994.

[PHG+99] A Preece, K Hui, A Gray, P Marti, T Bench-Capon, D Jeans, and Z Cui. The KRAFT architecture for knowledge fusion and transformation. In *Expert Systems*. Springer, 1999.

[PM98] A Poulovassilis and P McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28:47–71, 1998.

[PSB+99] N Paton, R Stevens, P Baker, C Goble, S Bechhofer, and A Brass. Query processing in the TAMBIS bioinformatics source integration system. In *Proceedings of the 11th International Conference on Scientific and Statistical Databases*, 1999.

[SA99] A Sahuguet and F Azavant. Building light-weight wrappers for legacy web data-sources using W4F. In *Proceedings of the International Conference on Very Large Databases (VLDB'99)*, 1999.

[SGHB01a] R Stevens, C Goble, I Horrocks, and S Bechhofer. Building a bioinformatics ontology using oil. *Special issue: IEEE Information Technology in Biomedicine*, 2001.

[SGHB01b] R Stevens, C Goble, I Horrocks, and S Bechhofer. Oiling the way to machine understandable bioinformatics resources. *Special issue: IEEE Information Technology in Biomedicine*, 2001.

[Sha76] G Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[Sha99] M Shanahan. The event calculus explained. In M.J.Wooldridge and M.Veloso, editors, *Artificial Intelligence Today*, volume 1600 of *Springer Lecture Notes in Artificial Intelligence*, pages 409–430. Springer, 1999.

[SL90] A Sheth and J Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22:183–236, 1990.

[SO99] K Smith and L Obrst. Unpacking the semantics of source and usage to perform semantic reconciliation in large-scale information systems. In *ACM SIGMOD RECORD*, volume 28, pages 26–31, 1999.

[TKM98] S Tejada, C Knoblock, and S Minton. Handling inconsistency for multi-source integration. In *Workshop on AI and Information Integration (AAAI'98)*. AAAI, 1998.