Driving User Interfaces from FaCT

Sean Bechhofer and Ian Horrocks
Department of Computer Science
University of Manchester, UK
Email: {seanb, horrocks}@cs.man.ac.uk

Abstract

We describe a mechanism that can be used to drive interfaces from a description logic (DL) model of the domain. A simple layer with limited expressivity sits on top of the DL, with the interface behaviour described using a collection of application specific assertions. DL reasoning is then employed to ensure that the choices offered to the user in expression construction are "reasonable" as well as valid.

1 Introduction

An approach for driving user interfaces from a Description Logic (DL) model of the domain is described in [2, 4]. This uses a mechanism called *sanctioning*, which is an integral part of GRAIL DL implementation [6]. As well as user interface generation, the GRAIL sanctioning mechanism has been used for a range of other purposes, in particular as an aid to knowledge modellers.

Although suited to a number of applications, GRAIL provides a limited expressiveness and uses structural subsumption algorithms which are known to be incomplete. In addition, the multiple roles played by sanctions sometimes caused conflict and confusion and led to compromise in the conceptual modelling. However, the ability to drive an interface which allows users to construct expressions without having to explicitly deal with the underlying DL syntax has proved indispensable, particularly in the context of projects such as TAMBIS [1].

In this paper, we describe a sanctioning-like mechanism which can be used with a more expressive logic. The sanctions are no longer part of the underlying logic, but are implemented in a separate layer that makes use of the reasoning services provided by the underlying DL. This architecture provides a cleaner separation of application-specific information and functionality from the logical reasoning of the DL. The separation also makes it clear exactly what the sanctions are for, allows a clearer specification of how the interfaces should behave, and makes explicit the role of the DL reasoner in the interface generation.

The mechanism described here is not intended to be equivalent to sanctioning as it was defined in GRAIL, but is a (more flexible) substitute for sanctioning in user interface applications. To avoid confusion we will call the mechanism *reasonableness.*

2 Interfaces

A DL model of a domain can be used to drive an interface which allows a user to form a (possibly new) description by navigating through the hierarchy to an existing concept, and then (optionally) specialising that concept. For example, the TAMBIS system [1] allows the user to phrase DB queries using a DL model of bioinformatics. The query (a DL concept) is formed by navigating the model to find a concept that nearly expresses the query and then (optionally) further specialising this concept by conjoining new existential restrictions (concepts of the form $\exists R.C$). The query is then rewritten to appropriate queries over distributed information sources. The query interface relies on users being able to construct concept expressions using a graphical interface that helps to insulate the naive user from the underlying representation.

Such an interface can be constructed by exploiting a mechanism which is able to answer the question "what might I want to say about this concept?"; the answer to this question can then be used to present possible specialisation options to the user. For example, in a model concerned with costume and clothing, we may know that, in general, items of clothing are worn on parts of the body, and we may therefore wish to prompt the user as to which part of the body an item is worn on. However, it may not be the case that all items are worn on a part of the body, or that items can only be worn on parts of the body, in which case we would not wish to include such restrictions in the model. Using reasonableness, we can capture the fact that, *in general*, it is reasonable to form new concepts by specialising items of clothing with information about the part of the body where they are worn.

3 Reasonableness

The basic purpose of reasonableness is to restrict the way in which (naive) users can form new concepts so that:

1. only "reasonable" concepts can be formed, and
2. only a "reasonable" number of specialisation choices are offered at any point.

In GRAIL, the sanctioning mechanism is used to restrict the possible specialisation choices to "sensible" existential restrictions. The notion of sensibleness is defined by sanctions and is inherited down the concept hierarchy. For example, the existential restriction $\exists$*worn-on*.body-part may be sanctioned for the concept garment, meaning that it is sensible to form new concepts by conjoining (sub-concepts of) $\exists$*worn-on*.body-part with (sub-concepts of) garment. In GRAIL, the expressiveness of the sanctioning mechanism more or less corresponds with the expressiveness provided by the underlying language, but there is no particular reason why this should be the case.

3.1 Restricted Concept Language

In contrast to GRAIL sanctions, the reasonableness mechanism uses two different concept languages. The complete language will be available to sophisticated users (for example knowledge engineers) when designing the concept hierarchy.[1] A simpler (subset) language will be available to naive users when navigating the hierarchy and forming new queries/concept descriptions. Reasonableness applies to this restricted language, and guides and restricts the way that naive users can form composite concepts. In our initial implementation, this restricted query language consists only of conjunction and existential restriction (a concept of the form $\exists R.C$).

In order to further restrict the way in which the query language can be used, only concepts that are "reasonable" can be formed. This is imposed by having, for each concept name C in the Knowledge Base (KB), a list of those concepts that may reasonably be conjoined with C. The reasonableness mechanism is not part of the underlying DL (FaCT in this case), but uses the concept hierarchy as a "hanger" for reasonableness information and the DL's reasoning services to maintain this information (the reasonableness layer is a client of the CORBA-FaCT server [3]). Reasonableness information consists of a set $\mathbf{R}$ of assertions of the form `reasonable`(C, D) or `reasonable`$(C, \exists R.D)$, where C and D are concept names occurring in the KB and R is a role name occurring in the KB. A concept $C' \sqcap D'$ is said to be *reasonable* iff there is an assertion `reasonable`$(C, D) \in \mathbf{R}$ such that $C' \sqsubseteq C$ and $D' \sqsubseteq D$.

In order to restrict the number of possible ways in which an interface might prompt a user to specialise a concept C, we will define the minimal non-redundant set $\mathcal{R}_C$ of concepts that might reasonably be conjoined with C, such that $D \in \mathcal{R}_C$ iff:

1. $C \sqcap D$ is reasonable,
2. $C \sqcap D$ is satisfiable (i.e., $C \sqcap D \not\sqsubseteq \bot$),
3. $C \sqcap D$ is not equivalent to C (i.e., $C \not\sqsubseteq C \sqcap D$) and
4. D is not tautological (i.e., D is not subsumed by some other $D' \in \mathcal{R}_C$.

These sets can be pre-calculated for concepts in the hierarchy, but must be calculated on the fly for new concepts created by the application.

[1] In the current implementation, this language is $\mathcal{SHIQ}$ [5].

Note that, while a restricted language is being used in query formulation, the full power of the underlying logic is available to knowledge engineers when constructing the knowledge base. One could envisage further "layers" of expressivity that could be supplied, depending on the sophistication of the users. Moreover, although reasonableness guides the specialisation and query construction process, but is not intended to be a hard and fast restriction on the expressive power of the query language—experienced users may be able to "break out" of the interface in order to use some or all of the richer expressive power of the underlying DL.

4 Comparison with GRAIL

It is useful to compare and contrast the functionality of Reasonableness with that provided by GRAIL's sanctioning.

- Reasonableness has the advantage that it supports conjunction with other named concepts, not just existential restrictions (GRAIL attribute-value pairs), and extending the mechanism to deal with other deterministic constructs (e.g., number restrictions) does not appear difficult. Moreover, as reasonableness does not have any semantic significance w.r.t. the underlying DL, it would be easy to allow more sophisticated users to override the restrictions it imposes.

- Reasonableness interacts in a natural way with value restrictions ($\forall R.C$ concepts) in the KB, potentially reducing the number of choices offered as concepts become highly specialised and more value restrictions apply. This also allows "reasonable" assertions to be cancelled in a clean way (but not then re-applied, as is possible in GRAIL).

- As described above, reasonableness has no built in mechanism for guiding the knowledge engineer, such as GRAIL's grammatical sanctioning—reasonableness assertions can be added as and where the knowledge engineer likes. However, it would be easy to add authoring extensions that warned of unsatisfiable, non-specialising or tautological assertions.

5 Prototype Implementation

A prototype of the *reasonableness* layer has been implemented, and is being included in the latest release of the TAMBIS system (which now uses FaCT rather than GRAIL). The prototype provides a layer that sits between the CORBA-FaCT server and the application as shown in Figure 1. The application can, of course, still communicate directly with the server.

In the TAMBIS application, all the reasonableness assertions must be of the form `reasonable`$(C, \exists R.D)$ (i.e., there are no assertions `reasonable`(C, D) where

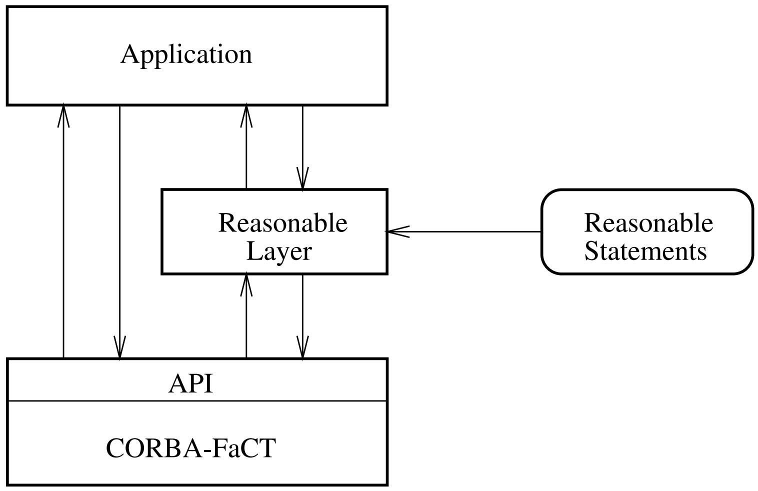

Figure 1: Architecture

D is not an existential restriction). The reasonableness information is used by the interface in order to generate data entry forms which allow specialisation of a query concept by offering relevant role-concept pairs as possible additional existential restrictions (see [2] for further details). This represents a slight restriction w.r.t. the language supported by the reasonableness mechanism, as the user can only generate "frame-concepts" of the form $C \sqcap \exists R_1.D_1 \sqcap \exists R_2.D_2 \sqcap \ldots \sqcap \exists R_n.D_n$, where C is a concept name and each of the $D_1, \ldots, D_n$ are, in turn, "frame-concepts". However, the result is a simple "framelike" language that corresponds almost directly with the form of GRAIL expressions, thereby allowing us to reuse our original graphical interface.[2]

In addition, the prototype allows the specification of a concept as "invisible", which prevents it appearing directly on a form—its direct sub-concepts will appear instead. This is simply "syntactic sugar", allowing us to reduce the number of reasonableness assertions by applying them at a more general level.

5.1 An Example

As an illustration of reasonableness in action, consider a simple model of costume. We have items of clothing (e.g., shirts, hats, boots and so on), along with parts of the body that these things can be worn on (e.g., leg, arm) and purposes to which they can be put (e.g., decoration, protection, support). In addition, we have regions of the body such as above or below the waist. Axioms are used to assert facts about articles (e.g., $Hat \sqsubseteq \exists wornOn.Head$), to define how parts of the body relate to

[2]There is no theoretical reason for not extending the interface to support (at least) the full expressive power of the reasonableness mechanism, but providing efficient graphical renderings for arbitrary conjunctions (or further expressivity) is an issue that needs further investigation.

regions (e.g., $Head \sqsubseteq \exists partOf.AboveWaist$) and to add general knowledge about the domain (e.g., $\exists wornOn.Head \sqsubseteq \exists worn.AboveWaist$).[3] We can now add the following reasonableness assertions:

$\texttt{reasonable}(Item, \exists wornOn.BodyPart)$
$\texttt{reasonable}(Item, \exists worn.BodyRegion)$
$\texttt{reasonable}(Item, \exists hasPurpose.Purpose)$
$\texttt{invisible}(BodyPart)$
$\texttt{invisible}(BodyRegion)$
$\texttt{invisible}(Purpose)$

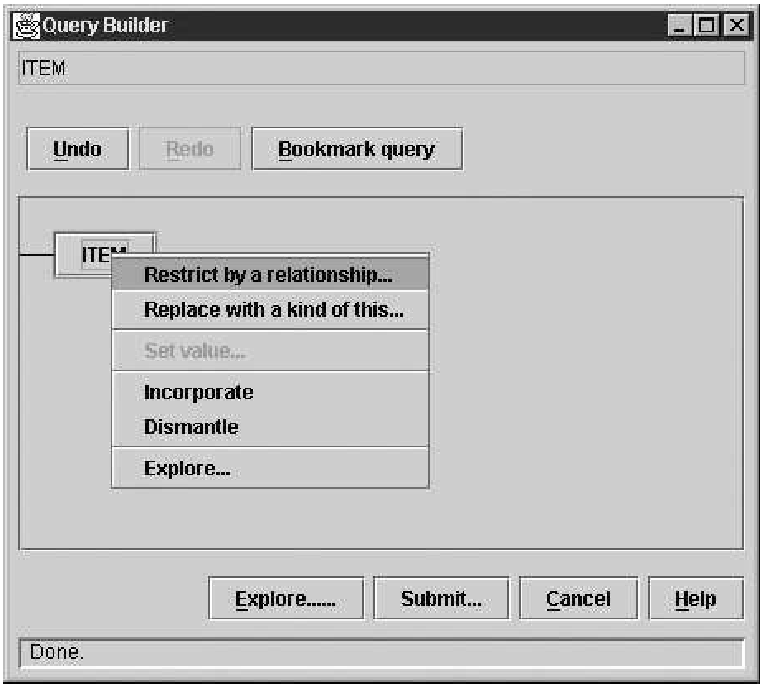

Figure 2: Initial Query

Figure 2 shows the initial query builder window for a query based on the concept $Item$. The user has clicked on $Item$, indicating that they wish to specialise the query. This results in the form shown in Figure 3, where the user has already made their specialisation selections. The result will be the new query shown in Figure 4. This query corresponds to the concept $Item \sqcap \exists hasPurpose.Protection \sqcap \exists wornOn.Head$.

In order to illustrate the interaction of the reasoner and the interface, consider the case where the model contains an axiom $Item \sqcap \exists hasPurpose.Decoration \sqsubseteq$

[3]The modelling here is not really of interest, but it is useful to describe some of the model in order to illustrate the behaviour of reasonableness.

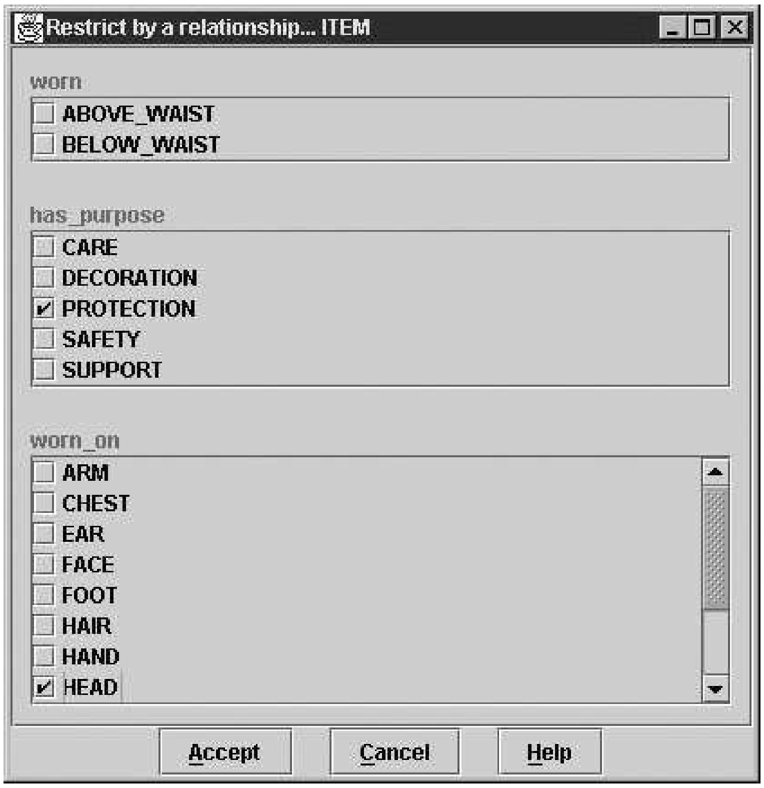

Figure 3: Initial Restrictions

$\forall worn.\neg BelowWaist$, i.e., an assertion that decorative items cannot be worn below the waist.[4] If we now take the query shown in Figure 5, and attempt to specialise, the options provided are as shown in Figure 6. The important point here is that $BelowWaist$ is no longer offered as a possible specialisation (and neither are any of the possible subconcepts of $BodyPart$ which are said to be below the waist, such as $Foot$).

It is important to note that the use made of the reasonableness information is under the control of the application. The interface shown here has a fairly *loose* coupling to the reasonableness layer. The application uses the reasonableness layer to calculate $\mathcal{R}_C$ for the focus concept C, and then generates the form. However, once the form has been generated, the interface does not communicate with the reasonableness layer until all choices have been confirmed. A tighter coupling would allow the form to change dynamically: $\mathcal{R}$ could be recalculated after each selection and used to grey out those options no longer applicable (or add new options which have become available). In the example above, if the user first selected $hasPurpose$—$Decoration$ on the

[4]Again, this is possibly a strange assertion to make, but serves the purpose of illustrating the mechanism's behaviour.

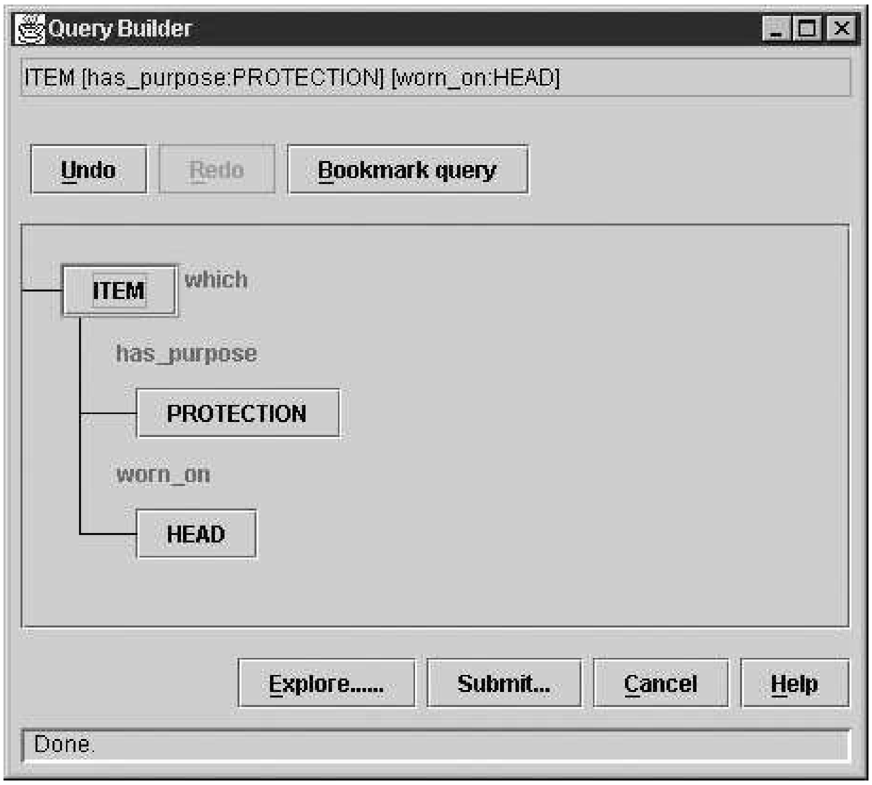

Figure 4: Refined Query

form, the options referring to body parts below the waist would then be disabled.

6 Conclusion

We have described a reasonableness mechanism consisting of a restricted query language (a subset of the underlying DL language) and a technique for exploiting additional domain knowledge, in the form of reasonableness assertions, to constrain the way in which new query concepts can be formed. The use of this mechanism has been illustrated by a user interface application, the purpose for which it was originally conceived. However, the mechanism could also be useful in other applications, e.g., supporting/constraining the extension of an existing KB by less sophisticated users.

A prototype has been implemented and has already been used in the TAMBIS system. This implementation uses a modular architecture, with the CORBA-FaCT server providing the DL reasoning services. As well as cleanly separating reasonableness from the underlying DL, this architecture would make it relatively simple to use the mechanism with other DL implementations.

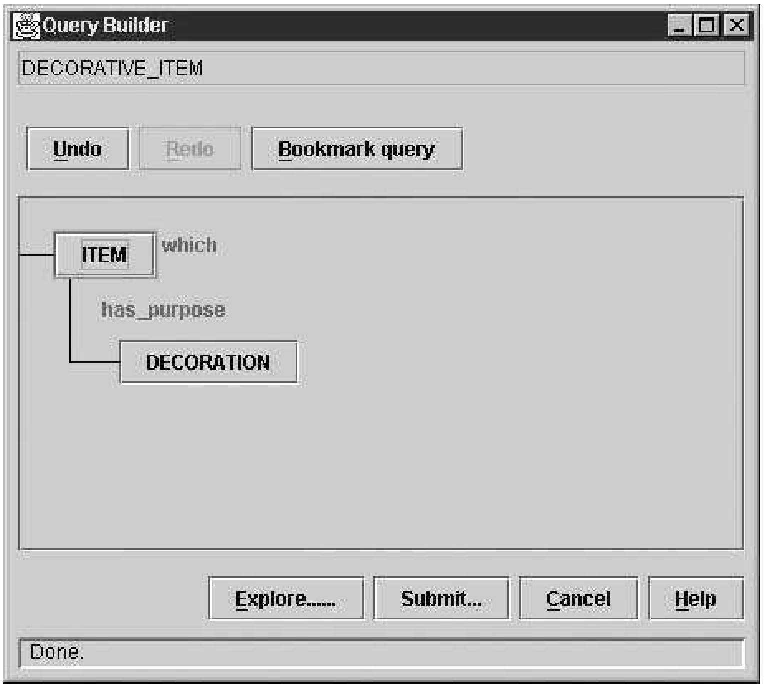

Figure 5: Query

Acknowledgement

This work greatly benefited from discussions with Franz Baader during a visit he made to Manchester University. The work was supported in part by EPSRC grant GR/L71216.

References

[1] P.G. Baker, A. Brass, S. Bechhofer, C.A. Goble, N.W. Paton, and R Stevens. Tambis: Transparent access to multiple bioinformatics information sources. an overview. In *Proceedings of ISMB98, 6th International Conference on Intelligent Systems for Molecular Biology*, 1998.

[2] Sean Bechhofer and Carole A. Goble. Using Description Logics to Drive Query Interfaces. In *Proceedings of DL'97, International Workshop on Description Logics*, 1997.

[3] Sean Bechhofer, Ian Horrocks, Peter F. Patel-Schneider, and Sergio Tessaris. A Proposal for a Description Logic Interface. In *Proceedings of DL'99, International Workshop on Description Logics*, pages 33–36, 1999.

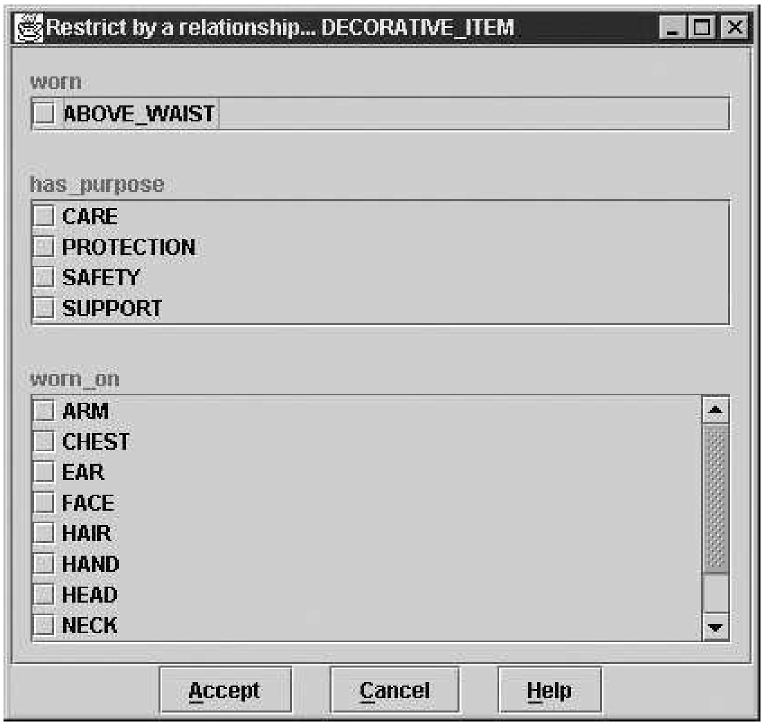

Figure 6: Restrictions

[4] Sean Bechhofer, Stevens Robert, Gary Ng, Alex Jacoby, and Carole A. Goble. Guiding the user: An ontology driven interface. In *Proceedings of UIDIS, Workshop on User Interfaces to Data Intensive Systems*, pages 158–161. IEEE Computer Society Press, 1999.

[5] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedingsof LPAR'99*, number 1705 in LNAI, pages 161–180. Springer, 1999.

[6] A.L. Rector, Bechhofer S.K., C.A. Goble, I. Horrocks, Nowlan W.A., and Solomon W.D. The grail concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.