GALEN-IN-USE
Deliverable 9.2

TeleKnoME Knowledge Management and Version Control Environment with Telematic Support

GALEN-IN-USE

Generalized Architecture for Languages Encyclopaedias

and Nomenclatures in Medicine

Project No : HC1018



Document Code:	D9-2
Project Code:	HC1018
Project Acronym:	GALEN-IN-USE
Project Title:	Generalized Architecture for Languages Encyclopaedias and Nomenclatures in Medicine
Deliverable Number:	9.2
Deliverable I.D:	D9-2
Deliverable Date:	January 31 1998
Deliverable Title:	TeleKnoME Knowledge Management and Version Control Environment with Telematic Support
Deliverable Nature:	Prototype
Deliverable Type:	Project Deliverable
Authors:	W.D.Solomon
Contributors:	C. J. Toft, J. E. Rogers, A.L.Rector

Distributor: (Project Coordinator -University of Manchesster- or local address)
Project Coordinator,
University of Manchester,
Medical Informatics Group,
Department of Computer Science,
Manchester M13 9PL
UK

Tel: +44 161 275 6133

Fax: +44 161 275 6932

Email: galen@cs.man.ac.uk

http://www.cs.man.ac.uk/mig/giu/index.html

Copies of this document can be ordered from the distributor.

Copyright Statement

All rights are reserved.

Reproduction in whole or in part is prohibited
without the written consent of the copyright owner.

Executive Summary

The GALEN-IN-USE project supports different kinds of applications: clinical applications as part of the day-to-day provision of medical care, and tools for managing existing, or building new, coding or classification schemes. All of these applications and tools require the presence of an underlying model of clinical terminology, known as the Common Reference Model, managed and presented via a GALEN Terminology Server. The Common Reference Model is built using a formal, compositional and generative concept representation language known as GRAIL.

Only by using GRAIL can the Common Reference Model support these different kinds of tasks, to the complexity required. However, its use places additional requirements on the infrastructure required to support it:

i) to build and extend the model (which may take place concurrently at different sites);

ii) to manage the collaborative and distributed nature in which the model is built;

iii) to support the process of mapping elements of existing classification schemes in the model.

This deliverable describes the requirements for, and implementation of, an environment for supporting the collaborative development of the GALEN Common Reference Model within the GALEN-IN-USE project. The environment as a whole is known as 'TeleKnoME', and comprises the following software components corresponding to the tasks above, together with the telematic infrastructure which link them together:

i) 'KnoME-lite' to browse, explore or extend a model;

ii) 'KnoME-pro' to support the integration of model extensions, and the version control and delivery of the model;

iii) 'Tigger' to support the process of mapping existing classification schemes to the Common Reference Model.

In addition, TeleKnoME supports other aspects of GALEN-IN-USE; it provides *configuration* information to the Classification Workbench (which is used within classification centres) and *perspective* information to some of the clinical demonstrators within the project. Configuration and perspective information provides an additional layer which transforms the generic, application independent model (which may offer more complexity and scope than is required for any one individual application), into one which is appropriate to support a specific application.

TeleKnoME is in daily use within the project, and will continue to be used throughout the third year of the project.

This deliverable relates to work described in:

- GALEN-IN-USE Deliverables 5.1 & 5.2, which describe the Common Reference Model in detail; TeleKnoME provides tools to build and maintain this model;
- GALEN-IN-USE Deliverable 7.2, which describes the Classification Workbench in detail; TeleKnoME provides configuration information for the Classification Workbench, and integrates and harmonises the results of its use in classifcation centres across Europe.

Contents

LIST OF FIGURES 8

1 INTRODUCTION 9

1.1 THE GALEN PROGRAMME 9
1.2 BACKGROUND 10
1.2.1 The GRAIL concept representation language 10
1.2.2 The task of 'modelling' 11
1.2.2.1 GRAIL modelling 11
1.2.2.2 Mapping elements of existing schemes to GRAIL 12
1.2.3 Supporting the modelling processes 12
1.2.3.1 The TeleKnoME 14
1.3 ORGANISATION OF THIS DELIVERABLE 14
1.3.1 Further information 14

2 OVERVIEW 15

2.1 TWO KINDS OF 'MODELLING' 15
2.1.1 GRAIL modelling 15
2.1.1.1 The 'dictionary' 15
2.1.1.2 The 'grammar' 15
2.1.1.3 Pre-defined 'phrases' 16
2.1.1.4 Creating new 'phrases' 16
2.1.1.5 How it works in practice 16
2.1.2 Mapping to coding systems 16
2.1.2.1 An intermediate representation 17
2.2 DISSECTION AUTHORING 17
2.2.1 Mapping to coding systems 17
2.2.2 Supplying requirements for GRAIL modelling 18
2.2.3 Translation of dissections into GRAIL 18

3 REQUIREMENTS 19

3.1 REQUIREMENTS AROUND GRAIL 19
3.1.1 Supporting GRAIL modelling 19
3.1.2 Extending the Common Reference Model 20
3.1.3 Model harmonisation and maintenance 21
3.1.3.1 Integrating extensions 21
3.1.3.2 Maintaining the Common Reference Model 21
3.1.3.3 Additional common requirements 21
3.2 REQUIREMENTS AROUND THE INTERMEDIATE REPRESENTATION 22
3.2.1 Supporting dissection authoring 22

3.2.2 Supporting dissection translation 22
3.2.3 Integrating & managing authored dissections 23

4 IMPLEMENTATION 24

4.1 SOFTWARE COMPONENTS WITHIN TELEKNOME 24
4.1.1 KnoME-pro 24
4.1.1.1 Tigger 24
4.1.2 KnoME-lite-extender 25
4.1.3 KnoME-lite-browser 25
4.2 RELATIONSHIP TO OTHER PARTS OF GALEN-IN-USE 25

5 QUALITY ASSURANCE 28

APPENDICES 31

APPENDIX ONE: THE GRAIL KNOME: OVERVIEW & USER GUIDE 32
1 Introduction 37
2 The Environment 43
3 Managing connections to the GALEN Terminology Server 49
4 KnoME-lite-browser 53
5 KnoME-lite-extender 66
6 KnoME-pro 71
APPENDIX TWO: TIGGER: OVERVIEW, MOTIVATION AND INTERFACE 77
1 Introduction 84
2 Dissection authoring, import, validation & management 95
3 Descriptors: their management and their mapping to GRAIL concepts 109
4 Links: their management and their mapping to GRAIL attributes 117
5 Managing Collaborative Work 122
6 Dissection Translation 125
7 CRM Extension & Validation 135
8 Discussion 143
Appendix : Conparing Read & Grail 149
APPENDIX THREE: GRAIL TUTORIAL 169
1 Introduction 174
2 GALEN, and modelling with GRAIL 176
3 An Introductory GRAIL tutorial 182
4 Using GRAIL well: A tutorial in basic GRAIL style 199
5 Refinement and Transitivity: What makes GRAIL different 206
6 Other Features 211
7 Glossary 213
APPENDIX FOUR: MIE PAPER: RUBRICS TO DISSECTIONS TO GRAIL TO CLASSIFICATIONS 216
1 Introduction 217
2 An Intermediate Representation 218
3 Expanding dissections into GRAIL 219
4 Limitations and Future extensions 221

5 Discussion and Conclusion 222
6 References: 223

List of Figures

Figure 1 : The GALEN Terminology Server providing services to different kinds of application 11

Figure 2 : Different kinds of modelling activities within GALEN-IN-USE; those supported by TeleKnoME are emphasised in grey 13

Figure 3 : GRAIL syntax for the 'phrase' representing the Read code X20WzB Appendectomy 16

Figure 4 : Intermediate representation of the Read code X20WzB Appendectomy 17

Figure 5 : Extending the Common Reference Model to support clinical applications in different areas 20

Figure 6 : The relationship and information flows between KnoME-pro (including Tigger), KnoME-lite, the Classification Workbench and the GALEN Terminology Server 26

1 Introduction

This deliverable describes the requirements for, and implementation of, an environment for supporting the collaborative development of the GALEN Common Reference Model within the GALEN-IN-USE project. This section provides some background to the project, and the programme of which it forms a part, and provides an overview of the areas of the project that this deliverable addresses.

1.1 The GALEN Programme

The GALEN-IN-USE project is a part of the GALEN Programme; it follows from and continues the work of the GALEN project (1992-1995). The GALEN Project demonstrated the use of an application-independent concept model (the Common Reference Model, or CRM) to support a variety of different applications, including clinical data entry and classification management. The CRM is written in a concept modelling language (GRAIL); it is maintained and classified automatically by a GALEN Terminology Server. This server also provides access to the model by providing services to client applications.

However, the GALEN project was never intended to produce a clinical model covering the entire range of clinical specialties, or to cover them to the degree of depth and complexity that might be required by any particular application. It was recognised that the task of maintaining and extending the CRM would continue, and tools and environments to support this activity would have to be developed and maintained.

A major focus in the GALEN-IN-USE project has been to concentrate on the task of providing Classification Management facilities to several co-operating classification centres across Europe, in the area of surgical procedures. This has demanded a large extension of the CRM to support this activity, both in terms of extending the CRM itself, and of mapping elements of existing classification schemes to the CRM. The GALEN-IN-USE Classification Workbench is the software developed in the project for use in classification centres: a prerequisite for its overall operation (and one of the tasks that it supports) is that the classification which it is helping to manage is mapped to the CRM.

As well as continuing the process for model building that was developed in the GALEN project, GALEN-IN-USE has developed the idea of an *intermediate representation* which greatly increases the capacity for users to provide requirements for the extension of the CRM, and for directly mapping existing classification schemes to it.

As well as being required to support classification management, the CRM has also been locally extended to support other parts of the GALEN-IN-USE project, such as the diabetes data entry system.

These pressures on the CRM have demanded the development and maintenance of tools and environments to support:

- the development and maintenance of the CRM itself;
- the mapping of existing classification schemes to the CRM.

These tools and environments are together known as the 'TeleKnoME', and form the subject of this deliverable.

1.2 Background

1.2.1 The GRAIL concept representation language

Traditional coding and classification schemes have been built by enumerating their *terms* or *rubrics* as pieces of text, manually asserting the inheritance relationship between them, and perhaps additionally representing some structure of the terms by asserting other links or relationships between parts of the scheme. This method has the advantage that traditional software tools such as text processors and database management systems can be used to support the building and maintenance process. It has the disadvantage, however, that it is very difficult, and as the scheme grows, impossible, to adequately maintain such a structure; for example, ensuring that the inheritance links between rubrics are complete.

The alternative approach that has formed the basis of the GALEN Programme has been a method that enables the computer to provide more support. Sometimes called 'declarative modelling', it turns the process on its head by saying that to build a GALEN classification system, the human modeller does *not* have to explicitly express the inheritance relationship between all the concepts in the model, nor even does he need to explicitly enumerate all the individual concepts that are covered by the classification. Instead, the GALEN modeller describes a series of elementary concepts and relationships, and builds more complex concepts using *composition*. If this is done within the context of a well-specified, formal language — or *description logic* — then a software implementation of that language can be used to interrogate the structure of all the entities in the system, and infer a complete set of inheritance links between them.

The advantages of this approach are:

- a more complex structure can be built than would otherwise be possible;
- the structure that is built is more complete and coherent than would be possible by doing it manually;
- additional organisations of the system can be produced, automatically, on request.

However, the software system is now not an optional extra; it is a crucial and indispensable part of the process, and it is significantly more complex than that required to support the building of traditional classification systems.

The language with which atomic concepts are defined, and combined with others, together with the rules for interpreting the structure to infer classification links, is known as GRAIL (the GALEN Representation and Integration Language). The GRAIL language is implemented as part of a software system called the GALEN Terminology Server, which maintains the Common Reference Model, and provides access to the model for application systems (such as the Classification Workbench software) and for tools and environments to manipulate (build, extend and maintain) the Common Reference Model.

Figure 1 provides a diagrammatic representation of how a GALEN Terminology Server supports applications, via an application programming interface, to different kinds of applications. Two instantiations of server are shown, each supporting a different kind of application. Each server is configured by the model it holds; as shown, an integral part of the server is the Concept Module, which manages the Common Reference Model. The model is developed and maintained using building and maintenance tools which require a read / write connection to the server to augment or modify the model. This model is then supplied as part of

the configuration of servers (though also locally extensible) that support different kinds of clinical applications, which typically require a read-only connection[1].

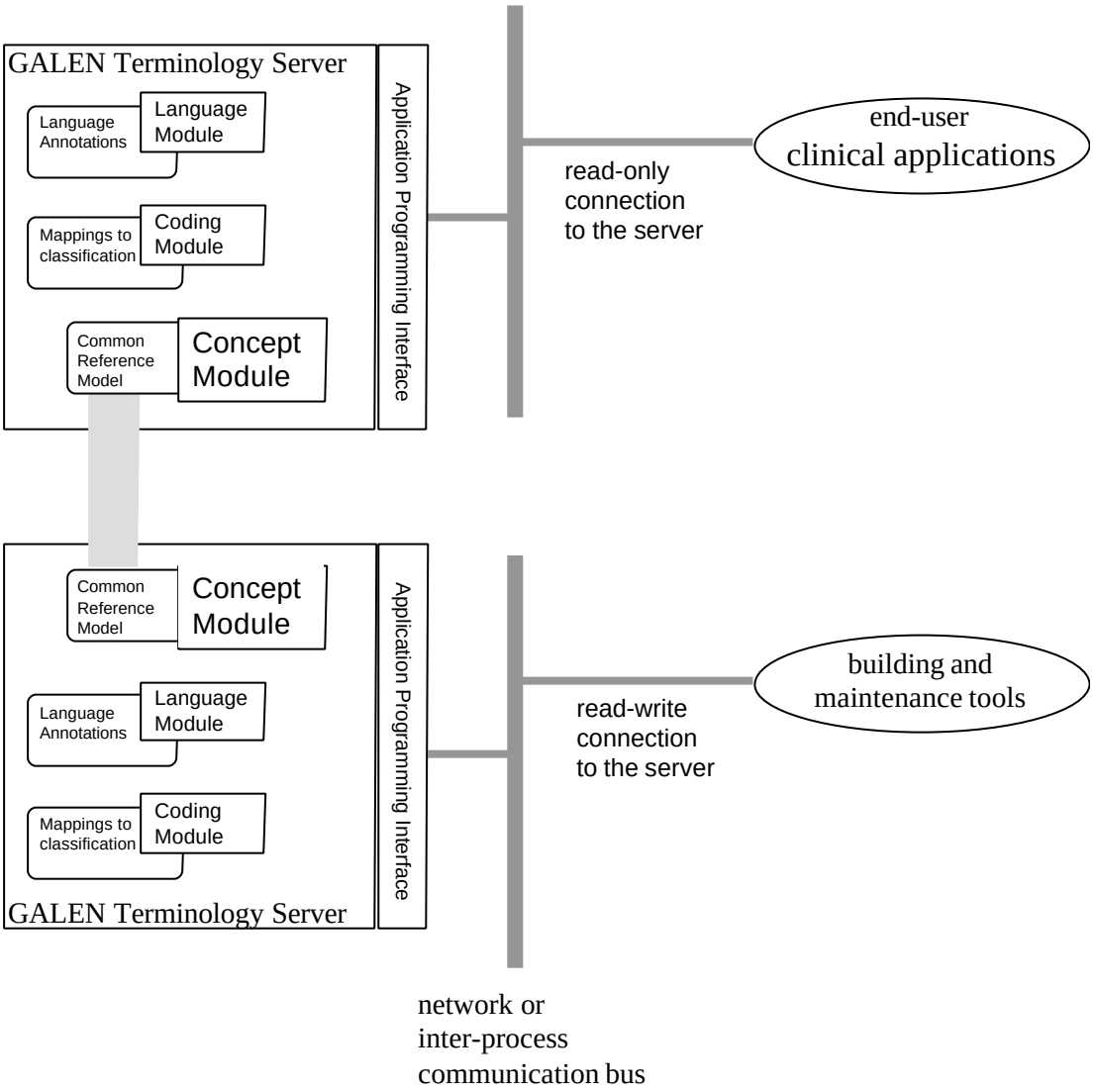


Figure 1 : The GALEN Terminology Server providing services to different kinds of application

1.2.2 The task of 'modelling'

The term 'modelling' has come to mean whatever process or processes are necessary to support the development of the model, which is in turn used to support different kinds of clinical applications. It is commonly used to refer to two different activities: GRAIL modelling, and mapping elements of existing classification schemes to GRAIL.

1.2.2.1 GRAIL modelling

An essential part of the overall process is 'GRAIL modelling'; this is the task of specifying clinical elements to the system, including their structure and properties. For example, that there

[1] Note that this because GRAIL is a generative language, the notion of a 'read-only' connection does *not* mean that the internal state of a server may not change; rather, it means that any changes only occur as a result of reification; this is discussed further in Section 0.

is the concept of an *appendix*, there is the notion of *excising*, and that *excising* can be applied to an *appendix*.

This type of modelling produces a self-contained system (a GRAIL model within a GALEN Terminology Server), that is useful for supporting particular sorts of applications. For example, the GALEN Terminology Server would be able to answer questions (presented by applications) such as *what can you do to appendices?* or *what can be excised?*

1.2.2.2 Mapping elements of existing schemes to GRAIL

However, a good deal of the focus of GALEN-IN-USE has been aimed towards the task of supporting classification centres in managing their classification schemes: this functionality is based upon the connection, or mapping, between elements of that classification scheme, and the Common Reference Model. Given that, the software that organises and classifies the GRAIL Common Reference Model can manipulate or reclassify an existing scheme.

For example, we need to make the mapping between the code

X20WzB Appendectomy

with the GRAIL representation of the clinical concept represented by that code. This process is more complex than that of mapping between two existing schemes, where two end-points are identified, and a link is added between them. When mapping between an existing classification scheme and the Common Reference Model, the GRAIL representation may need be *reified*.

Reification

GRAIL is a generative language — it is designed to support arbitrary levels of complexity that an applications may require, and thus it cannot, and does not, insist that all concepts that may ever be required are pre-enumerated. This does imply, however, that such enumerations may need to be made at the point of first use, and this is often required when mapping existing classification schemes to the Common Reference Model.

A further analogy is that of viewing existing classification schemes as a phrase-book, whilst the Common Reference Model, written in GRAIL, is more like a grammar and a dictionary of words. The combination of a grammar and dictionary allows more phrases to be expressed than can be written in a phrase-book, but the words have to be put together to form new phrases as required. That process, in the context of GRAIL, is known as 'reification'.

The task of providing (many such) mappings is also sometimes known as 'modelling' but is more correctly known as 'GRAIL mapping'. In the previous GALEN project, GRAIL mapping was done manually, which required a detailed knowledge of both the GRAIL language and the form and structure of the underlying Common Reference Model. Given the demands for much more complicated mappings (as surgical deeds can get very complex) and that the project had to produce many thousand of such mappings, rapidly, we have introduced the notion of an *intermediate representation*, as an abstraction, or half-way-house, between rubrics and GRAIL expressions. Together with automatic software to translate between the intermediate representation and GRAIL, this enables clinicians to be significantly more productive, but adds yet another layer of complexity to the tools and environments that are required.

This section has been a very brief overview; more details are given in Section 1.3.1; for full details of the structures of the Common Reference Model, see GALEN-IN-USE Deliverable 5.1.

1.2.3 Supporting the modelling processes

The previous section has introduced, at a high level, a description of two of the different processes that are involved. Software tools are required to support each of these processes at two levels:

i) support for localised activity, for example creating new GRAIL models, or adding new mappings;

ii) support for integrating the results of concurrent activity by multiple users.

These two kinds of tasks (each in the context of both GRAIL modelling, and mapping) require significantly different tools, which will in turn be used by different kinds of users:

- clinicians within classification centres require tools to help them map existing classifications to GRAIL; it is intended that there are many such people, and that it should be easy to recruit and train such people;
- the builders and maintainers of the Common Reference Model need support to pull together sets of mappings, to harmonise them and extract any requirements they may bring to the underlying Common Reference Model; this also requires clinical skills, but also demands a more specialised knowledge of the Common Reference Model itself;
- clinicians within the project, and others attached to local application builders, need tools to help them extend the Common Reference Model itself. This may be either part of a planned expansion of the model without focus on a particular application, or it may be required to support a particular clinical application's requirements. Sometimes, these will overlap. In the past, this task has also required a specialised knowledge of GRAIL and the Common Reference Model; as the demands for additional coverage of the model grow, it must be made more accessible;
- the builders and maintainers of the Common Reference Model need support to pull together GRAIL model extensions that may be created concurrently by different groups or individuals.

These activities are illustrated in Figure 2.

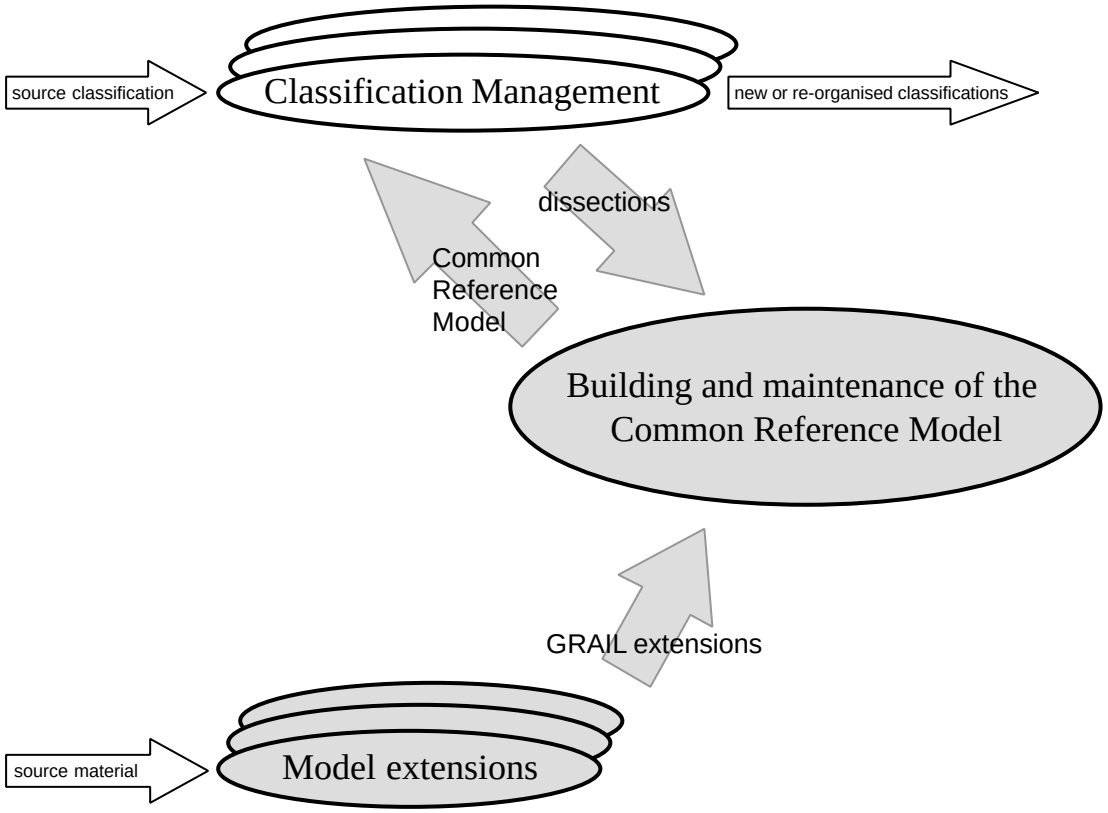


Figure 2 : Different kinds of modelling activities within GALEN-IN-USE; those supported by TeleKnoME are emphasised in grey

1.2.3.1 The TeleKnoME

The TeleKnoME is the name given to the combination of tools and environments for supporting three of the processes described above. The first is supported by the Classification Workbench, which is described in GALEN-IN-USE Deliverable 7.2.

The TeleKnoME comprises stand-alone software for use by individuals, software to harmonise and integrate the results of such work (i.e. to maintain the Common Reference Model itself), together with telematic techniques for pulling these tasks together.

The specific activities that TeleKnoME is designed to support are emphasised in grey in Figure 2.

1.3 Organisation of this deliverable

This introductory section has introduced some background to the GALEN programme, and some of the modelling processes that take place within it.

Section 2 goes into more detail of the processes involved, and how the work described in this deliverable fits within the wider scope of the GALEN-IN-USE project.

Section 3 describes some high-level requirements that arise from these processes, and Section 4 provides an overview of how these requirements are satisfied in software. Detailed descriptions of the software itself is provided in Appendices One and Two.

1.3.1 Further information

The following documents provide further information about other aspects of the GALEN-IN-USE project, which may help to put the software described in this deliverable into context:

- **GALEN-IN-USE Deliverable 7.2** describes the Classification Workbench in detail. This is the toolset which is in use by several classification centres across Europe. The Classification Workbench provides a mechanism for clinicians to enter dissections — representations of terms in the intermediate representation — as a means of analysing existing coding or classification schemes (or creating new ones), for translating these into elements in the GRAIL model, and for using the resulting GRAIL classification to validate and re-organise these schemes;
- **GALEN-IN-USE Deliverables 5.1 & 5.2** describe the structure and form of the GRAIL Common Reference Model which supports the Classification Workbench (and other applications) and is the target model into which dissections are translated.

For a tutorial introduction to the GRAIL language with which the Common Reference Model is built, see Appendix Three of this deliverable.

For more background information, see

http://www.cs.man.ac.uk/mig/galen-in-use

2 Overview

The previous introductory section introduced the kinds of modelling processes that are involved in GALEN-IN-USE. This section describes them in more detail, and further motivates the provision of an intermediate representation.

2.1 Two kinds of 'modelling'

The previous section introduced two different kinds of processes that take place in quantity within GALEN-IN-USE: GRAIL modelling, and mapping existing classification schemes to the Common Reference Model.

2.1.1 GRAIL modelling

'GRAIL modelling' is the process of extending or maintaining the GALEN Common Reference Model. As briefly described earlier, in comparison to an existing classification scheme, the Common Reference Model is analogous to a dictionary and grammar, rather than a phrase-book. This has implications for the task of building it: if it were a phrase-book, then building it would be a matter of writing down the phrases[2].

2.1.1.1 The 'dictionary'

To continue the analogy, the 'dictionary' is the set of elementary concepts that need to be provided. They are typically manually arranged in a hierarchy for convenience as they are introduced. There is a relatively small number of elementary concepts, compared to the number of combinations in which they might appear.

2.1.1.2 The 'grammar'

The 'grammar' is a combination of the rules of the language of GRAIL (which are implemented by the GALEN Terminology Server), together with the specific set of links and other information that is part of the Common Reference Model (GRAIL itself is not specific to medicine).

For example, that a link exists ('acts on') describing where something is applied, that it is possible to create a combination consisting of the deed 'excising' with the structure 'appendix' using that link, but that it is not possible to use the link between arbitrary concepts (i.e. that 'blood' 'acts on' 'fracture' is not allowed).

Further, that conditions which apply to part of a structure should be considered as kinds of conditions that apply to the structure (for example that a 'fracture of the neck of the femur' should be considered as a kind of 'fracture of femur'). This kind of information is used by the

[2] This is something of a simplification, as the analogy becomes stretched: the 'phrases' in a traditional classification scheme would also need to be arranged in a hierarchy. With GALEN's 'dictionary and grammar' approach, the arrangement of this hierarchy is provided automatically by the underlying software.

GALEN Terminology Server as it organises the hierarchy of concepts according to what has been said about them.

2.1.1.3 Pre-defined 'phrases'

It is often useful to create some 'pre-defined phrases' with this dictionary and grammar, either for further use directly in the model (for example to add some further information about them), or because they are common and well-known, and external systems may wish to use them, and expect to be able to do so with as much ease as possible.

2.1.1.4 Creating new 'phrases'

Sometimes, a pre-defined phrase will not have been previously created, and an application may need to create one specially for its purpose. This is often the case, as GRAIL can cope with arbitrary levels of complexity; it is precisely the fact that not everything *needs* to be said first that enables applications to say what they want to say, when they want to say it. GRAIL defines a syntax for creating new phrases as required.

This process — known as *reification* — is often needed during the process of mapping existing classification systems to GRAIL: the 'phrases' in the existing scheme need to be matched up with equivalent 'phrases' in the Common Reference Model. Often they will not have been pre-defined, especially as the project enters a new area, as it has in GALEN-IN-USE with its concentration on surgical procedures.

2.1.1.5 How it works in practice

There has been copious use of quote marks in this section, to indicate the analogy being followed. A tutorial introduction to GRAIL as it is actually used and defined is included in the deliverable as Appendix Three, and details of the actual implementation of the Common Reference Model are provided in GALEN-IN-USE Deliverables 5.1 & 5.2.

2.1.2 Mapping to coding systems

As indicated earlier, the GALEN-IN-USE project, with its emphasis on supporting the task of classification management, requires that a classification that is to be managed needs to be mapped to the Common Reference Model. This requires each element of that classification to be linked to the equivalent concept in the Common Reference Model.

To take a concrete example, consider the Read code X20WzB Appendectomy. This has to be mapped to the Common Reference Model phase for 'excision of appendix', assuming, as is the case, that this is not an elementary concept[3]. The actual GRAIL syntax for this is as follows:

```
(SurgicalDeed which
        isMainlyCharacterisedBy (
                performance whichG isEnactmentOf (
                        (Excising which playsClinicalRole SurgicalRole) which
                                actsSpecificallyOn AppendixVermiformis)))
```

Figure 3 : GRAIL syntax for the 'phrase' representing the Read code X20WzB Appendectomy

This is included as an illustration of the complexity required when creating new 'phrases' for this purpose (and consider that the actual surgical deed that it represents is not, in itself, a complex one). GALEN-IN-USE Deliverables 5.1 & 5.2 motivate and explain the form of the Common Reference Model, and why this seemingly simple surgical deed is represented in this way. Suffice it to say here that it *is* necessary so that the resulting classification of surgical

[3] If it were, or if it were a 'pre-defined phrase', then the task of mapping between it and an existing classification scheme would be simpler.

deeds, in the context of the overall, application-independent model, behaves as we require it to do, to support the specific task of classification management.

However, creating these 'phrases' demands fairly specialised knowledge of:

i) clinical medicine;

ii) the syntax and operation of GRAIL;

iii) the format and style of the Common Reference Model.

Knowledge of clinical medicine is why clinicians have to be involved, but experience in the earlier parts of the GALEN programme demonstrated the high cost of training clinicians in the GRAIL language and the form and style of the model, and of giving them sufficient experience to be productive.

2.1.2.1 An intermediate representation

As the GALEN-IN-USE project demanded a large-scale increase in production, an *intermediate representation* was developed that enables clinicians to use their medical expertise to create such phrases more easily.

The intermediate representation is a format for describing the structure of rubrics or terms, using a controlled (though open-ended) vocabulary, in a way that is amenable to the production of graphical data entry systems, and is automatically transformable into the underlying GRAIL syntax.

For example, the intermediate representation of the Read code X20WzB Appendectomy is as follows:

```
MAIN excising
       ACTS_ON appendix
```

Figure 4 : Intermediate representation of the Read code X20WzB Appendectomy

This representation, known as a *dissection*, is clearly much simpler that that shown in Figure 3, to which it can be automatically translated. The format and style of the intermediate representation has shown to be quickly and easily taken on by clinicians, and has enabled many thousand such dissections (and hence, new GRAIL 'phrases') to be created. This is an essential part of the infrastructure that then enables the effective use of the GALEN technology for supporting the task of classification management.

2.2 Dissection authoring

The introduction of the intermediate representation has changed the emphasis of some of the tasks that are undertaken by the classification centres in GALEN-IN-USE. Rather than have to explicitly create GRAIL 'phrases', they need to author dissections; i.e. analyse and record the structure of the elements or phrases in their existing classification schemes, in the format of the intermediate representation. This process has an additional benefit to the underlying development of the Common Reference Model.

A paper presented at MIE '97 is included as Appendix Four, which presents an overview of the process.

2.2.1 Mapping to coding systems

The process of authoring dissections provides enough information to provide the mappings between an existing classification and GRAIL. Additional information is required, as described in Section 2.2.3, but that must be supplied and managed centrally as a well-defined task.

2.2.2 Supplying requirements for GRAIL modelling

Dissections are authored using a combination of links (e.g. `ACTS_ON` as shown in Figure 4) and descriptors (e.g. `excising` or `appendix` as shown in Figure 4). Whilst the set of links are reasonably static, the number of descriptors may grow as new domains are represented. Dissection authors are provided with a managed set of preferred descriptors, but they may add their own as they find the need; for example as they cover areas that have not been seen before.

Comprehensive coverage of links and descriptors is provided by GALEN-IN-USE Deliverable 5.2.

Dissections provide a useful indication of coverage requirements to the Common Reference Model —the use of any new descriptors indicates areas of the domain that need to be covered by the Common Reference Model itself.

2.2.3 Translation of dissections into GRAIL

The previous sections have described how the use of an intermediate representation (dissection authoring) has enabled the efficient representation of rubrics found in existing classification schemes. By providing an abstract layer, separated from the underlying GRAIL syntax, clinicians do not have to be concerned about the syntax of GRAIL, or the form and structure of the Common Reference Model[4].

However, it has introduced another stage into the process, and to complete the process of mapping rubrics into GRAIL, the process of translating dissections into GRAIL must be addressed. This process is automatic, given the presence of a set of mappings and other information which specify how to translate a dissection containing a specific set of descriptors and links into the specific style of GRAIL demanded by the Common Reference Model. This includes (but is not limited to) a mapping between dissection descriptors and GRAIL concepts, and instructions for transforming dissection links into a corresponding GRAIL construct. This is an important requirement on TeleKnoME, which is explored further in the next section.

[4] In fact, it is clear that it is no longer reasonable to expect even expert users to represent complex surgical procedures directly in GRAIL; the required resulting 'phrases' are simply too complex.

3 Requirements

The previous section described the process of GRAIL modelling more fully (though see the tutorial included as Appendix Three for a more thorough description), and motivated the use of the intermediate representation within GALEN-IN-USE. These processes imply requirements on TeleKnoME which are described in this section. Section 4 describes how these requirements are implemented.

This section is not intended to provide an exhaustive list of requirements for an implementation, but rather to indicate the sorts of functionality that are required to support the activities that take place within the project, and which are supported by TeleKnoME.

3.1 Requirements around GRAIL

This section introduces some of the requirements on TeleKnoME's support of GRAIL modelling.

3.1.1 Supporting GRAIL modelling

Section 1.2.3 introduced two aspects of the support services required for GRAIL modelling:

- extending the Common Reference Model;
- integrating extensions to the Common Reference Model, known as 'model harmonisation'.

The scenario is that there may be several groups concurrently extending the model in different areas, perhaps for different purposes, whilst there is a single point of integration, harmonisation and version control, from which the Common Reference Model is distributed for use by different clinical applications. This is illustrated in Figure 5.

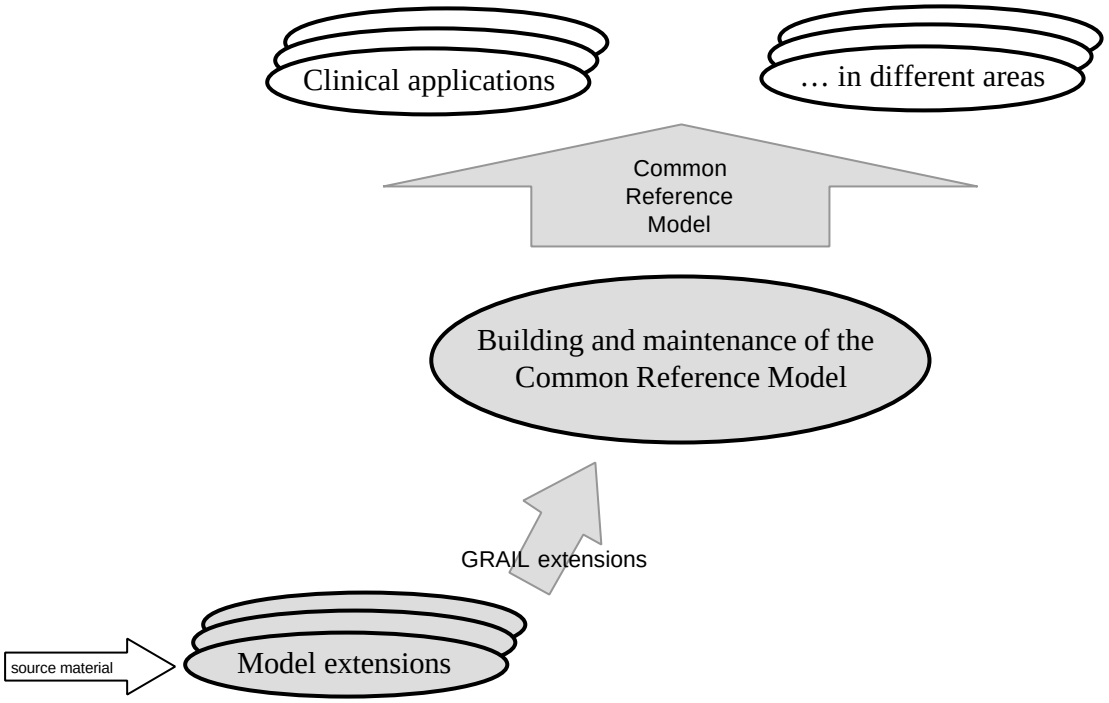


Figure 5 : Extending the Common Reference Model to support clinical applications in different areas

3.1.2 Extending the Common Reference Model

There are several different aspects to extending the Common Reference Model:

i) extending the model to cover a specific sub-domain to cover a particular application's requirements. For example, one of the partners in GALEN-IN-USE has added a Diabetes extension to support their specific requirements;

ii) extending the model in an area intended to be of general use to a number of applications. For example, the Common Reference Model includes a model of anatomy, which will form a common part of many applications' requirements; deciding which parts of anatomy to add at specific times is often influenced by the dissections that are authored (as described in Section 2.2.2);

iii) adding specific kinds of knowledge to concepts already in the model. For example, there may be a series of concepts in the model representing muscles, nerves and bones that are found in the body. Just having those concepts present in the model is one stage of covering that part of anatomy, but a subsequent and useful additional piece of modelling is the creation in the model of a series of statements indicating how these various bones, muscles and nerves interconnect.

Each of these kinds of modelling will typically be done at different times by different people, using different parts of TeleKnoME. The requirements for these parts of the tools and environment include:

- supporting a 'modelling style', and directing conformance to it; there are, potentially, different ways of representing a concept model in GRAIL, and the GALEN programme has developed a particular style in building the Common Reference Model;
- model submission; having built a local extension, this needs to be able to be extracted as a unit and delivered telematically to the central harmonisation / integration site.

3.1.3 Model harmonisation and maintenance

Given a situation where there may be many groups creating extensions to the Common Reference Model, it is important that the TeleKnoME provide facilities for integrating these extensions, as well as providing facilities for maintaining the model and delivering it for use to external applications within the GALEN Terminology Server.

3.1.3.1 Integrating extensions

Whilst integrating extensions, TeleKnoME should (amongst other things) be able to:

- load extensions created as a result of the processes described in the previous section;
- check them for conformance to the defined modelling style, and to transform common constructions so that they do conform;
- check extensions for overlap, either between themselves or with pre-existing parts of the model

3.1.3.2 Maintaining the Common Reference Model

There are different kinds of activity that are required to maintain the Common Reference Model

- correcting any problems that may be reported by users; for example ranging from simple typographical errors to incorrect or incomplete classifications that may be a result of incorrect or incomplete properties in the model;
- quality assurance; providing views of the model such that it can be subject to external validation, and ensuring that certain internal consistency checks and constraints are passed;
- providing version control of the model: delivering versions with different extensions integrated into it, together with detailed information of how one model differs from the last, which can, if presented in a machine-readable format, be used to 'auto-update' local extensions that have not yet been integrated, or local applications that use the model. Also includes commenting and documentation of the model itself.

3.1.3.3 Additional common requirements

All of the requirements described so far demand ways of seeing (browsing and exploring) what is already present in the model:

- to check that what is required is not already present;
- to find the points where new knowledge should be added;
- to find points where knowledge may be interconnected;
- for quality assurance and validation.

These requirements are similar to those presented by some kinds of clinical applications (for example data entry applications). Whilst a view on the underlying model is important for detailed internal quality assurance at the harmonisation site, it is also important to be able to apply a level of customisation on the underlying GRAIL model. In an analogous manner to the notion of an intermediate representation for dissections to hide the complexities of GRAIL, the Common Reference Model, in its raw form, often offers too much complexity for any single particular application. The set of filters and simplifications that offer the ability to customise the Common Reference Model for a particular application, or for internal quality assurance, is known as a *perspective*. Perspectives must be authored, to capture the modelling style in the Common Reference Model that is to be simplified, and they must be able to be applied on demand to present different views of the underlying model.

3.2 Requirements around the Intermediate Representation

The previous section provided an overview of the requirements on TeleKnoME as they relate specifically to GRAIL modelling. This section focuses on the additional requirements that arise as a result of the use of the GALEN-IN-USE intermediate representation.

The use of the intermediate representation offers many advantages to the project:

- it enables the form of structured representation that is required to map to existing classification schemes to be simple enough to be easily taken on by clinicians with little or no previous experience in this kind of work, yet rich enough to represent the essential aspects of the rubrics being covered;
- it provides a level of abstraction between the work of many clinicians working concurrently, and the low-level representation within the Common Reference Model. The gap between is bridged automatically, meaning that, if required, the form of the underlying representation can develop, and be maintained, without having to force users at classification centres to re-do any of their dissections.

However, these advantages have come at a price to the project, which needs to provide tools and environments, as part of TeleKnoME, to:

- support the authoring of dissections;
- support the process of translating those dissections into the underlying form required: a GRAIL representation conforming to the style of the Common Reference Model;
- manage, harmonise and integrate the dissections produced.

3.2.1 Supporting dissection authoring

Dissections are authored at local classification centres by clinicians using part of the Classification Workbench (described in GALEN-IN-USE Deliverable 7.2), known as the SPET (Surgical Procedure Entry Tool).

To make it as productive an environment as possible for its users, SPET enforces a number of *constraints*. For example, that all dissections must start with a descriptor representing a deed, and that all deeds must be further specialised by something upon which they act.

These rules are not inherent to the software, or to the translation mechanisms which subsequently translate these dissections into GRAIL, but are a reflection of the particular kinds of rubrics that we are trying to capture (in this case, surgical procedures), and are provided as part of a *configuration file*[5] which is managed by TeleKnoME.

In addition, the configuration file contains other information, such as context-specific help information for users.

3.2.2 Supporting dissection translation

Dissections form a stage along the process of mapping rubrics into GRAIL; the next stage is that of translating dissections into GRAIL; in particular into the form of GRAIL required by the form and style of the Common Reference Model.

[5] Colloquially known as the 'common.txt' file

This translation is supported by both the Classification Workbench, and by TeleKnoME. Whatever software is performing the translation, it needs to be supported by information specifying:

- how each of the descriptors used in a dissection should translate to concepts in the Common Reference Model;
- whether, and how, any further rules should be applied when translating descriptors into GRAIL concepts, to reflect the particular domain; for example, that any descriptors representing actions (such as `cutting`) should not be translated to the general GRAIL concept *cutting*, but rather to the GRAIL concept representing *cutting in the context of surgery*;
- how each of the links in a dissection should be expanded into either a GRAIL attribute, or some more complex combination of attributes and concepts; this is dependent on the surrounding context of use of the dissection link.

TeleKnoME creates and manages this information, and supplies it to the Classification Workbench as part of the configuration file introduced in the previous section.

3.2.3 Integrating & managing authored dissections

Given that there are multiple users of the Classification Workbench across Europe, the need arises for central management of the combined resource of authored dissections, and for distributed support of those users.

To support the users of the Classification Workbench, and to integrate a series of distributed resources, the TeleKnoME:

- distributes configuration files and versions of the Common Reference Model using a combination of two WWW sites and email;
- captures locally authored dissections using a combination of FTP sites and email;
- supports the local use of the Classification Workbench with a combination of email lists and feedback of management information using WWW.

The central integration and harmonisation effort includes:

- extracting new descriptors and links that may have been introduced to cover local requirements;
- mapping those descriptors and links and making that information available as part of an updated configuration file, or alternatively transforming those dissections (with others from other sites) to use part of an existing or extended set of preferred descriptors or links;
- extracting new requirements for the Common Reference Model based on the sub-domains that appear in new dissections;
- harmonising (possibly including transforming) the structure of dissections authored to a common style;
- validating the Common Reference Model based on constructs used in the sets of authored dissections.

4 Implementation

Sections 1 & 2 introduced the motivation behind, and a description of, the various modelling processes that take place in GALEN-IN-USE. Section 3 provided a more detailed overview of the requirements that these activities place on TeleKnoME.

This section provides an overview of the software components which implement these requirements, who is intended to use them, and how they relate and interact with other parts of the GALEN-IN-USE project. Appendices One and Two provide a complete description and user guide for these software components.

4.1 Software components within TeleKnoME

'TeleKnoME' as a whole comprises a series of different, though overlapping, software components.

4.1.1 KnoME-pro

KnoME-pro is the environment designed to support the integration, harmonisation and delivery of GRAIL models. It is designed for use at the central model building and integration site, and requires specialised knowledge of GRAIL and of the form of the Common Reference Model. It is described in Appendix One on page 32.

KnoME-pro includes tools to:

- maintain the overall structure and content of the model;
- define and carry out internally-defined consistency and completeness checks
- add additional sections of GRAIL model;
- document and comment the model;
- import and harmonise extensions created elsewhere
- define sets of extensions and describe perspectives for use by KnoME-lite-browser.

In addition, an important aspect of KnoME-pro is designed to support dissection authoring, which is known as 'Tigger'.

4.1.1.1 Tigger

Tigger is a sub-component within KnoME-pro, specifically designed to support the use of the GALEN-IN-USE intermediate representation. It is usually used at the central dissection authoring and integration/harmonisation site, though is sometimes used at individual classification centres for local harmonisation before reaching the central site. Tigger is described in full in Appendix Two on page 77.

Tigger includes tools to:

- create a configuration file defining the preferred vocabulary, and the style of the Common Reference Model, which is required for the use of the Classification Workbench;
- manage and map the set of descriptors and links;
- import and harmonise sets of externally-authored dissections;
- validate the Common Reference Model based on the constructions that have been authored in dissections.

4.1.2 KnoME-lite-extender

KnoME-lite-extender, by contrast, is designed for use by multiple centres to build local extensions to the GRAIL Common Reference Model. It supports the controlled extension of the model, either to add new sub-domains, or to extend the knowledge in the model in a pre-existing area. It is described in full as part of Appendix One on page 32.

It includes tools to:

- add additional GRAIL elements;
- add additional knowledge to pre-existing parts of the model;
- export changes for integration (by users of KnoME-pro).

4.1.3 KnoME-lite-browser

KnoME-lite-browser provides functionality for finding and exploring elements within the model, without having the authority to change or add to the model itself. As such, it is useful set of components to be available on its own, but these tools are also found in the KnoME-pro and KnoME-lite-extender. It is described in full as part of Appendix One on page 32.

KnoME-lite-browser includes tools such as:

- browsers to see the current structure of the model;
- explorers to see the potential effect of the model.

These tools operate in the context of any perspectives that may be in place, having been authored by users of KnoME-pro.

4.2 Relationship to other parts of GALEN-IN-USE

This section describes the relationship that the software components within TeleKnoME have with the rest of the GALEN-IN-USE project, and of the information flow between the various components. In summary, the tasks that require support in the project are:

i) creating extensions to the Common Reference Model;

ii) managing the development and maintenance of the Common Reference Model (for example, integrating extensions of the model);

iii) classification management at classification centres across Europe;

iv) supporting classification management, and integrating results from that activity into the Common Reference Model.

The tools that have been developed to support these tasks are:

i) KnoME-lite (part of TeleKnoME, described in this deliverable);

ii) KnoME-pro (part of TeleKnoME, described in this deliverable);

iii) Classification Workbench (described in GALEN-IN-USE Deliverable 7.2);

iv) Tigger (part of KnoME-pro and TeleKnoME, decribed in this deliverable);

These tools are integrated using telematic links:

- KnoME-pro maintains the Common Reference Model; it supplies versions of this model as part of the vital infrastructure supporting both KnoME-lite and the Classification Workbench;
- extensions are transferred from KnoME-lite to KnoME-pro to contribute to the development of the Common Reference Model;
- configuration information is provided from KnoME-pro to Classification Workbench, describing the style of the Common Reference Model, and a controlled vocabulary of links and descriptors which are mapped to the Common Reference Model;
- output from the Classification Workbench is transferred to KnoME-pro to contribute to the further development of the Common Reference Model.

Figure 6 presents a diagrammatic view. It illustrates:

- multiple instances of the Classification Workbench software being used. The software is used by clinicians within classification centres. The software is configured as shown by the transfer of descriptor and link mappings (and other information) from TeleKnoME via a configuration file. Dissections authored within the Classification Workbench are sent back to the KnoME-pro for integration and harmonisation. This takes place over telematic links;
- multiple instances of the KnoME-lite software being used to create new extensions of the Common Reference Model, or to add additional knowledge to it. These extensions are passed back to the KnoME-pro using telematic links;

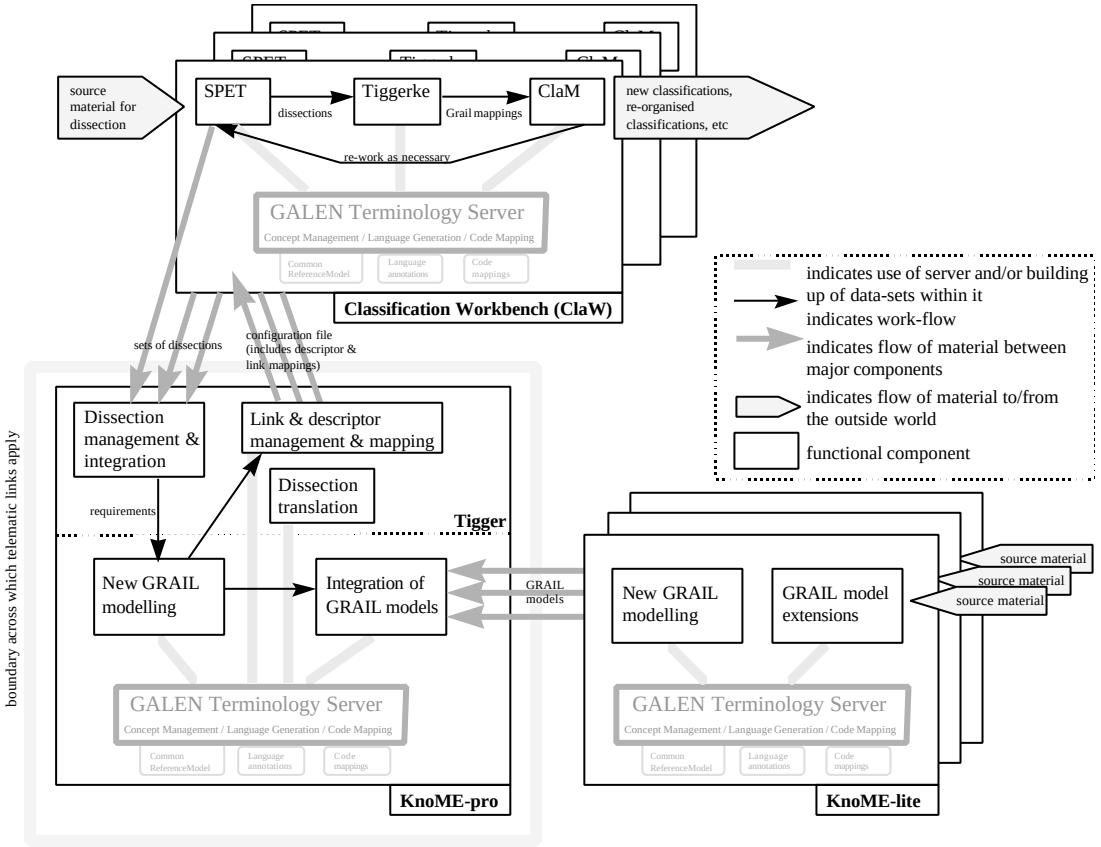


Figure 6 : The relationship and information flows between KnoME-pro (including Tigger), KnoME-lite, the Classification Workbench and the GALEN Terminology Server

- a single instance of the TeleKnoME in use at the central model integration and management centre: producing configuration files for the Classification Workbench, developing and integrating extensions to the underlying Common Reference Model, and integrating and harmonising sets of authored dissections.

All the tools are shown being supported by the GALEN Terminology Server, which contains the Common Reference Model. For clarity, the illustration does not attempt to show the distribution and version maintenance of the Common Reference Model which is supported by KnoME-pro.

The combination of KnoME-pro & KnoME-lite, together with the telematic links which connect them to each other, and to the Classification Workbench, are together known as the TeleKnoME.

5 Quality Assurance

The main validation and quality assurance of TeleKnoME is in its use within the project:

- supporting the development and maintenance of the Common Reference Model, and
- supporting the use of the Classification Workbench in classification centres across Europe.

TeleKnoME is in daily use, and will continue to be so during the third year of the project.

This section also includes forms describing the quality assurance of this deliverable.

GALEN-IN-USE Deliverable Assessor's Report

Deliverable ID: D9.2 (draft)

Deliverable name: TeleKnoME Knowledge Management and Version Control Environment with Telematic Support

Assessor's name: Nicholas Hardiker

CONFORMITY

Does the deliverable conform with the description in the Technical Annex Yes

Comments: the deliverable describes the TeleKnoME environment.

COMPREHENSIVENESS

Does the deliverable cover its subject comprehensively Yes

Comments: It is complete in its overview of the major functional components of TeleKnoME.

CLARITY

Is the deliverable presented clearly and easy to follow Yes

Comments: It is easy to follow. Ideas are presented clearly and concisely.

STATEMENT ON GENERAL QUALITY

Comments: very high quality

RECOMMENDATIONS

Acceptable for delivery Yes following minor editing

ACTIONS REQUIRED

No major revision needed. Minor editing suggested separately. In 2.1.1 a picture might clarify GRAIL modelling to naïve readers. 2.1.1.3 and 2.1.1.4 would benefit from further clarification - 'pre-defined phrases' should not be confused with enumerated terms. 4.1.3 An example of what is meant by potential effect of the model would be informative.

Note: The appendices were not reviewed. References to external information sources were not validated.

ACTIONS REQUIRED (IF NOT ACCEPTABLE FOR DELIVERY)

None.

Date: 26-01-98

Signature: Nicholas Hardiker

GALEN-IN-USE Deliverable Assessor's Report

Deliverable ID: D9.2 (draft)

Deliverable name: TeleKnoME Knowledge Management and Version Control Environment with Telematic Support

Assessor's name: Egbert J. van der Haring

CONFORMITY

Does the deliverable conform with the description in the Technical Annex: Yes

Comments: the deliverable describes the requirements for, and implementation of the TeleKnoME.

COMPREHENSIVENESS

Does the deliverable cover its subject comprehensively: Yes

Comments: the deliverable completely describes the TeleKnoME and gives introductions to all related topics together with appendices describing these topics more thoroughly.

CLARITY

Is the deliverable presented clearly and easy to follow: Yes

Comments: it is clear for specialists. Laymen may have a harder time but should be able to follow the general idea and may gain more understanding by studying the appendices.

STATEMENT ON GENERAL QUALITY

Comments: high quality

RECOMMENDATIONS

Acceptable for delivery: Yes

Acceptable for delivery after revision: (no major revision needed)

ACTIONS REQUIRED (IF NOT ACCEPTABLE FOR DELIVERY)

Comments: No major modifications needed.

Some minor modifications (mostly typo's) were suggested via e-mail.

Date: 27-01-98

Signature: Egbert J. van der Haring

Appendices

The following appendices are included as part of this deliverable:

- Appendix One on page 32 provides an overview and user guide for the KnoME-pro and KnoME-lite software — this is an important part of the implementation of TeleKnoME;
- Appendix Two on page 77 provides an overview and user guide for the Tigger software which is a part of KnoME-pro — this is also an important part of the implementation of TeleKnoME;
- Appendix Three on page 169 provides a tutorial introduction to the GRAIL language (in which the Common Reference Model is built); TeleKnoME supports the integration and maintenance of models built in GRAIL;
- Appendix Four on page 216 presents a paper from MIE '97 which describes the route from rubrics in existing classification schemes to a GRAIL classification, via the use of the GALEN-IN-USE intermediate representation; this process is supported by TeleKnoME.

Appendix One:
The GRAIL KnoME: Overview & User Guide

GALEN Documentation

The GRAIL KnoME

Overview and User Guide

Danny Solomon (danny@cs.man.ac.uk)
Conrad Toft (conrad@cs.man.ac.uk)

January 1998

Summary

This document presents an overview of the modelling environment that has been developed in the GALEN & GALEN-IN-USE projects to support the production of GRAIL models: the **GRAIL KnoME (Knowledge Modelling Environment)**. In particular, it is used to build and maintain the Common Reference Model, which is used to support different kinds of applications within the project. We provide a high-level description and user guide of the facilities it offers as a whole, together with details of individual tools provided within it.

The KnoME as a whole comprises a number of different software components:

- **KnoME-lite-browser** is used to view and explore the model;
- **KnoME-lite-extender** is used to build extensions to the model;
- **KnoME-pro** is a suite of tools for integrating extensions to the model, and for managing harmonisation and release of the model.

KnoME-pro also includes a series of tools to help support the process of mapping elements of existing classification schemes to the Common Reference Model. These tools are known as Tigger, and are described separately (in "Tigger: Overview, Motivation and Interface").

The KnoME is used in conjunction with a GALEN Terminology Server; details are also included of the tools used to view and manipulate the state of the server. The KnoME can communicate with two different implementations of server: the **TeS** and the **Puppi**.

We also provide some information concerning the anticipated development of the KnoME in the short- and medium-term future. This document is valid for the [KnoME 2.1 or KiT 2.1] versions of the KnoME tools.

For help or further information, contact
the authors or Project Co-ordinator:

email: galen-help@cs.man.ac.uk
phone: (+44) 161 275 6133
web: http://www.cs.man.ac.uk/mig/galen

Contents

1 INTRODUCTION 37

1.1 ARCHITECTURE 38
1.1.1 Overview: the process of 'modelling' 39
1.1.1.1 Concept Modelling 39
1.1.1.2 The Coding Module: Code Mapping 40
1.1.1.3 The Language Module: Validation and Linguistic Annotation 40
1.2 IMPLEMENTATION AND DELIVERY STRATEGY 40
1.3 PHILOSOPHY 41
1.4 DOCUMENT HISTORY 41
1.5 FURTHER INFORMATION 42
1.6 FEEDBACK 42

2 THE ENVIRONMENT 43

2.1 SMALLTALK 43
2.1.1 Mouse operations 43
2.2 STARTING AND CLOSING THE KNOME 44
2.2.1 Transcript 44
2.2.2 Snap-shotting the image 44
2.2.3 Exiting the environment 45
2.3 THE TES CONSOLE 45
2.3.1 The TeS Console Main Window 45
2.4 PREFERENCES 45
2.5 HELP 46
2.5.1 GRAIL Syntax and Operations 46
2.6 PROBLEMS, COMMENTS AND ENHANCEMENTS 46
2.6.1 System errors 46
2.6.2 Reporting a comment or problem to us 47
2.6.3 Applying a patch 47

3 MANAGING CONNECTIONS TO THE GALEN TERMINOLOGY SERVER 49

3.1 PUPPI 49
3.1.1 Starting the PUPPI 49
3.1.2 Managing Grail Models 50
3.1.3 PUPPI Shutdown 50
3.1.4 Behaviour over a Smalltalk snapshot 50
3.2 TES CONSOLE 50
3.2.1 Server Menu 50
3.2.2 Diagnostics Menu 51
3.2.3 Concepts Menu 51

3.2.4 Language Menu 52
3.2.5 Coding Schemes Menu 52

4 KNOME-LITE-BROWSER 53

4.1 THE KNOME-LITE-BROWSER MAIN WINDOW 53
4.2 BROWSING TOOLS 53
4.2.1 Overview 54
4.2.1.1 A picture of the entire model 54
4.2.2 Concept Browser 54
4.2.2.1 Opening 54
4.2.2.2 Hierarchical display of subsumption relationships 55
4.2.2.3 Criteria 56
4.2.2.4 Description: statements 56
4.2.2.5 Description: annotations 57
4.2.2.6 Concept usage 57
4.2.3 Attribute Browser 57
4.2.4 Source Browser 58
4.2.5 Subsumption Explainer 58
4.2.6 Display interface 58
4.2.7 Word outliner format 59
4.3 EXPLORATION TOOLS 59
4.3.1 Concept Generation 59
4.3.2 Structured Data Entry: the Concept Specialiser 60
4.4 CODING TOOLS 63
4.4.1 Browsing Code Schemes 63
4.4.2 Browsing Code Mappings 64
4.4.2.1 Code Converter Tool 64

5 KNOME-LITE-EXTENDER 66

5.1 THE KNOME-LITE-EXTENDER MAIN WINDOW 66
5.2 TEXTUAL EDITING TOOLS 66
5.2.1.1 Caching 67
5.2.2 GRAIL Source Editing 67
5.2.2.1 Operate menu 67
5.2.2.2 Find / replace 68
5.2.2.3 Formatter 68
5.2.2.4 Syntax checking / pre-compiling 68
5.2.3 GRAIL Explainer 68
5.2.4 GRAIL Workspace 69
5.2.5 Model Editor 69
5.3 EXTENDING NECESSARY PROPERTIES 69

6 KNOME-PRO 71

6.1 THE KNOME-PRO MAIN WINDOW 71
6.2 HARMONISATION AND INTEGRATION 72

6.2.1 Tigger 72
6.2.2 Model Editor 72
6.2.2.1 Overview 72
6.2.2.2 Model Structure Editor 72
6.2.2.3 Model Unit Editor 74
6.2.3 Source Harmonisation 74
6.2.4 Perspective Editor 75
6.2.5 Internal Validation Tools 75

1 Introduction

This document presents an overview and user guide to the modelling environment designed and implemented by the Medical Informatics Group at the University of Manchester as part of the GALEN and GALEN-IN-USE projects: the **GRAIL KnoME** (Knowledge Modelling Environment). The KnoME[6] is a suite of software tools which allow modellers to create, view, explore and maintain a GRAIL concept model maintained in a GALEN Terminology Server. 'The KnoME' is an overall term for three separate, though overlapping, suites of tools, designed to support different kinds of activities, and for use by users with differing degrees of experience and specialised knowledge:

- **KnoME-lite-browser** is intended to support the browsing and exploration of the model, without adding any additional knowledge to the system; it should not demand specialised knowledge of the underlying GRAIL language, nor the specific style of modelling that has been used;
- **KnoME-lite-extender** is designed to support well-controlled additions to the model: either new domains to cover the requirements of a specific application, extending detail and/or coverage in a common area to support the potential requirements of a number of different applications, or adding new properties to areas of the model; this tool also includes functionality provided by KnoME-lite-browser.
- **KnoME-lite-pro** is designed to support the overall building and maintenance of the model: for example integrating and harmonising extensions produced using KnoME-lite-extender. KnoME-pro also provides facilities for supporting the process of mapping elements of existing classification schemes to a GRAIL model; these facilities (provided by 'Tigger') are described separately (in 'Tigger: Overview, Motivation and Interface').

This document is intended to provide an overview to the architecture and requirements behind the GRAIL KnoME (and its organisation into the three suites as described above), as well as describing the individual tools within each suite. It is intended to be suitable for those new to GRAIL, as well as for those with more experience in the 'fine art' of GRAIL modelling. Micro-level help on specific tools will be provided elsewhere; this document is intended to describe what kinds of activities need to be supported, and to describe the style, organisation, and general 'flavour' of the tools that are provided within the KnoME to support those activities.

The rest of **this introductory section** provides an overview of the KnoME as a whole: a description of the overall architecture and implementation of the system within which the KnoME fits. We conclude with a brief note on the history of this document, some details of where to get further information, and how to give us feedback on this document and the software that it describes.

Section 2 provides general information about the Smalltalk development environment within which the GRAIL KnoME has been built and is delivered, which has had some effect on the general user-interface techniques used. It further introduces and describes two logically separate (though commonly physically integrated) interfaces which give overall access to the GALEN Terminology Server (either the TeS or the Puppi) and to the GRAIL KnoME itself. We

[6] pronounced with a silent 'k'

describe the facilities provided in the system for sending feedback to the developers, and for applying any system corrections or developments that we may distribute separately.

The KnoME is not a 'stand-alone' system: it communicates with a GALEN Terminology Server which stores and maintains the model. There are two implementations of server that can be used: a Smalltalk-based TeS and a ROIS-based PUPPI. The KnoME is often operated with a personal TeS, and there are some system-administration facilities which are provided via a 'Terminology Server Console (TeS Console)'. **Section 3** introduces facilities for managing connections to the GALEN Terminology Server.

Sections 4-6 describe the three suites within the GRAIL KnoME proper. We provide an overview of their facilities, then motivate and describe the individual tools with them: in sum, tools for browsing individual elements of that model, for exploring the generative effect of the model, for adding extensions to the model, and for maintaining and integrating the model as a whole.

Readers with some experience of early GRAIL modelling environments, or previous versions of the KnoME may wish to skim or skip sections 2 & 3.

1.1 Architecture

At the heart of GALEN is the **GALEN Terminology Server**: a software system that provides terminological services to different kinds of clinical applications. Within the terminology server, separate modules provide different kinds of services: conceptual, linguistic, coding and other services. Key to all of these is the model of clinical terminology contained within the 'concept module', and represented in the GALEN modelling language known as GRAIL (the GALEN Representation and Integration Language). Clinical applications (such as systems for data entry or protocol authoring) use these services by communicating with the server across a defined API (Applications Programming Interface). This scenario depends on a detailed and structured clinical model existing within the server.

To achieve that model, a large proportion of the effort in GALEN & GALEN-IN-USE has been, and continues to be taken up in providing an environment to enable clinicians to:

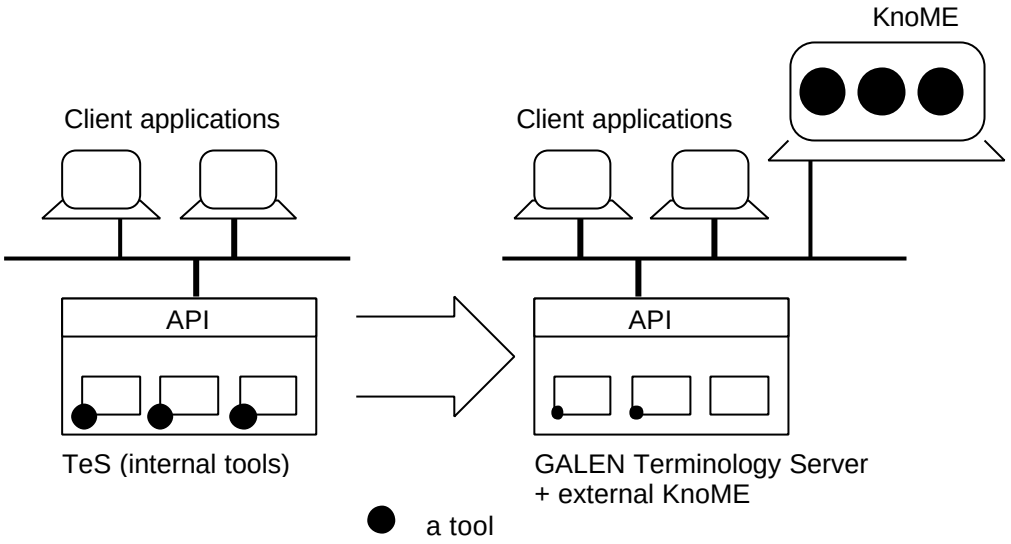
- build and extend such a model (using 'modelling tools'),
- provide the linguistic annotations to support the language module in its task of producing multilingual natural language representations of GRAIL concepts (using 'language annotation tools'),
- produce the mappings to existing coding systems to support the provision of coding services: for example the conversion to and from (and between) codes found within existing clinical classification schemes or terminologies, and GRAIL concepts (using 'coding tools').

Due to the generative and compositional nature of GRAIL models, all these tools are quite unlike the tools that are used to build traditional clinical terminologies.

This document focuses on the first of these: the modelling tools which are provided within the GRAIL KnoME for building, extending and managing the underlying GRAIL model. The KnoME (partly for historical reasons) also includes some tools to support language annotation and code mapping.

Initial versions of all these kinds of tools were built in the same software environment in which the Smalltalk-based TeS was built: the tools were intimately connected with the modules to which they were associated. Together, they formed a tool-set with which modellers built and maintained successive versions of the Common Reference Model, but they did not form a cohesive and coherent (nor particularly maintainable) environment. Other activities with the GALEN project and elsewhere were also involved in building and developing different implementations of the various modules within the Terminology Server, with the aim of being able to 'plug-and-play' different modules within the server, and indeed be able to provide different implementations of the servers as a whole.

Together, these developments mitigated against the provision of the modelling tools as an intimate part of the server software; rather, the tools form an environment for modelling which itself communicates with the server in much the same way as a clinical application, but with a rather privileged kind of dialogue within the server: they have the potential to modify the model rather than just ask questions. This external environment is what we call the KnoME. Its development over time is illustrated below:



From tools within a server, to a KnoME

1.1.1 Overview: the process of 'modelling'

The term 'modelling' is over-used, and describes several different activities. This section describes the scope of the KnoME in terms of the tools provided within it the different kinds of 'modelling' activity that need to take place.

1.1.1.1 Concept Modelling

The KnoME is a 'knowledge modelling environment'. However, 'modelling' means different things to different people. It may refer to:

- building a concept model from scratch;
- extending a concept model for a particular purpose or sub-domain;
- annotating a model in various ways: for example, adding code mappings or linguistic properties;
- exploring a model as part of the task of understanding what it is and what it contains: for example as an application builder;
- integrating and harmonising multiple extensions to a model.

There are overlaps between these activities in terms of the KnoME; one organisation is to divide the tools within the KnoME into those that add knowledge to the system, and those that purely look at the state of the system: editing tools *vs.* browsing tools. This is the distinction that is made between KnoME-lite-browser and KnoME-lite-extender. Building a concept model from scratch will involve browsing tools as well as editing tools.

Editing Tools

At present, most 'adding of knowledge' takes place by the modeller writing textual GRAIL expressions which are then compiled. It is anticipated that more graphically-oriented tools for editing may be developed in the future. We provide facilities for writing 'model units' (collections of textual GRAIL expressions), and organising and storing these units.

Browsing Tools

Whilst not everyone will need to use the editing tools, the browsing tools are ubiquitous. They fall into three main categories:

i. Browsing the overall model: seeing, at a high level, what model is present.

ii. Browsing individual concepts: finding out: what is *x*? What is it a kind of? What does it subsume? What is its structure? What necessary statements have been applied to it? What has it been mapped to? What is its natural language representation? What has been said as a comment about it? or about the way it has been modelled? Is the author happy with it?

iii. Seeing the effect of your model: given this set of elementary concepts and sanctions, what is implied — what could now be said? What would a data-entry form based on this model look like?

KnoME-pro is the tool-set required to pull together the results of concurrent activity by multiple users: to integrate, maintain and manage the developing common resource that is the Common Reference Model.

1.1.1.2 The Coding Module: Code Mapping

There are several aspects to the tools associated with Code Mapping which are available as part of the KnoME:

i. Adding or loading new coding schemes;

ii. Viewing the structure and content of existing coding schemes;

iii. Adding mappings between GRAIL concepts and codes;

iv. Browsing these mappings;

v. Using the functionality of the Coding Module based on all this information.

In addition, the GALEN-IN-USE project has developed the notion of an *intermediate representation* to help the process of relating elements of existing coding or classification systems. This intermediate representation is supported by other tools within the project (notably the Classification Workbench), and its support and development requires an additional set of functionality with KnoME-pro known as Tigger. Tigger is described fully in a separate document 'Tigger: Overview, Motivation & Interface'.

1.1.1.3 The Language Module: Validation and Linguistic Annotation

There are two distinct types of activity that take place in relation to the Language Module. The first is that of validation: checking that the natural language expressions generated by the Language Module for any GRAIL concept are ‘correct’. Support for this is provided within some of the concept modelling tools.

The second type of user-level activity is adding linguistic information into the system with which the Language Module generates textual expressions. The KnoME is not intended to support this activity on a large scale, though facilities do exist to demonstrate the overall techniques: specifically to annotate individual concepts with words in different languages.

1.2 Implementation and Delivery Strategy

The TeS and the KnoME are two logically separate software systems. In a UNIX environment, for example, they can exist as two separate programs (perhaps running on different machines), communicating via sockets. However, they may also be integrated into a single, integrated program called the 'KiT' (KnoME Integrated with TeS).

This is a typical way of releasing the KnoME; it is done for the following reason. Both the TeS and the KnoME are implemented using Smalltalk (specifically VisualWorks 2.5 from ObjectShare, Inc.[7]). One of the advantages of this environment is its platform independence: by providing the TeS and the KnoME integrated into a single Smalltalk program (the KiT), it can be run on any platform supported by VisualWorks 2.5 or later[8]. This is true even if the target platform has no notion of UNIX-style sockets, as all communication between the TeS and the KnoME takes place internally within the KiT. The KiT is delivered as a single 'image file'.

This document assumes that a KiT is being used; it therefore introduces and explains both the TeS Console functions and the KnoME itself. In addition, the KnoME can communicate with an external PUPPI: a ROIS-based implementation for use on Windows platforms: this document also introduces the additional functionality provided to connect to the PUPPI, and to open models within it.

Another effect of the choice of VisualWorks as our development environment is its effect on the style of the user interface of the TeS Console and the KnoME; these aspects are described further in section 2.1.

1.3 Philosophy

There are some general features that we have tried to maintain in the KnoME:

- Re-usability: we have tried to build a series of reusable components to do specific, well-defined tasks, which we then use to build the tools themselves. As well as reducing the amount of raw effort needed to build the KnoME, the advantage of this approach is that a common look-and-feel is easier to achieve. This is carried through to the separation of the KnoME as a whole into three distinct suites (KnoME-lite-browser, KnoME-lite-extender & KnoME-pro) — common functions are provided in each, often extended for particular requirements within KnoME-pro;
- Provide more than one way of doing something: for example, a common task is opening a browser on a concept; we provide several different ways of achieving this. There are two motivations for this: different users will prefer some ways over others, and secondly, different ways will be more useful in different circumstances.

1.4 Document History

The origins of this document date back to 1992, when a guide was produced to describe SMK[9] syntax and compiler operations. Since that time, the guide developed into versions of the 'SET Tools User Guide', released to accompany versions of the Terminology Server and tools software which were produced during the GALEN project, and to Volume H of the GALEN documentation set. These guides provided detailed illustrations of browsing and editing tools that were the fore-runner of the KnoME. Since that time, the KnoME has developed to be a separable environment to the Terminology Server; this document (as of July 1996) was the first description of that development, and of its results. Note that the detailed description of the GRAIL compiler operations themselves are now provided separately.

This document was first released in July 1996, describing [KnoME 1.2c or KiT 1.2].

In April 1997, the document was updated to [KnoME 2 or KiT 2]. Major additional features:

- provides connectivity to the Puppi (ROIS-based implementation of a GALEN Terminology Server) for use on Windows platforms);

---

[7] formerly ParcPlace-Digitalk, Inc.

[8] This includes many flavours of UNIX, Mac & PC operating systems.

[9] Structured Meta-Knowledge - the fore-runner to GRAIL

- new Model Editor which provides support for the structure and management of Grail source files (see section 6.2.2).

In January 1998, this document was updated to reflect the new three-tier structure within the KnoME (KnoME-lite-browser, KnoME-lite-extender & KnoME-pro), supported as an implementation by KnoME version 3. The document was incorporated as an appendix within GALEN-IN-USE Deliverable 9.2.

1.5 Further Information

This document assumes some familiarity with the GALEN project and with the GRAIL modelling language. This level of familiarity is covered by the tutorial introduction to GRAIL, which also introduces some of the tools that are described more fully in this document. The tutorial is also included in GALEN-IN-USE Deliverable 9.2, which describes the motivation and requirements of the KnoME as a whole.

The 'GALEN FAQ' (Volume F) is another useful source of information about GALEN, its products and its services, or see www.galen-organisation.com for further background information. For details of the Common Reference Model that has been produced during the GALEN programme, see GALEN-IN-USE Deliverables 5.1 & 5.2.

These documents, together with other documentation, can be obtained from the GALEN-IN-USE Project Office; contact details are provided on the front cover of this document.

1.6 Feedback

We welcome your feedback on this document, and on the software it describes. Please feel free to contact us[10] for questions, clarification, requests, reports or anything else you'd like to say. Use the on-line tools within the KnoME (see Section 2.6), or more traditional methods: contact details are provided on the front cover of this document.

[10] Email to galen-help@galen-organisation.com

2 The Environment

This section introduces and explains the overall environment within which the TeS and KnoME are built. This includes the Smalltalk development environment, and some generic facilities that we have put in place to support the use of the system.

2.1 Smalltalk

The GRAIL KnoME has been created in the VisualWorks Smalltalk environment; this has some effects on the style of the user interface: many of the KnoME tools inherit various aspects of the VisualWorks development environment, particularly in the use of the mouse.

2.1.1 Mouse operations

The VisualWorks environment, and, through inheritance, many of the KnoME tools, take advantage of three logical mouse buttons:

- 'Select': used for selecting items from a list or menu, pressing buttons, or highlighting pieces of text;
- 'Operate': used to bring up 'floating' menus which are context-sensitive: different menus will appear with an appropriate choice of actions depending on where the mouse pointer is;
- 'Window': brings up a menu to access window manipulation functions such as resize, move, close. Thus menu is typically the same throughout the system, and, whilst it duplicates many of the facilities commonly provided by the host window manager (such as close or resize a window), it adds some extra facilities, such as aiding window navigation by presenting a list of all windows that are open, and opening or bringing forward the selected one.

On a three-button mouse, the standard mappings are:

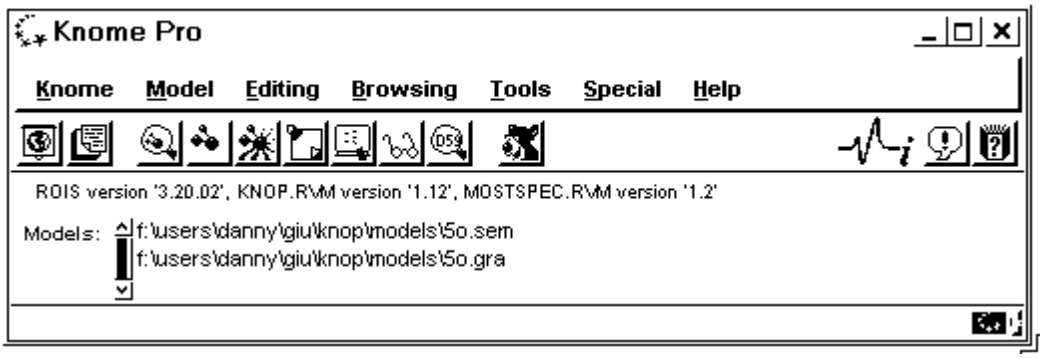
left button	Select
middle button	Operate
right button	Window

On a two-button mouse, the left and right buttons are mapped to Select and Operate respectively; the Window menu is available by using the right button in conjunction with a keyboard character: typically the <control> key.

2.2 Starting and closing the KnoME

The KnoME is distributed either separately as one of its individual components (KnoME-lite-browser, KnoME-lite-extender, or KnoME-pro), or it may offer you the choice of environment when you start the software[11, 12].

In either case, when the software starts you will see a 'Main Window, from which all the functionality in the tool may be accessed. For example, the KnoME-pro Main Window is shown below:



The main purpose for the Main Window is to provide a comprehensive menu bar for access to all tools and functions; in addition, tool-bar buttons are provided to access commonly used functions. The illustration shows a KnoME-pro connected to an external PUPPI version of the GALEN Terminology Server; in this case details of any opened models are also shown in the Main Window. The bottom portion of any Main Window includes a status area where messages are displayed.

When features are described that are common to the Main Windows for either KnoME-lite-browser, KnoME-lite-extender or KnoME-pro, the generic term 'KnoME Main Window' will be used.

2.2.1 Transcript

The menu option Special→View transcript opens a window in which any system messages are displayed.

2.2.2 Snap-shotting the image

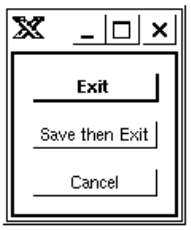
A further facility that is supported is that of 'snap-shotting the image': the entire environment, including whatever KnoME tools are open, and the positions of any windows, together with the state of the server (including any open models) can be saved into a single file, using the KnoME→Save... option from the KnoME Main Window.

[11] See local installation and release notes for details of how to start the software

[12] Note that you may be prompted to supply a hostname and port number if you wish to connect to a separate TeS, or start a TeS which may accept external connections. These parameters may be set using the Preferences tool (see section 2.4). This only applies to non-KiT users. The KiT is the only way to use the KnoME and the TeS on non-UNIX platforms, though the KnoME can be used separately on Win32 platforms in conjunctions with the PUPPI version of the GALEN Terminology Server

2.2.3 Exiting the environment

To exit the KnoME, select KnoME→Exit … To maintain the state of your environment, choose Save then Exit from the resulting dialog (shown right); if you choose Exit, all changes you may have made since you last either started the image, or snap-shotted it (as described above) will be lost.

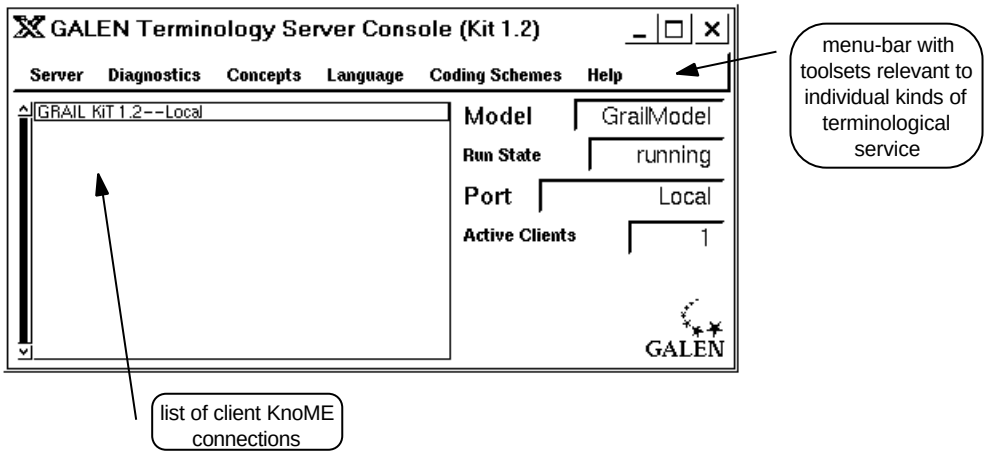


2.3 The TeS Console

This section introduces the TeS Console Main Window. It is the main entry point for the tool-sets controlling the TeS[13].

2.3.1 The TeS Console Main Window

The TeS Console Main Window is described in detail in Section 3. It is used to keep track of connections to the TeS (for example of KnoME 'clients') and has a small set of tools relevant to each internal module. In this document, we will mainly be interested in the Concept Module tools; in particular, the menu option Concepts→Reset Model→Confirm is commonly used during model development to 'throw away' a model to start again.



2.4 Preferences

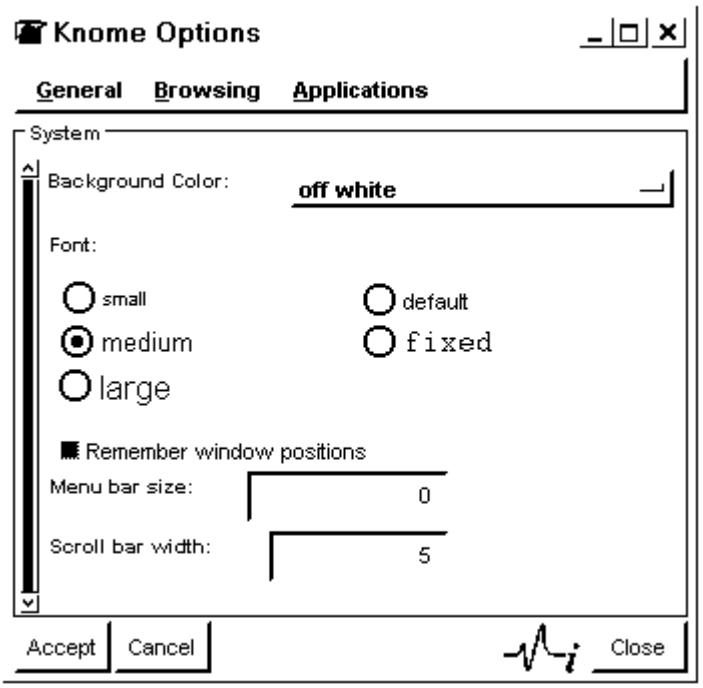
This section has illustrated two 'Main Windows' that are commonly available: the TeS Console for access to and control of the TeS version of the GALEN Terminology Server, and the KnoME Main Window for access to the tool-set which interacts with the server, and which forms the main focus of this document.

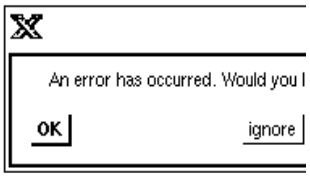
There are a set of preferences and options that are applicable to each of these sub-systems. Control over these options and preferences is achieved via the 'Preferences Tool', available from either the Server→Open Preferences Tool menu option on the TeS Console, or the Special→Options… menu from the KnoME Main Window.

The set of options available from the KnoME Main Window is shown below:

---

[13] recall that the TeS is one of the versions of GALEN Terminology Server to which the KnoME may connect. If it is used in combination with the PUPPI, this section may be skipped.





This shows options applicable to the overall system, such as the font size and background colour for the tools; different panes are chosen using the menu bar to control different aspects of the system (this pane is selected using General→System).

2.5 Help

This document is intended to offer a relatively high-level overview of the tools within the KnoME; it is specifically not intended to offer micro-level help on every menu option available in every tool. Some tools have on-line help giving more detailed information than is maintainable in this document, though this is not yet exhaustive.

We are currently discussing future directions for comprehensive help; a move to a web-based approach is planned for the near future; this would offer the best maintainability, at the potential cost of reliance on a swift network connection[14].

In the meantime, please don't hesitate to contact us at galen-help@cs.man.ac.uk for specific help or advice you may require. Full contact details are given on the cover of this document.

2.5.1 GRAIL Syntax and Operations

The full documentation for the GRAIL operations and textual syntax is presented separately; on-line documentation for the GRAIL compiler is available from the TeS Console's Help menu.

2.6 Problems, Comments and Enhancements

2.6.1 System errors

If a problem should occur with any of the tools you are using then you will see a modal[15] dialog box as shown right.

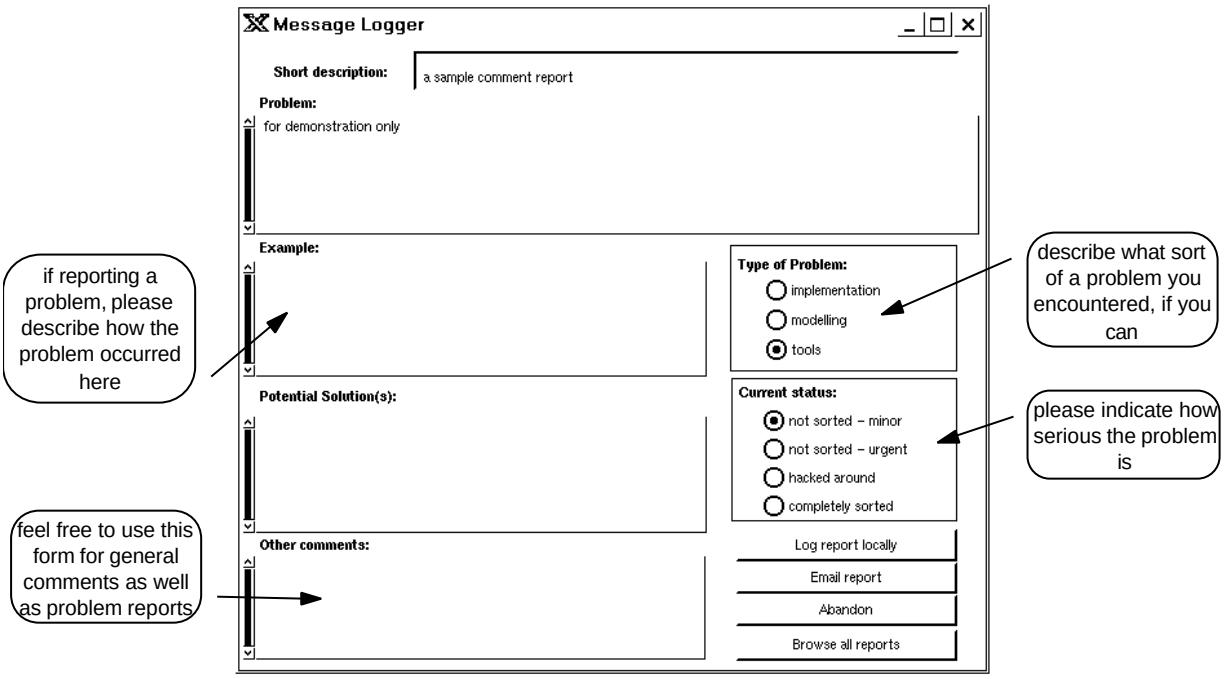
[14] Although this is likely to be mitigated by local cacheing

The normal response is to press the **OK** button (or just hit <return>), which will start an interactive problem reporter as described in the next section.

Alternatively, choosing <ignore> often allows you to continue; <smalltalk debug> enables system debugging which is intended solely for the developers.

2.6.2 Reporting a comment or problem to us

There may be occasions where you may wish to report a problem (as described in the previous section), or make a comment to us about the tools as they stand, or others that you may wish to see in future versions. We have provided an interactive reporting tool within the KnoME, illustrated below:



The tool may be started from the error dialog described in the previous section; you can use the KnoME Main Window's Help→Report Problem menu option, or click the icon shown right.

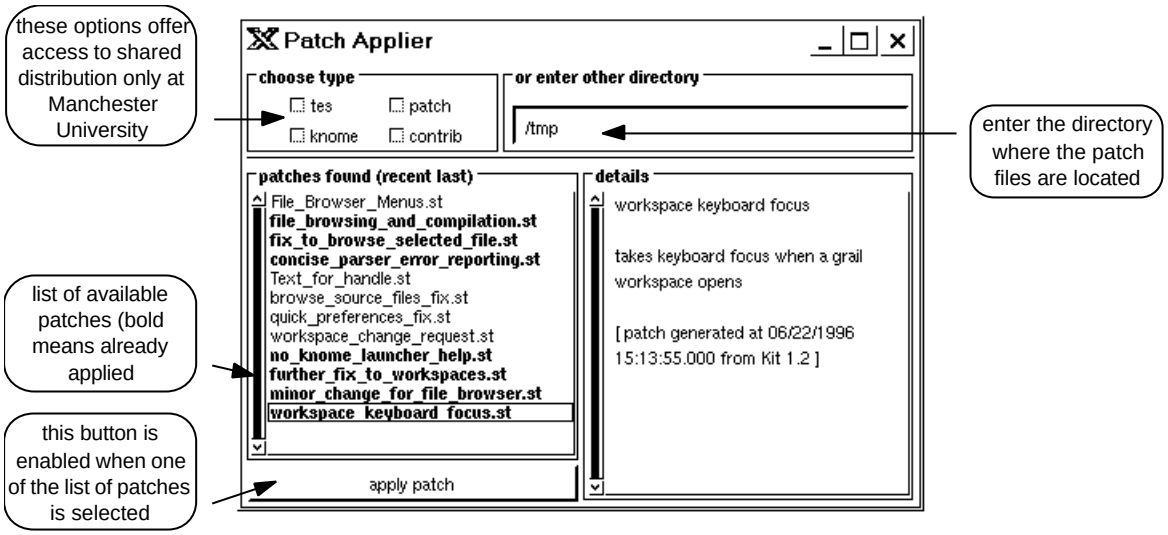


If operating within a UNIX environment, the <Email report> button sends a message directly to us. In other environments, choose the <Log report locally> button which saves each report in a file (location specified using the Preferences Tool) which you can then send to us manually later.

2.6.3 Applying a patch

As a KnoME release is typically a single large (several megabyte) file, we may occasionally respond to problem or comment reports by distributing a 'patch': a small file that can be loaded into the whole environment which will modify or correct its behaviour. To load such a patch, choose the Special→Apply Patch... menu option from the KnoME Main Window. A tool is opened as shown below:

[15] you must choose one of the options presented before moving on.



Having applied one or more patches, save your image as described in Section 2.2. You can see a list of what patches have been applied to your image by choosing Help→About….

3 Managing connections to the GALEN Terminology Server

So far, we have provided an overall introduction to the GRAIL KnoME, and the way in which it is divided into three tiers to support different kinds of activity. In addition, we have briefly described the TeS Console and the KnoME Main Windows.

In this section, we describe facilities provided to manage connections to whichever version of GALEN Terminology Server is in use. If the PUPPI is being used, additional menus are found on the KnoME Main Window. If the TeS is being used, a separate interface is provided (the TeS Console, as introduced in Section 2.3).

Sections 4-6 describes the different tools and facilities that form the heart of the KnoME itself.

3.1 PUPPI

If the KnoME is connected to a PUPPI[16], an additional menu is available on the KnoME Main Window, to manipulate the various models that might be opened by the PUPPI. In addition, the Main Window provides a list of models that have been opened. These models represent GRAIL models, existing classification schemes, language models, and the links that connect all of these together.

3.1.1 Starting the PUPPI

The act of opening a KnoME Main Window starts an external PUPPI on the local machine, if it is not already running. The version number of the underlying ROIS engine is shown in the KnoME Main Window.

Initially, there will be no model open[17]. To compile or build a Grail model using source files, simply create a new model using the menu option Model→New Model… (you will be prompted for a name for it). A new model will be created in the standard models directory[18].

---

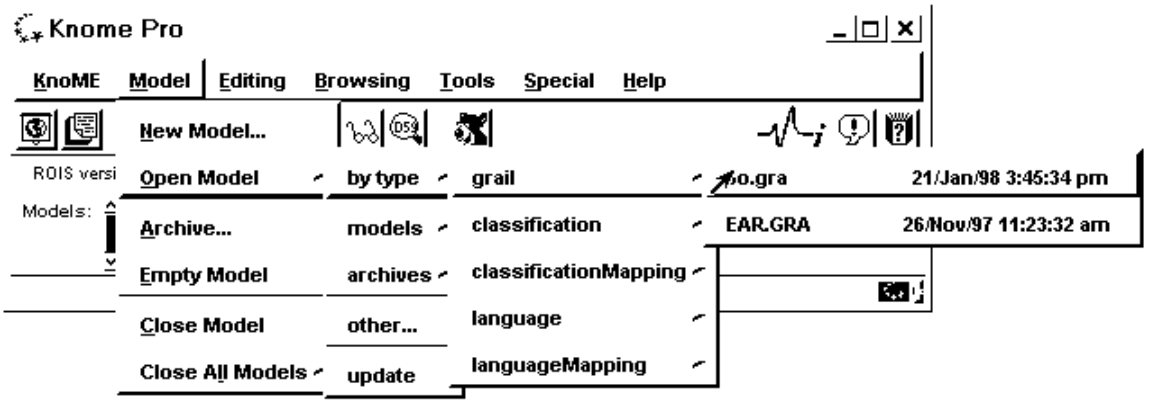
[16] The PUPPI is a ROIS-based implementation of a GALEN Terminology Server developed at the University of Nijmegen; it offers significant improvements in performance on the Windows platform in comparison to the Smalltalk-based TeS

[17] Note that if the image was previously saved with model(s) open, these will be re-opened automatically

[18] this is a directory named models, either in the current working directory, or elsewhere as specified in the ROIS configuration file PROJECTS.INI, which is itself found in the directory where the ROIS executable (ROIS.EXE) is found

3.1.2 Managing Grail Models

The PUPPI provides persistent storage of Grail models: the name of the model provided is the name given to the file. This model, or other, pre-compiled, models, can be subsequently opened using the Model→Open model option. There are 2 well-known places where such model files are stored: directories named 'models' and 'archives'. Newly created or opened models are always opened from the 'models' directories. The 'archives' directories are for backups of various kinds: when a model is opened, a backup of it is created, or a manual snapshot of its current state can be made by using the option Model→Archive. An archived model as well as a normal one can be open via the Model→Open model menu option:



The Open Model option provides routes to open a particular type of model, or any model from either the standard models or archive directory. When an archive model is loaded, it is first copied into the 'models' directory.

3.1.3 PUPPI Shutdown

Whilst the external PUPPI is started automatically, it is not closed down automatically. This must be done manually from within Windows.

3.1.4 Behaviour over a Smalltalk snapshot

The system will detect when VisualWorks is being closed, and remembers the state of any connection to the external PUPPI. When the image is re-started, it will attempt to re-connect and re-open the model that was previously opened; if you are closing an image for moving around to a different machine, for example, ensure that you close any opened model.

3.2 TeS Console

The TeS Console was introduced in section 2.3.1. Here we describe the major facilities available from it, organised according to the menus presented on the Console.

The TeS Console will be available if i) you are using the TeS, and ii) if the TeS that you are using is contained within your image (i.e. you are using the KiT version of the software). If you are using the TeS from a separate KnoME, then the TeS Console will be available wherever the TeS itself is running (in a separate process, perhaps also on a separate machine).

3.2.1 Server Menu

The Terminology Server is organised as a shell around a series of modules, each of which provides different kinds of terminological operations. Each of the following sections describes

functionality relevant to each of these modules; this section describes some overall operations relevant to the operation of the server as a whole.

The Server menu offers options to open a Preferences Tool (described in section 2.4), to adjust the behaviour of the server when errors occur, and to control interaction with any external clients that may be connected. Different levels of logging are available to show the traffic between TeS and KnoME clients; by default, no logging is selected (enabling logging will have a detrimental effect on performance). Any logging information will be displayed on the KnoME Main Window System Transcript area.

3.2.2 Diagnostics Menu

Ideally, the TeS forms an invisible part of the underlying infrastructure (analogously to the computer's operating system). Messages and data go back and forward between the TeS and its clients, with only the end result being visible. The Diagnostics menu option Open Call Meter opens a tool which 'spies' on the traffic over the TeS API. This is useful as a debugging aid, and for demonstrations. Its shows the number and type of calls across the API, and something of the internal communication within the TeS.

The other Diagnostics menu options are internal validation tools for the use of model integrators, and are not commonly needed by newcomers.

The Model Meter provides a snapshot of several kinds of statistics about the current state of the model.

The Concept Meter (if open) is updated constantly as concepts and sanctions are installed in the model.

The Model Checker provides an implementation check; there are several aspects to the underlying software regarding the composition of complex concepts that deliberately (for performance reasons) do not perform exhaustive internal consistency checks continuously. Rather, they can be checked in 'batch mode' using this tool.

The Sanction Checker performs a similar task to the Model Checker, but with regard to sanctions in the model. This tool is of more use to the modeller; it displays any sanctions that are redundant or crossed.

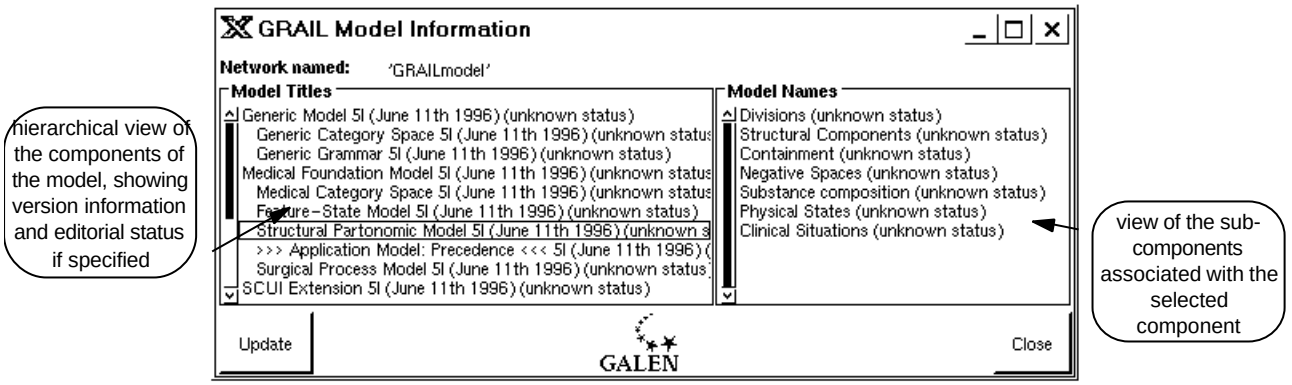
3.2.3 Concepts Menu

The Concepts menu offers several facilities that are needed by the GRAIL modeller. The menu option Reset Model→Confirm completely removes any existing model that may have been compiled into the system; it is used to 'throw everything away to start again'.

Model Names Browser

The option 'Open Model Names Browser' provides a display of what parts of a model are currently present in the system; the illustration below is typical of a system with model 5l loaded[19]:

[19] Note that this display is from 'within' the server itself. A more detailed overview of the model can be found by using the Model Editor tool from within the KnoME, as described in Section 6.2.2



The option File out comments … should be used to produce a textual version of any comments that you have added to the model using the interactive pane with a concept browser (described in section 4.2.2).

To add to, or view the current set of known authors (modellers), use the GRAIL Authors sub-menu.

Sometimes a pre-compiled model will be distributed not as a set of source files, but in an interchange format as a single file. To load this kind of file, use the Open Model Exchange Tool option to open an interface to specify its location, or to generate such a file.

The option GRAIL garbage collect removes (un-installs) from the model any concepts that may be present, but which have no knowledge attached to them.

3.2.4 Language Menu

The TeS includes a Language Module; its primary task is to present multi-lingual natural language expressions for any arbitrarily complex GRAIL concept. Section 4.2.2 describes the concept browser; you can choose to view concepts with natural language expressions generated by the Language Module. The Language Module maintains a set of 'linguistic annotations': words, syntax and other knowledge about how concepts relate to different languages. A suite of tools is available separately to enter such information into the system.

The TeS Console's Language menu offers some support for this task, but is not described further in this document.

3.2.5 Coding Schemes Menu

The TeS includes a Coding Module which offers support for:

managing traditional coding and classification systems;

managing the mappings between codes and GRAIL concepts;

using this information to map arbitrarily complex GRAIL concepts to and from existing coding schemes;

converting between elements of such schemes.

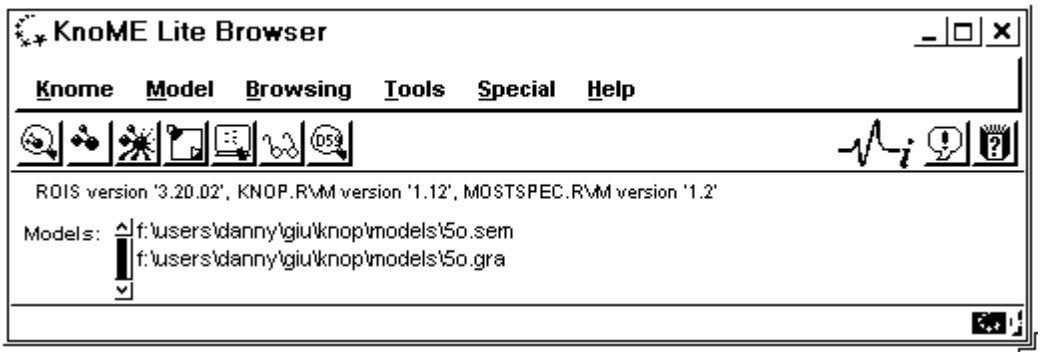
The KnoME provides tools for viewing coding schemes and examining the mappings between coding systems and GRAIL concepts (described in sections 4.4.1 & 4.4.2); the Coding Schemes menu on the TeS Console is for administrative arrangements such as connecting or disconnecting from an external database where the codes themselves are stored.

4 KnoME-lite-browser

This section describes KnoME-lite-browser in more detail. KnoME-lite-browser is a tool for browsing and exploring the model, but does not allow any changes or updates to be made to the model. Note that because GRAIL is a generative system, this does not mean that the internal state of the server may not change; rather, it implies that the internal state of the server only changes as a result of any reification of any concepts that have not previously been required.

4.1 The KnoME-lite-browser Main Window

As introduced in Section 2.2, all the tools and functions provided by KnoME-pro are accessed via the KnoME-lite-browser Main Window, illustrated below:



The menu bar is organised as follows:

- KnoME provides facilities for closing the software (either to switch to KnoME-lite-extender or KnoME-pro if available — using KnoME→Close — or to close the software altogether, using KnoME→Exit…);
- Model is only present if the KnoME is conneceted to a PUPPI; it provides facilities to open and close various kinds of model (as described in Section 3.1);
- Browsing opens tools for browsing and exploring the model;
- Special provides facilities for describing which language is to be used as the default in all the tools, for opening the system transcript if required, and for applying any patches that may be supplied.

The rest of this section describes some of these browsing and exploration tools in more detail.

4.2 Browsing Tools

An essential part of any interaction with the Common Reference Model as a whole, is to get a picture of what is in the model, what structure is there, and how it is inter-related.

This section describes the tools that are available in the KnoME for *browsing* the model. The next section introduces tools for *exploring* what is possible in the model: necessary due to the compositional and generative nature of GRAIL models.

4.2.1 Overview

There are several different dimensions to the notion of 'browsing the model'. One is that of getting an overall feel for the model: a high level picture. Another is seeing details of elements within the model at some detail: what is this concept? What is its structure? What is it related to? Where is it used? How does it relate to codes in existing coding schemes? What is a natural language expression for it? Most of the tools described in this section explore this aspect of browsing.

4.2.1.1 A picture of the entire model

It is a non-trivial task to provide an effective interface to modellers describing the model at varying level of detail, and we certainly haven't solved the problem. There is on-going work here on 3-D visualisation techniques, and of more traditional presentations of different kinds of information. At present, the Model Names Browser (if connected to the TeS) offers an overview of the named sections of the model, to give some idea of the sorts of things that you may expect to find there. Another technique is to use the concept browsing tools on high-level parts of the model, or to use the Model Editor as described in section 5.2.5.

4.2.2 Concept Browser

The GRAIL Concept (or Knowledge) Browser is probably the most heavily used browsing tool in the KnoME. It is always focused on a single concept or attribute (its 'subject', or 'focus'), and shows two distinct sets of features:

- the position of the focus with respect to the rest of the model; i.e. how it lies in the subsumption ('is-kind-of') relationship that is maintained automatically by the server;
- structural details of the focus itself: what is its structure, what properties have been applied to it.

In addition, it combines these two aspects by showing, for example, properties which apply to a concept due to inheritance.

4.2.2.1 Opening

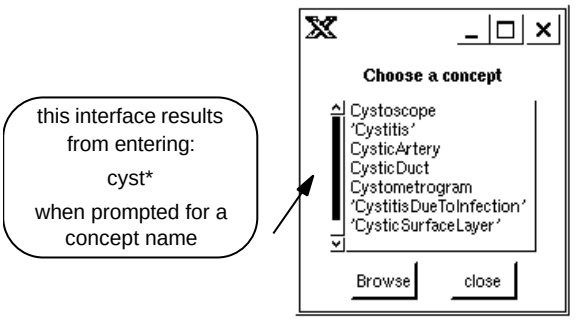
There are several ways to open a Concept Browser:



- Press the icon on the KnoME Main Window that looks like a magnifying glass, shown right:
- Choose the KnoME Main Window option Browsing→Knowledge browser…, and type a name of a concept or attribute (wild-cards are accepted);
- Type a GRAIL expression in a GRAIL Workspace (section 5.2.4) or other model unit area and use the Operate mouse button menu option Browse;
- Press the Browse button on a GRAIL Explainer (section 5.2.3);
- Use any of the 'Browse' options on the Operate mouse button menu options which are available from many of the browsing tools which show lists of concepts (perhaps organised in hierarchies), or of criteria (attribute-value pairs).

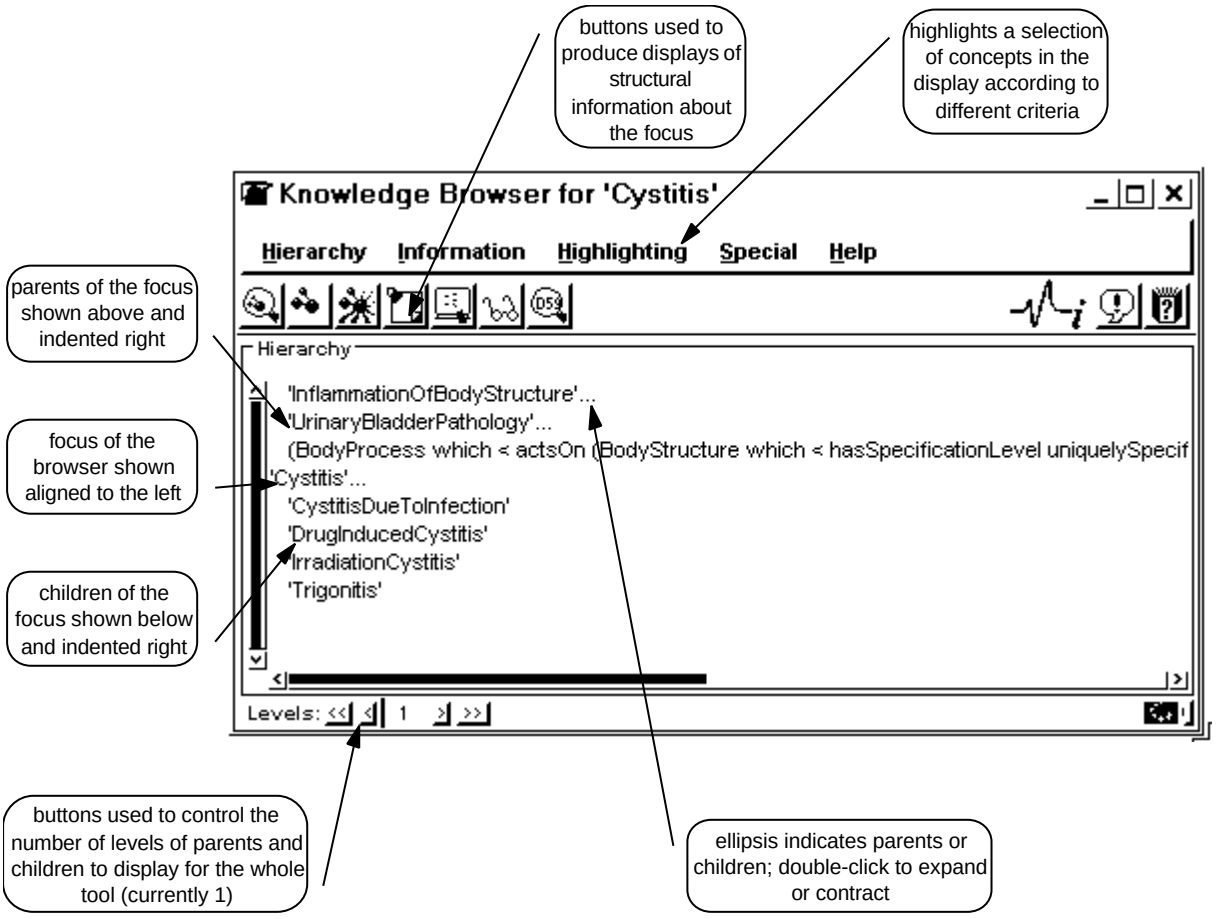
Sometimes it is useful to have lists of concepts available which you can then browse easily. Normally, using a wild-card when prompted for a concept name will pull up a menu with matching names that you can choose from; having selected one, the menu closes. To have this list permanently available, choose the option Use Action Lists from the Browsing pane of the

KnoME Options Tool (described in Section 2.4). Then, after any wild-card, the following interface will stay open:



4.2.2.2 Hierarchical display of subsumption relationships

In normal use, opening a Concept Browser provides the following interface containing a view of the immediate area of the model around the focus, together with buttons used to obtain further structural information about the focus:

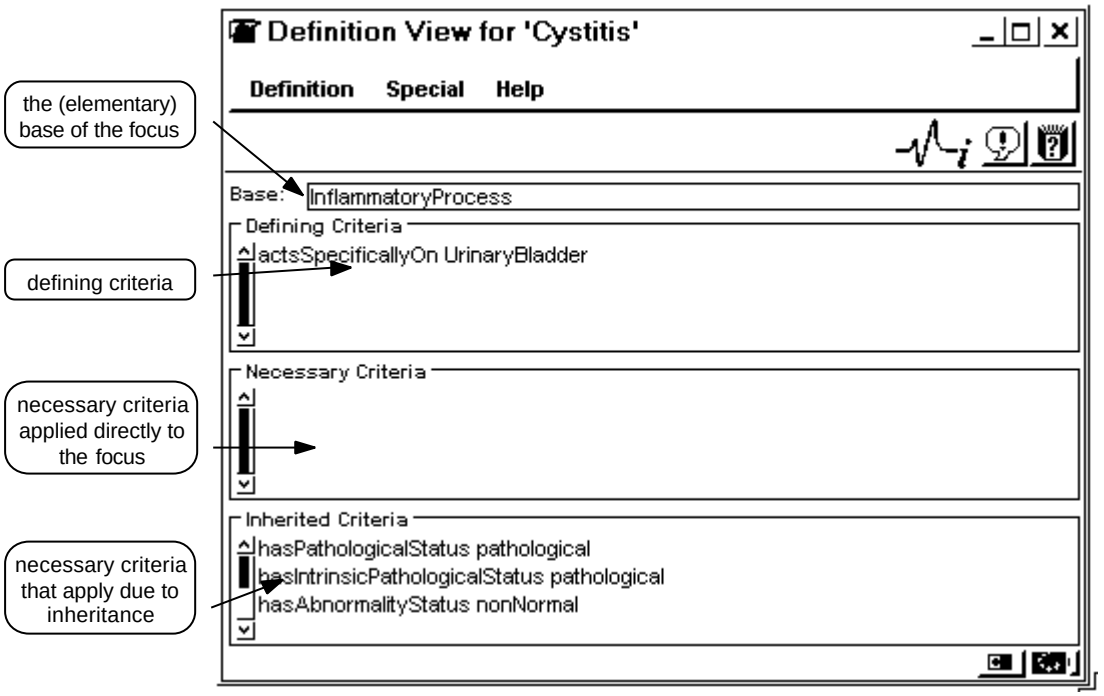


The Operate mouse button menu over the list of criteria offers many options for spawning other browsers described later in this section.

Concepts within the hierarchy view are printed using GRAIL names by default. The name will be in quote marks if it is a name provided for a complex concept (rather than an elementary concept).

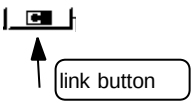
4.2.2.3 Criteria

To obtain information about the defining and necessary criteria that apply to the selected entity in a concept browser, press the Information→Definition option from the window menu of a concept browser, to produce the following display:



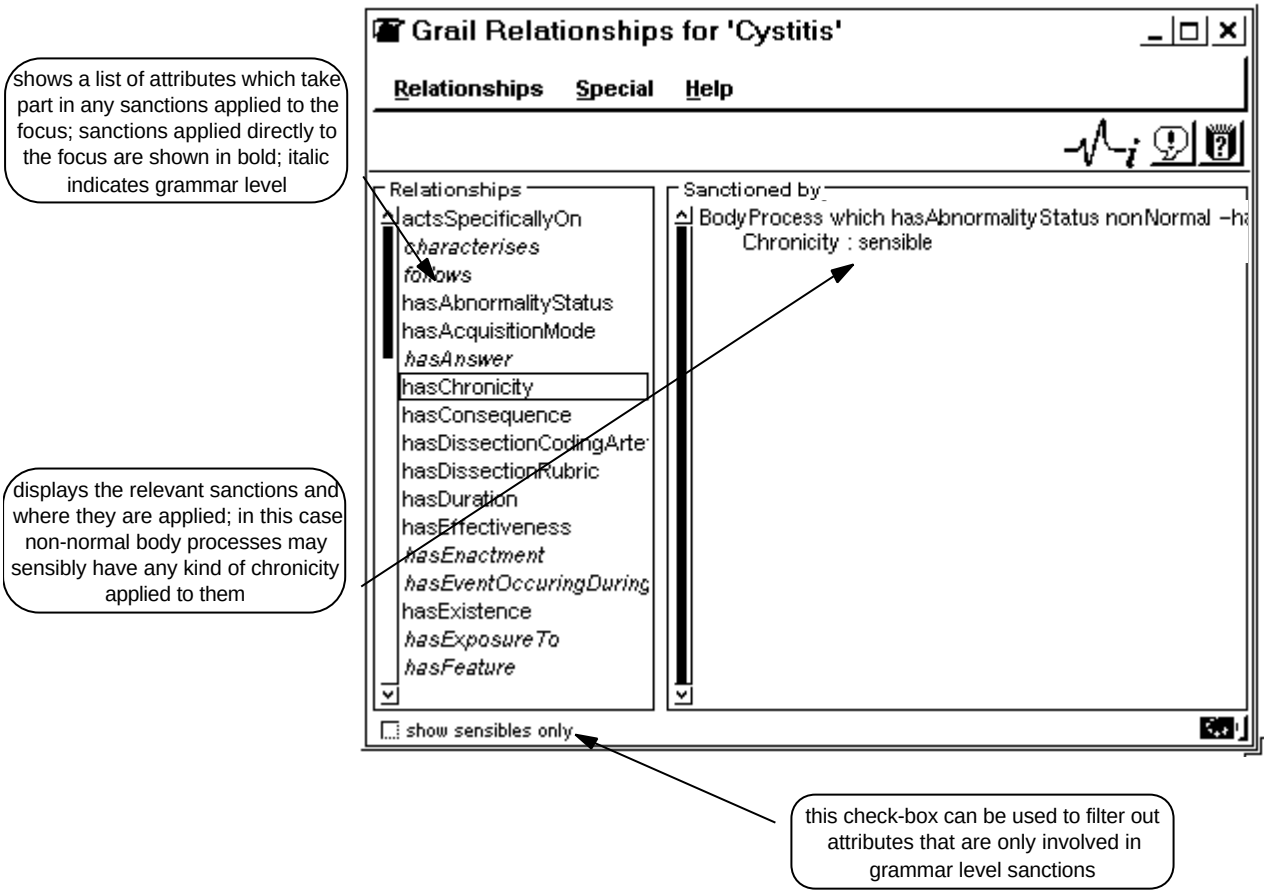
To see where any inherited criteria are inherited *from* (known as the *source*), use the Operate mouse button menu options over the list of inherited criteria.

Many tools that are 'spawned' from other tools in this way often contain a special button in the bottom right of their display. This is the 'link' button, used to specify whether it is linked to the originating tool (i.e. if the focus of the original tool changes, should this spawned tool update to reflect the change).



4.2.2.4 Description: statements

Other important information that can be made visible is the set of sanctions that have been applied to the focus or its parents. Choose the button Information…→Relationships to produce the display illustrated:



4.2.2.5 Description: annotations

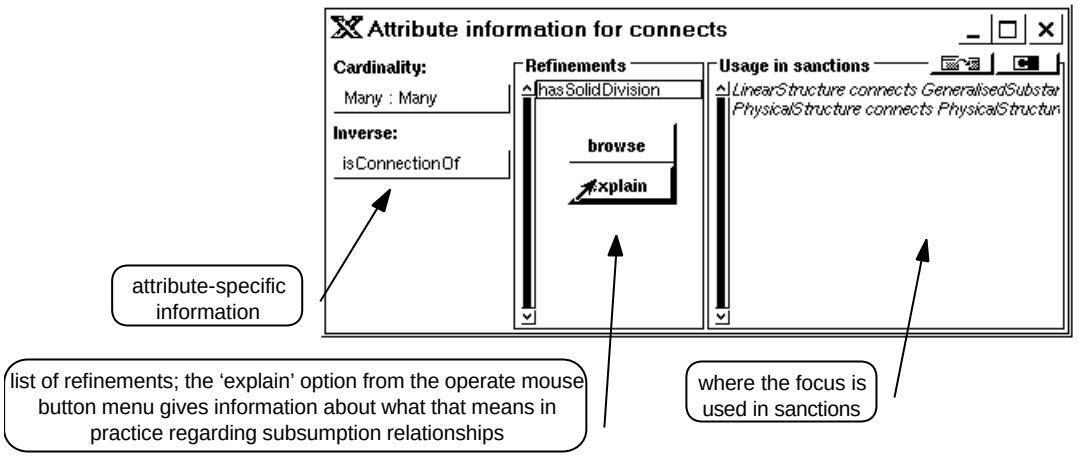
There is some information attached to concepts that is not strictly terminological in the GRAIL sense, but is important in navigating around the model; seeing its status and providing help in its (human) interpretation. Use the description→Annotations button to display an interface providing information about editorial status, editorial comments and external comments. Editorial comments are intended to provide information to modellers, whereas external comments are designed to be used by end-users. The editorial status flag can be used to get an indication of how fixed a particular area of model is considered to be.

4.2.2.6 Concept usage

To obtain an indication of where a particular concept is used, choose the Operate mouse button menu option display usage from the main hierarchy view. This tool shows where the concept is itself used as part of the defining or necessary criteria, or within sanctions.

4.2.3 Attribute Browser

The previous section described the Concept Browser in its common role of examining concepts whose subsumption relationships are maintained by the system. GRAIL attributes have many similar properties to concepts, in that they are also maintained in a hierarchy (though one managed by the modeller explicitly). Entering the name of an attribute rather than a concept will open an attribute browser whose main hierarchy display is similar to the standard Concept Browser. However, in place of the Definition and relationships options, there is an attribute info … option, which opens the following display:



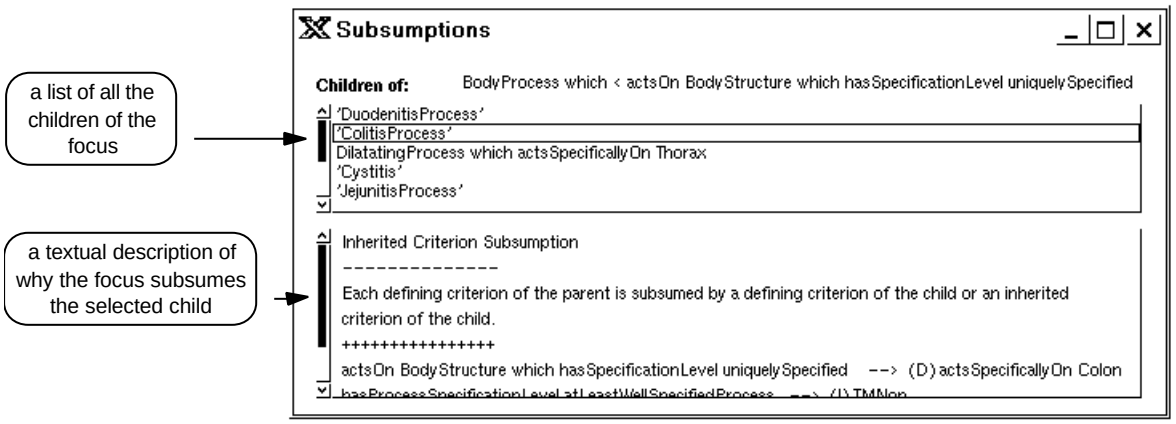
4.2.4 Source Browser

So far, we have described tools that show the structure and properties of concepts and attributes as they exist having been compiled into the system: divorced from their original textual GRAIL source. However, sometimes it is useful to be able to go back to the original GRAIL source to see where concepts or attributes are used (and occasionally extra comments might be available).

The Operate mouse button menu option browse source files is available from a GRAIL model unit workspace or from the hierarchy view of a browser. In either case, a Model Unit Editor (as described in Section 6.2.2.3) is opened containing just those model units where the focus has been used (the first in the list is where the focus was first defined).

4.2.5 Subsumption Explainer

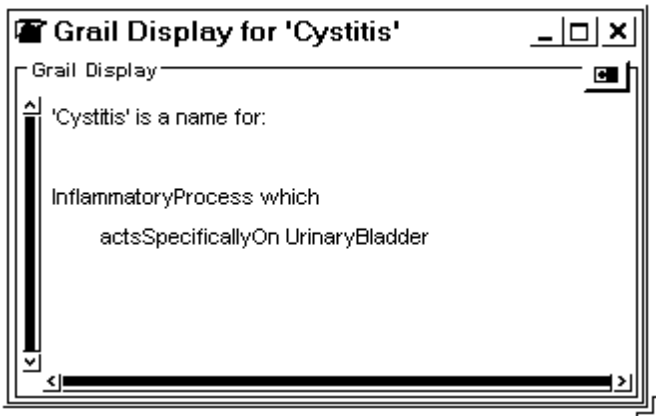
The subsumption relationships between concepts in the model are maintained automatically by the system; they are frequently complex, and take into account defining and necessary criteria, and also refinement properties applied to attributes. The combination of these factors means that sometimes it is not immediately clear *why* some concepts do subsume (as shown within the hierarchy view of a concept browser). There is a menu option Information→explain children which opens a tool designed to explain why concepts subsume:



4.2.6 Display interface

The browse menu option is ubiquitous within the KnoME, for opening new browsers from any interface element where lists of concepts or criteria are displayed. Commonly, there will also be

a display option, and it is also available directly from a button on a Concept Browser. It opens a small tool displaying the complete detail of the internal structure of the focus. For example:



4.2.7 Word outliner format

The hierarchy view within a concept browser enables you to navigate around the subsumptions within the model. This activity has some things in common with the task of managing outlines of documents using, for example, Microsoft Word Outliner mode (with at least one major difference that the Word header levels are mono-hierarchical). There is an Operate mouse button menu option from the Concept Browser hierarchy view labelled save→hierarchy in Word outliner format... which generates an RTF file of the model currently displayed in the browser, which can then be examined separately.

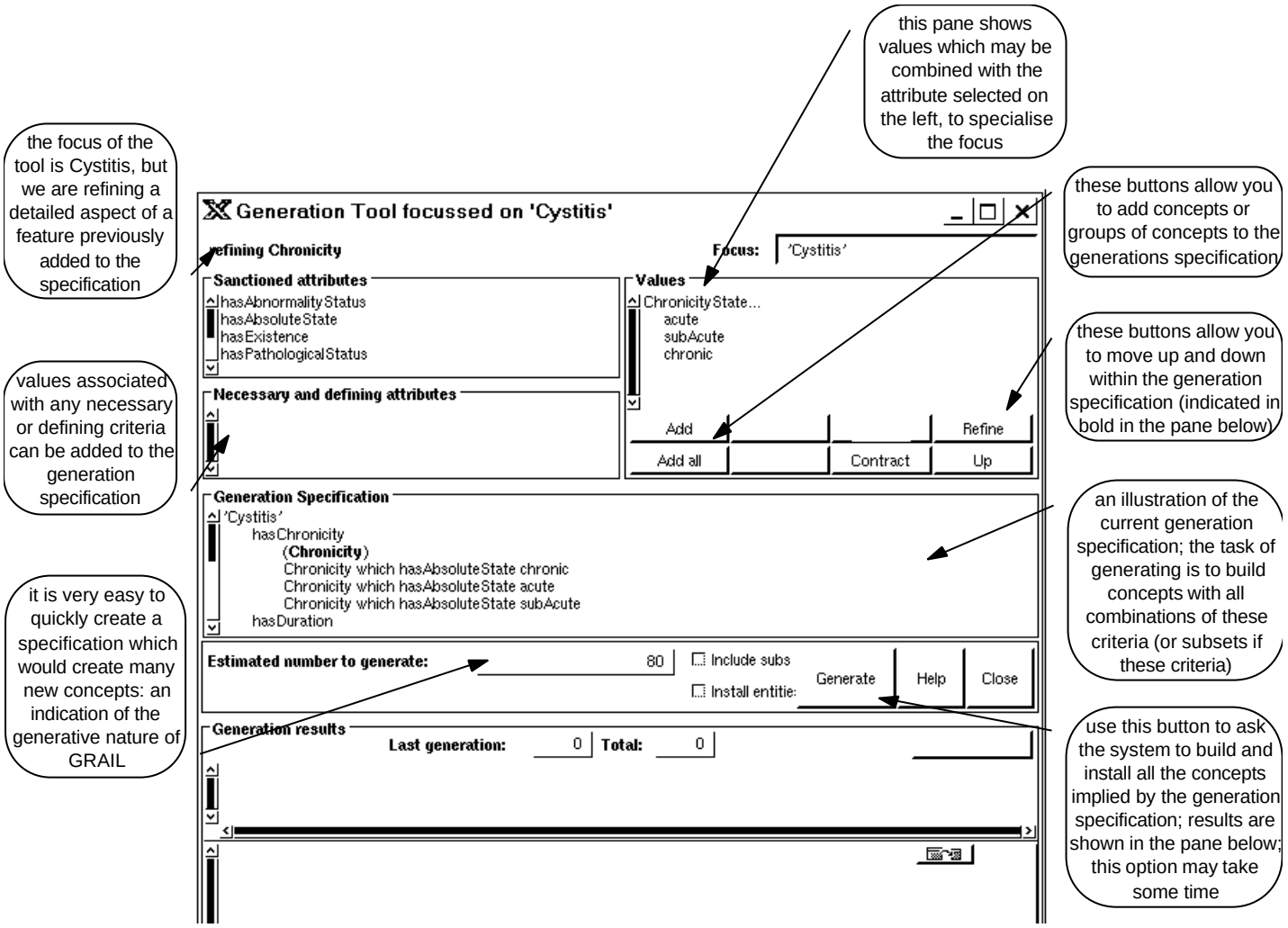
4.3 Exploration Tools

The previous section introduced some tools which offer a detailed, structural view of the model. However, the tools offer a view that is relatively close to the implementation structure of the model held within the TeS; they don't take into account the generative nature of GRAIL models. The tools described in this section are specifically designed to overcome these issues. The Concept Generator provides an interface for modellers to be able to manually 'fill out' some of the concepts that are implied by the model but which do not need to be installed normally. The Concept Specialiser is intended to provide a sample data-entry form of the style intended to be produced by clinical systems.

4.3.1 Concept Generation

To open a concept generator, use the menu option Special→Generate from here from a concept browser hierarchy view. Alternatively, use the KnoME Main Window menu option Tools→Generation Tool or press the icon shown right.

The top area of the tool is used for specifying which area of the model you would like to generate within; the bottom of the tool displays any newly-installed concepts.

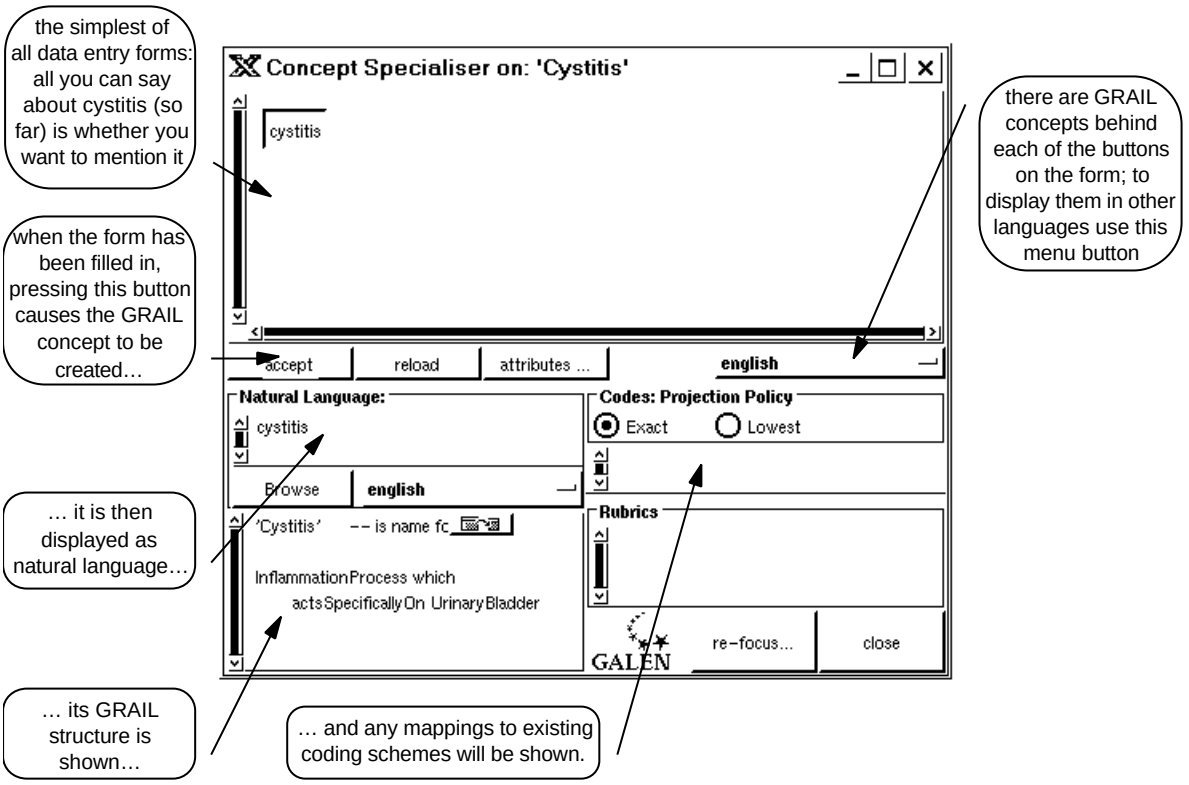


4.3.2 Structured Data Entry: the Concept Specialiser

A prime motivation behind the development of GRAIL has been to support structured data entry for clinicians. The Concept Specialiser is a tool which produces example data entry forms; it is useful for demonstration, but it is also a good way of having an overview of what the model allows, rather than just what it is; the form is built directly from the model, so if there are elements of the form which are 'incorrect', that might imply that the model needs review. Both the Concept Generator and the Concept Specialiser are thus useful validation tools.

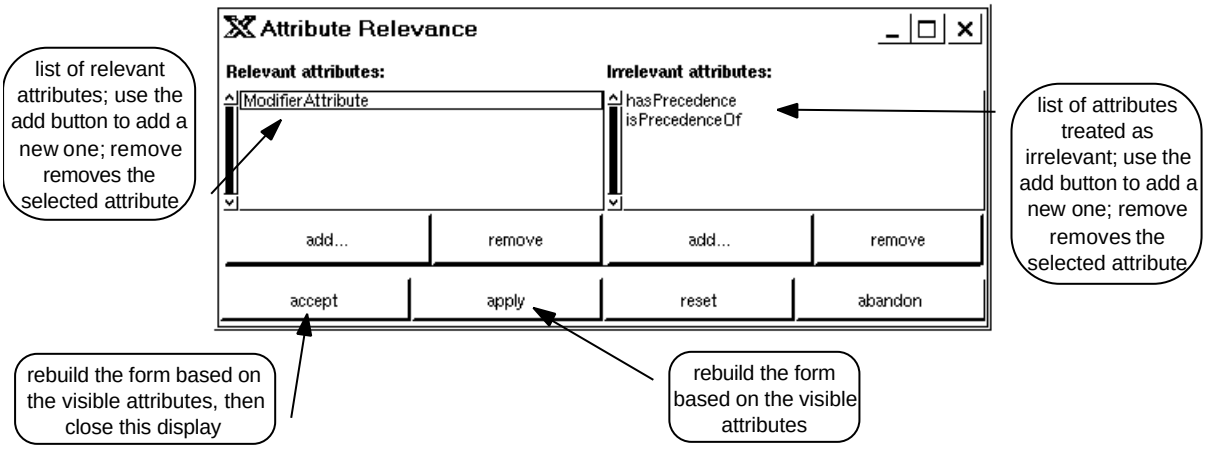
To open a Concept Specialiser tool, choose the KnoME Main Window option Browsing→What Can I Say About…

Initially, the tool will open as shown below[20]:



The form is empty because we have not yet specified which attributes we wish to be treated as relevant. This is necessary because the model is intended to be application-independent: any model useful for more than one area of use will, by necessity, have too much detail for any single application.

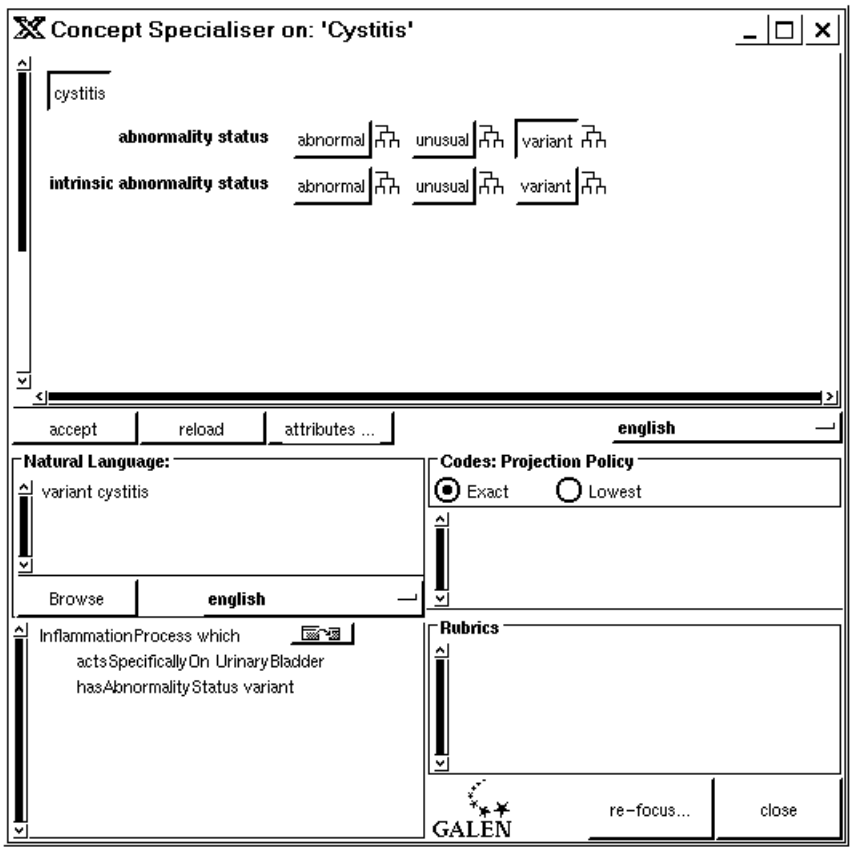
The specification of which parts of the model are relevant to us is controlled in this tool by selecting 'relevant' attributes, and de-selecting 'irrelevant' ones. This is done by using the interface opened by pressing the attributes … button, as illustrated:



[20] The current version expects particular concepts to exist within the model, which are present in the CORE Model. If you experience any difficulty with your own models, try compiling the following GRAIL source code (this inconvenience will disappear in future versions):

TopCategory newSub Feature.
Attribute newAttribute hasPrecedence isPrecedenceOf manyMany.

The settings shown (in the context of the GALEN CORE Model version 5l) will result in a form as follows:



4.4 Coding Tools

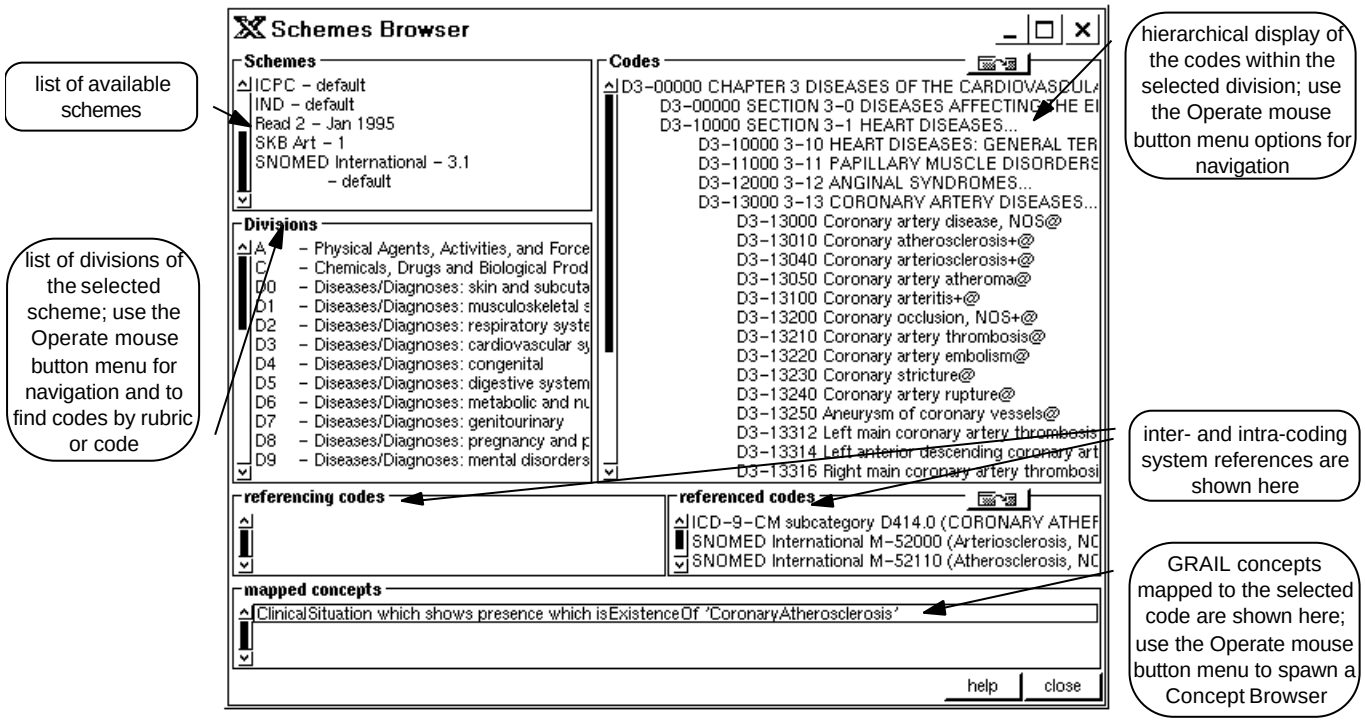
Much of the work in the GALEN & GALEN-IN-USE projects has been to do with the relationship between the GALEN concept models (a system of terminology based on GRAIL), and existing coding and classification schemes. So far, we have considered the KnoME browsing tools as they relate to the concept model alone; in this section, we describe the facilities within the KnoME for examining and mapping between GRAIL concepts and elements of existing coding and classification schemes.

The large-scale schemes (such as ICD or SNOMED) are stored externally to the server in a database[21]; adding new versions is a task for the local system administrator. However, even without a local scheme database installation, we can cache a selection of schemes and codes that are mapped to GRAIL concepts within the KiT environment.

4.4.1 Browsing Code Schemes

The KnoME includes a browsing tool for existing coding and classification schemes. Use the KnoME Main Window option Tools→Browse Coding Schemes, or press the button shown right, to open a Schemes Browser as shown below:

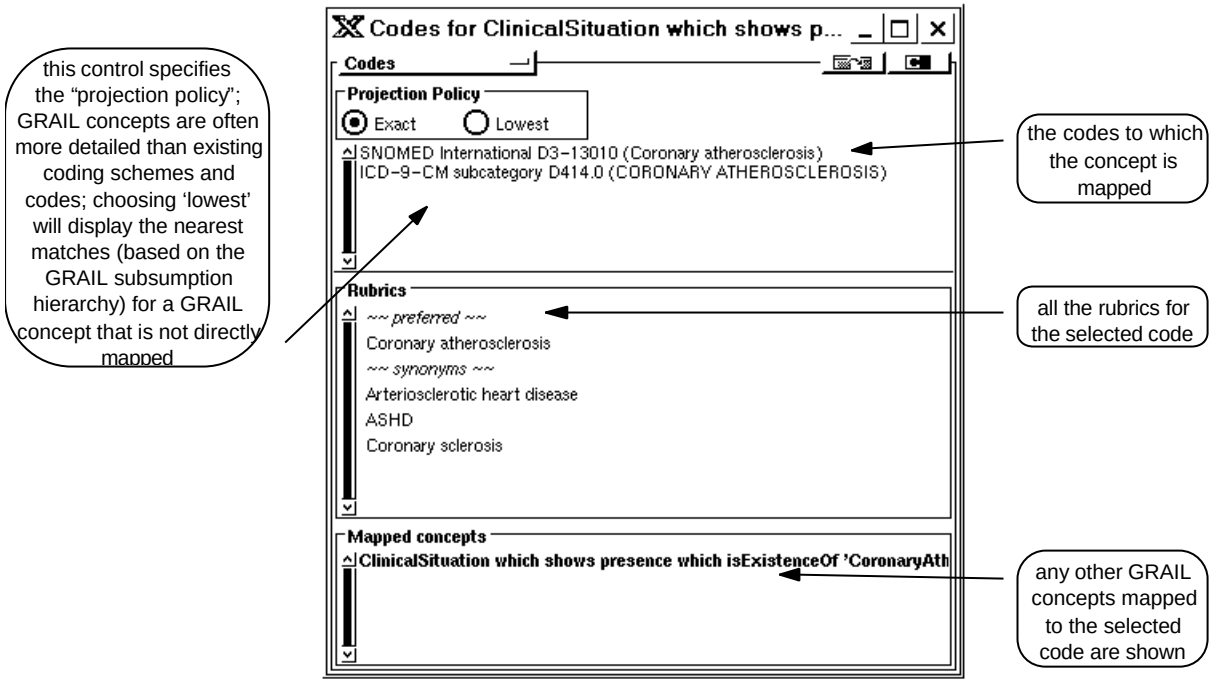




[21] or in the case of the PUPPI, stored as separate ROIS model files

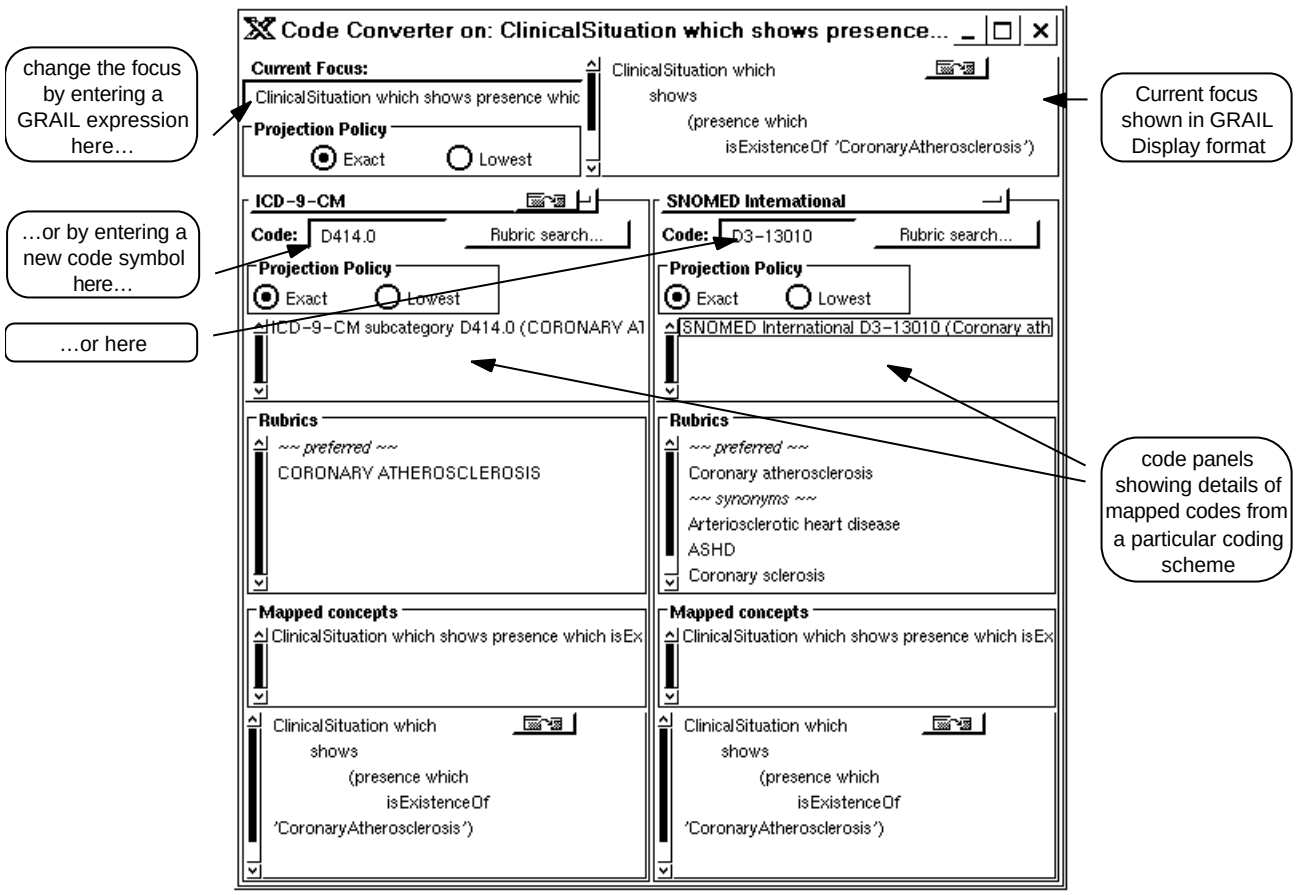
4.4.2 Browsing Code Mappings

The illustration of the Schemes Browser shown above includes a pane which shows any GRAIL concepts mapped to a selected code. In the inverse direction, there are facilities from a GRAIL Concept Browser (option Information→Codes) to display any mapped codes. For example:



4.4.2.1 Code Converter Tool

There is a separate tool known as the Code Converter for interactive exploration of the mappings between codes and GRAIL concepts. It can be opened by choosing the KnoME Main Window option Tools→Code Converter on….

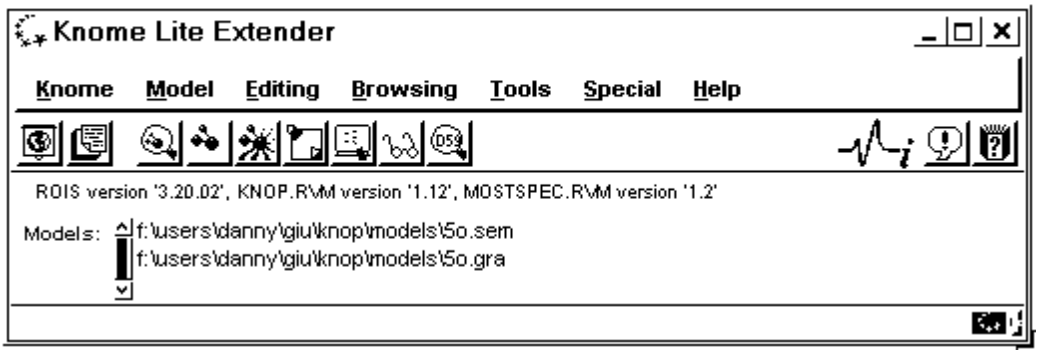

change the focus by entering a GRAIL expression here…
…or by entering a new code symbol here…
…or here
Code Converter on: ClinicalSituation which shows presence...
Current Focus:
ClinicalSituation which shows presence whic
Projection Policy
Exact
Lowest
ClinicalSituation which
shows
(presence which
isExistenceOf 'CoronaryAtherosclerosis')
Current focus shown in GRAIL Display format
ICD-9-CM
Code: D414.0
Rubric search...
Projection Policy
Exact
Lowest
ICD-9-CM subcategory D414.0 (CORONARY A
SNOMED International
Code: D3-13010
Rubric search...
Projection Policy
Exact
Lowest
SNOMED International D3-13010 (Coronary ath
Rubrics
~~ preferred ~~
CORONARY ATHEROSCLEROSIS
Rubrics
~~ preferred ~~
Coronary atherosclerosis
~~ synonyms ~~
Arteriosclerotic heart disease
ASHD
Coronary sclerosis
code panels showing details of mapped codes from a particular coding scheme
Mapped concepts
ClinicalSituation which shows presence which isEx
Mapped concepts
ClinicalSituation which shows presence which isEx
ClinicalSituation which
shows
(presence which
isExistenceOf
'CoronaryAtherosclerosis')
ClinicalSituation which
shows
(presence which
isExistenceOf
'CoronaryAtherosclerosis')

5 KnoME-lite-extender

The previous section introduced KnoME-lite-browser, with its tools for browsing and exploring a GRAIL model. KnoME-lite-extender includes all that functionality, but also includes features for extending the knowledge within the model.

5.1 The KnoME-lite-extender Main Window

As introduced in Section 2.2, all the tools and functions provided by KnoME-lite-extender are accessed via the KnoME-lite-extender Main Window, illustrated below:



The menu bar is organised as follows:

- KnoME provides facilities for closing the software (either to switch to KnoME-lite-browser or KnoME-lite-extender if available — using KnoME→Close — or to close the software altogether, using KnoME→Exit…);
- Model is only present if the KnoME is conneceted to a PUPPI; it provides facilities to open and close various kinds of model (as described in Section 3.1);
- Editing opens the tools for creating, and extending GRAIL models;
- Browsing opens tools for browsing and exploring the model;
- Tools provides facilities for exploring the generative nature of the model, and for creating and viewing perspectives on the model;
- Special provides facilities for describing which language is to be used as the default in all the tools, for opening the system transcript if required, and for applying any patches that may be supplied.

The rest of this section describes some of these tools in more detail.

5.2 Textual Editing Tools

The primary means of adding knowledge into the system is to write and compile textual GRAIL expressions. The expressions are written as 'model units' which are stored as files on the host

file system. The KnoME editing tools include a GRAIL compiler and syntax checker, described in the next section, and tools for managing model units in files. The model units may contain comments about the model, and meta-level information such as authorship, as well as the expressions which add, for example, new attributes or elementary concepts into the system.

KnoME-pro includes further facilities to help manage the storage of model units, and the maintenance of dependency relationships between different parts of the model.

The set of textual GRAIL model units, contained within files, is not the same as the 'model'. The model is normally thought of as the result of compiling these model units (using the KnoME tools described next) into the TeS. That compiled model within the TeS can (and often does) exist and be useful without the original GRAIL source model units; however, the model units are often distributed with a compiled model as additional documentation about that model.

5.2.1.1 Caching

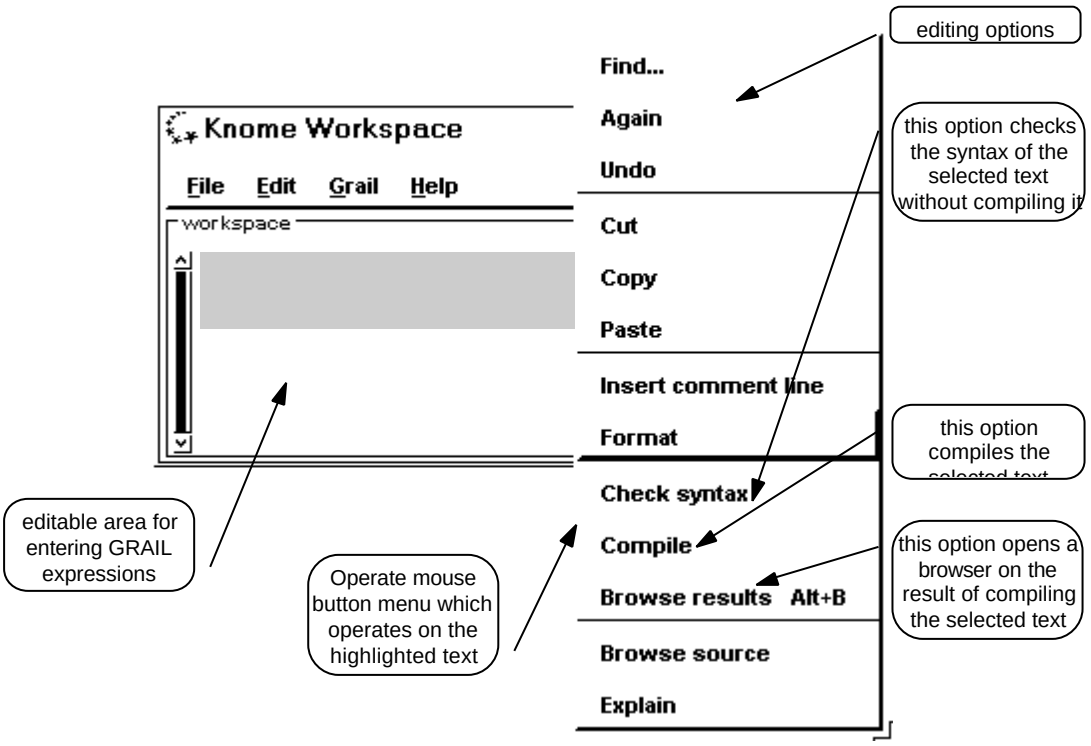
Many of the tools described in this section, and in the KnoME-lite-browser tools, cache information that they get from the server (for performance reasons). Occasionally, it may be necessary to manually refresh these caches if you add information into the model using the editing tools; this is done by choosing the KnoME Main Window option KnoME→Flush Cache. This inconvenience will be removed in future versions.

5.2.2 GRAIL Source Editing

This section describes facilities which are common to the writing and compilation of GRAIL model units; these units might be individual files, or as part of a wider collection of 'model units'.

5.2.2.1 Operate menu

A model unit is visible on the screen as an editable text area; there are many functions available from the Operate mouse button menu which operated on the highlighted portion of text: highlighting is performed using the Select mouse button, as illustrated below:



5.2.2.2 Find/replace

The Operate menu option find... opens a find/replace editing tool which operates within the current model unit.

5.2.2.3 Formatter

The Operate menu option Format re-formats the highlighted text: it adjusts indentation and layout to make the source text conform to a common style, which makes it especially easy for others to read your source. It also shows GRAIL comments in bold text; any strings as italics, and GRAIL keywords are underlined.

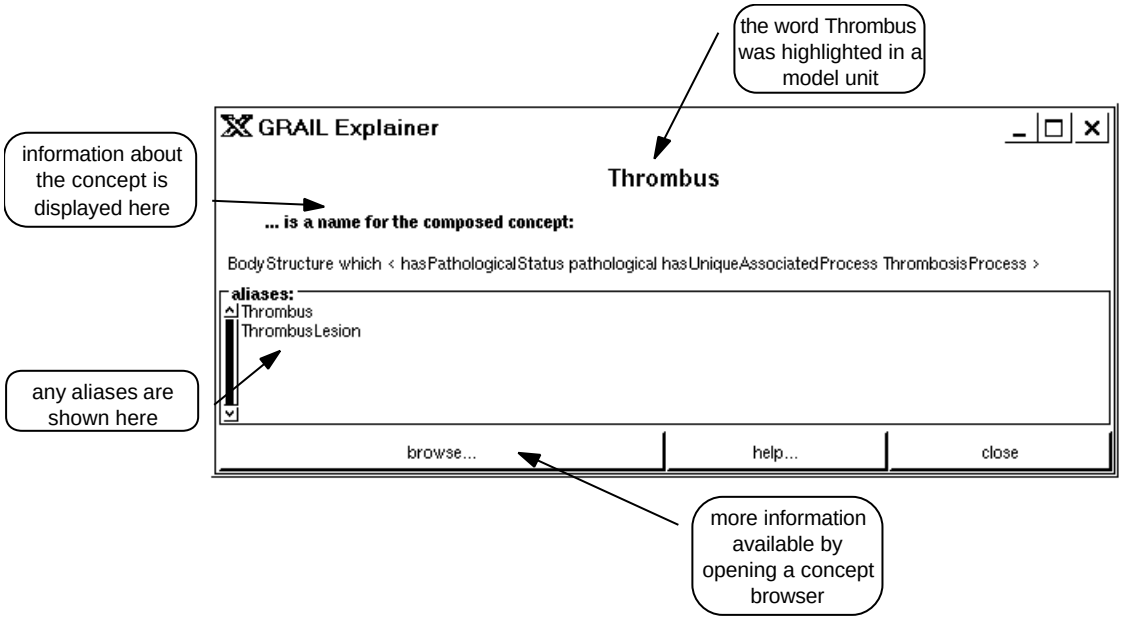
5.2.2.4 Syntax checking/pre-compiling

It is often the case that before compiling the contents of a model unit, you may wish to check the syntax; the Operate menu option Check syntax performs this action, and also performs a 'pre-compile' check: this does various consistency checks; for example, whether all the GRAIL names used have been previously defined.

Pure syntax errors (for example, the wrong number of arguments in a GRAIL expression) are often reported by inserting a message into the model unit at the place the error was found. Pre-compilation errors are reported by opening a separate window containing details of any errors or warnings. GRAIL errors may still occur during compilation (for example, sanctioning failures) which are reported using model error reporters (which also offer a quick route to opening a problem reporter).

5.2.3 GRAIL Explainer

The Operate menu option explain... can be used when any single word in a model unit is highlighted: if the highlighted word represents a GRAIL concept or attribute, some details about it are presented, as illustrated below; if the word is a GRAIL operation keyword some documentation is provided for it.



5.2.4 GRAIL Workspace

KnoME-lite-extender is often used to build extensions to the model. The 'GRAIL Workspace' is used to create a unit that can be submitted as a text-file to a central integration site, or simply to create a small new model. Often, it is also useful to have a model unit area available for 'scratch' work, or for entering concepts to be browsed. To open a 'GRAIL Workspace', use the KnoME Main Window Option Editing→Workspace or press the icon as shown right.



It open the tool as illustrate above in Section 5.2.2.1. Options are available to save the contents of the workspace to file, load the contents from file, and to check and compile the highlighted contents of the workspace.

5.2.5 Model Editor

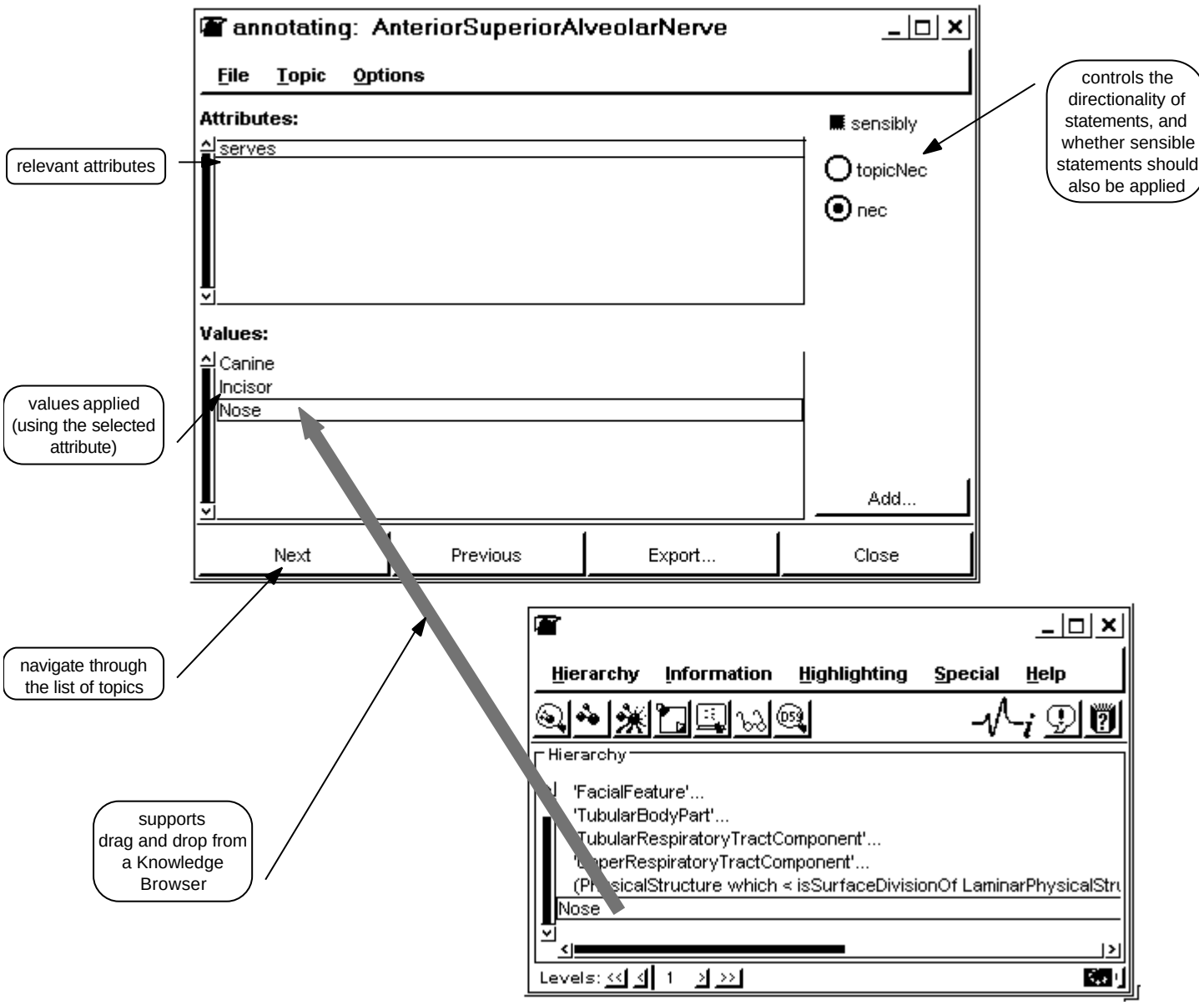
The Model Editor provides a more controlled environment for extending the model, and guides and enforces additional features such as status-annotation, etc., and provides an overall richer editing environment. Sections of model created using the Workspace, as described above, may well be harder to integrate at a different site. The Model Editor is described more fully in Section 6, as it is often used in its integration capacity as part of KnoME-pro.

5.3 Extending Necessary Properties

The previous section described facilities that KnoME-lite-extender provides for supporting the production of textual GRAIL source which could, in theory, represent any kinds of additions to the model (e.g. additional elementary concepts, additional sanctions, etc.). However, another kind of extension activity is delineated rather better: that of annotating various sections of the model with necessary statements. This can be done at any stage in the development of the model; within GALEN-IN-USE. For example, parts of the anatomy model can be introduced as elementary concepts, which in itself is sufficient for certain activities to continue. To fully represent what we wish to, requires a number of necessary statements to be added to the model.

To support this specific activity, a tool is provided to annotate necessary statements. It is opened from a Knowledge Browser Special menu, and is designed to enable easy annotation of a section of model. It is illustrated below. There is a context-button menu option to adjust the set of attributes; once set, values can be added: either using the Add… button, or use drag-and-drop from another Knowledge Browser (the Knowledge Browser from which the tool was opened is not shown for clarity). To move on to the next in the (in this case) set of nerves, use the Next button (for which a hot-key is also provided — see the menu-bar option Topic→Next).

As the tool is used, necessary statements are added to the model (to check for potential redundancies, etc.). After that, the equivalent GRAIL source for the statements added can be created and saved to file using the Export option. The resulting file is then passed on to the central integration site, where KnoME-pro is used to integrate it with the rest of the developing model.


annotating: AnteriorSuperiorAlveolarNerve
File
Topic
Options
Attributes:
serves
sensibly
topicNec
nec
Values:
Canine
Incisor
Nose
Add...
Next
Previous
Export...
Close
Hierarchy
Information
Highlighting
Special
Help
Hierarchy
'FacialFeature'...
'TubularBodyPart'...
'TubularRespiratoryTractComponent'...
(PhysicalStructure which < isSurfaceDivisionOf LaminarPhysicalStru
Nose
Levels:
1
relevant attributes
values applied (using the selected attribute)
navigate through the list of topics
supports drag and drop from a Knowledge Browser
controls the directionality of statements, and whether sensible statements should also be applied

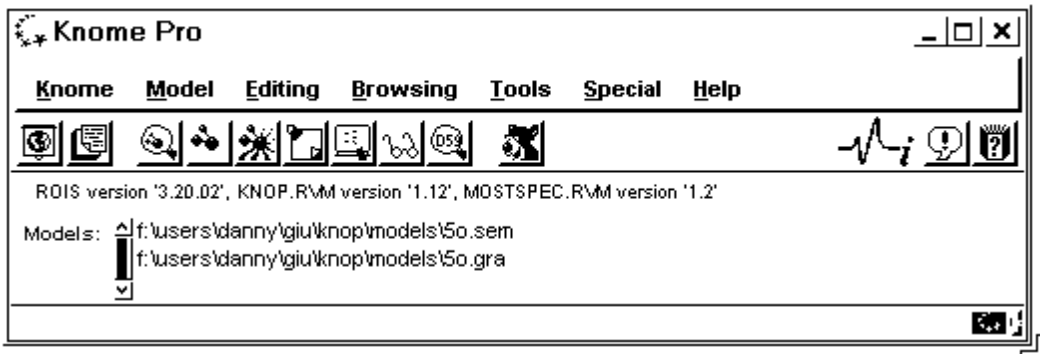
6 KnoME-pro

This section introduces KnoME-pro, which is a suite of tools that pulls together:

- all the tools from KnoME-lite-browser (Section 4), with additional properties and features;
- all the tools from KnoME-lite-extender (Section 4.4), with additional properties and features;
- Tigger (described in 'Tigger: Motivation, Overview and Interface'), to manage the use of the GALEN-IN-USE intermediate representation;
- specific GRAIL harmonisation and integration tools (introduced here in Section 6.2).

6.1 The KnoME-pro Main Window

As introduced in Section 2.2, all the tools and functions provided by KnoME-pro are accessed via the KnoME-pro Main Window, illustrated below:



The menu bar is organised as follows:

- KnoME provides facilities for closing the software (either to switch to KnoME-lite-browser or KnoME-lite-extender if available — using KnoME→Close — or to close the software altogether, using KnoME→Exit…);
- Model is only present if the KnoME is conneceted to a PUPPI; it provides facilities to open and close various kinds of model (as described in Section 3.1);
- Editing opens the tools for creating, extending and organising GRAIL models;
- Browsing opens tools for browsing and exploring the model;
- Tools provides facilities for integrating and harmonising models, for creating and viewing perspectives on the model, and for opening Tigger;
- Special provides facilities for describing which language is to be used as the default in all the tools, for opening the system transcript if required, and for applying any patches that may be supplied.

The rest of this section describes some of these tools in more detail.

6.2 Harmonisation and Integration

6.2.1 Tigger

Tigger is the part of KnoME-pro that is used to maintain and support the use of an intermediate representation within the GALEN-IN-USE project. This is designed to make it easier for clinicians to represent terms and rubrics from existing classification schemes in a structured way, such that it can be transformed automatically into GRAIL, in the specific form and style required by the particular implementation of the Common Reference Model. Clinicians doing this task no longer require a detailed and specialised knowledge of GRAIL and the Common Reference Model, and an extra layer of abstraction is introduced between existing classification schemes, and GRAIL. This provides extra resilience to change within the model itself.

Tigger is described fully in a separate document: 'Tigger: Motivation, Overview and Interface'.

6.2.2 Model Editor

As described earlier, GRAIL source is often written in a textual form, which can be presented to a compiler for incorporation into the model. With small models, developed in isolation, it is possible to manage these sources manually. However, as the scale of the model grows, and contributions to that model are developed concurrently, manual maintenance of this resource becomes implausible, especially as the structure may need to be maintained and re-arranged, and integration with other extensions is required.

The Model Editor organises individual parts or ‘units’ of the model; it provides facilities to manage and maintain the structure of the model, to import extensions produced elsewhere (for example by users of KnoME-lite-extender), and to export views of the model for use by different applications.

6.2.2.1 Overview

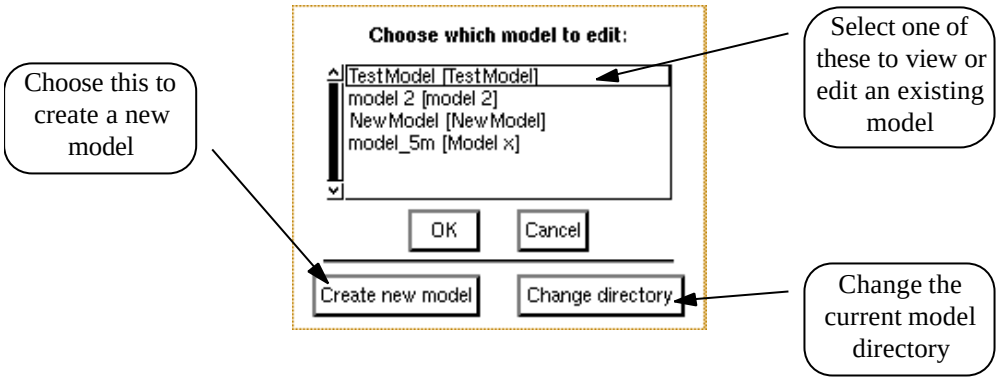
The model editor consists of two tools: the Model Structure Editor and the Model Unit Editor. The Model Structure Editor allows you to rearrange the model and manipulate several sections of the model at once. The Model Unit Editor allows you to edit the GRAIL source in the individual units.

6.2.2.2 Model Structure Editor

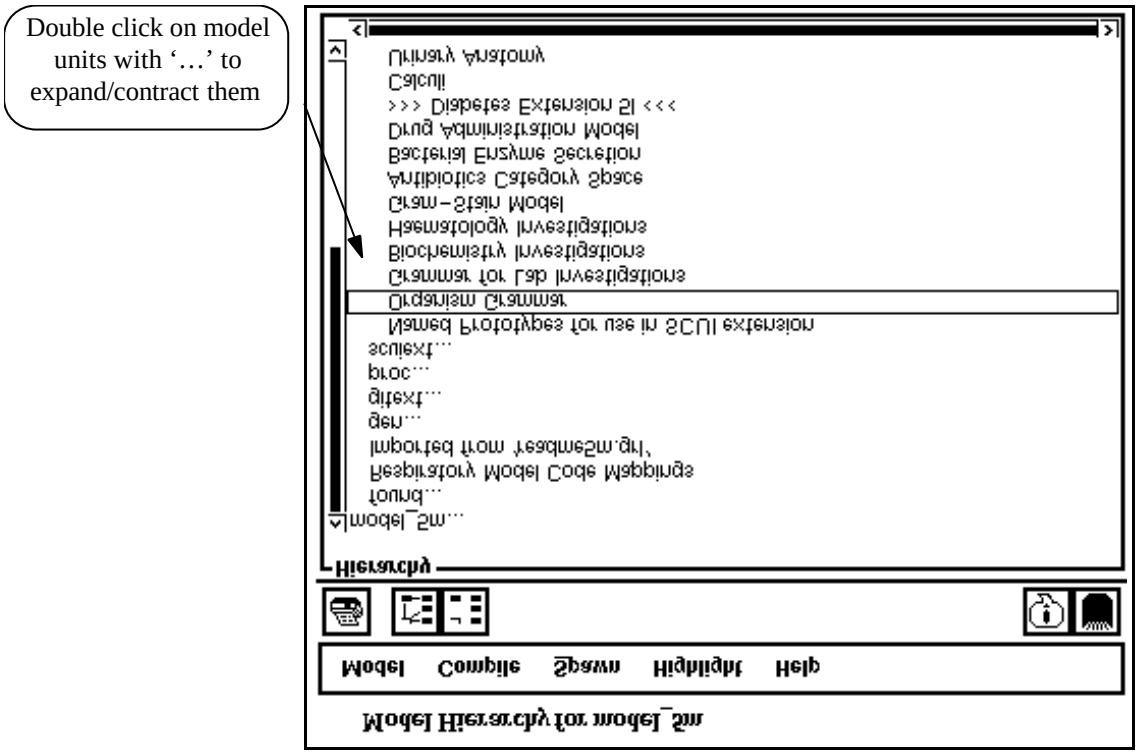
The Model Structure Editor can be started up from the KnoME Main Window by clicking the icon shown to the right, or by selecting Tools→GRAIL Model Editor from the menus.



Initially, you will be asked to select an existing model or to create a new one:



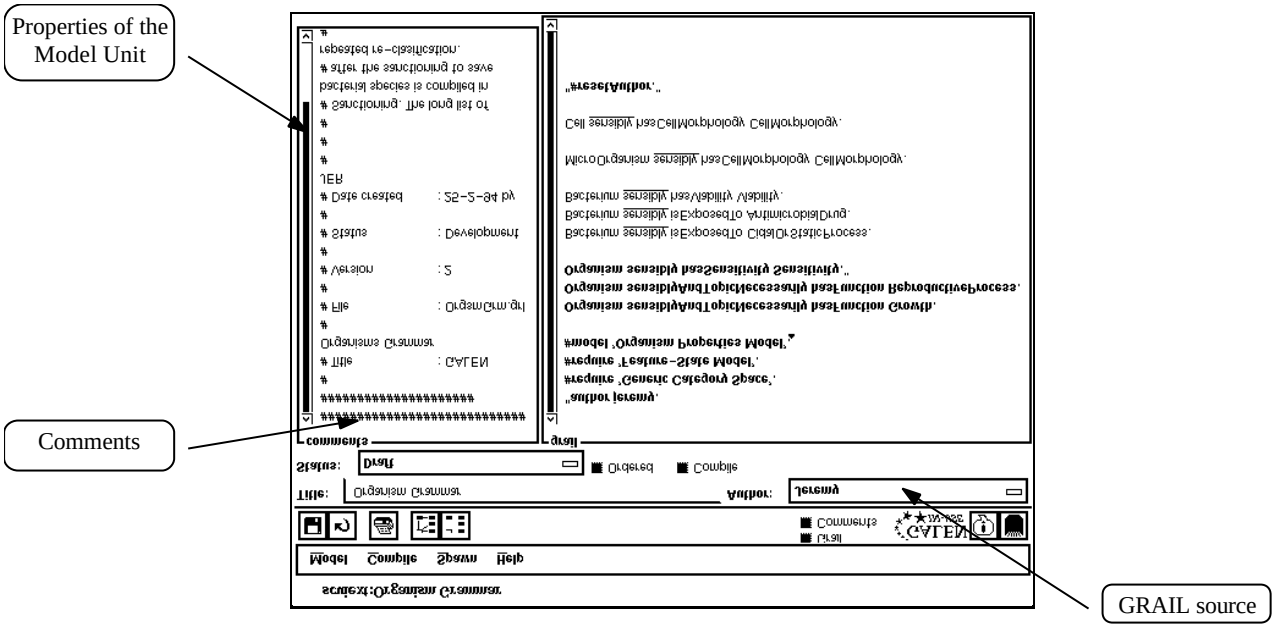
Once you have selected or created your model, you will then see the Model Structure Editor:



This tool provides a hierarchical overview of the structure of the model, and provides options to:

- import extensions or other existing models;
- export a description of the model in a form suitable for use by other applications;
- re-organise the model;
- open a editing tool to update a section of the model (see next section);
- view the log of changes that have been made.

6.2.2.3 Model Unit Editor



The Model Unit Editor is split into three sections. These are described in detail below.

In addition to these, on the button bar there are two checkbox which allow you to specify whether the GRAIL source and the comments are shown.

Any changes you make to the model unit will not take effect until you save them. This can be done by either selecting the disk icon from the button bar or Model→Save from the menu.

The compilation and check syntax buttons are available (as in the Model Structure Editor). These will compile / check the syntax of the model up to the model you are currently editing.

Properties

The properties section includes the management information for this model unit. The order checkbox specifies whether units below this one in the hierarchy are ordered. The compile checkbox allows you to choose whether to compile this units (and those below it), this allows you to test the rest of the model without compiling those sections that you still need to fix.

Comments

This is just a simple text window for you to attach comments to the model unit. You can still include comments within the source by surrounding your comments with double quotes.

Source

The GRAIL source area is similar to the GRAIL workspace described above.

6.2.3 Source Harmonisation

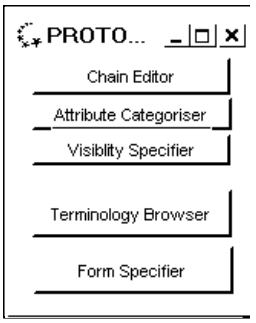
The Model Editor provides facilities to organise and maintain the model, and to integrate extensions to it. However, an another important part of this process is that of harmonising (or the checking of harmonisation) the style of modelling used in an extension, with that of the model into which it is being integrated. The menu option Tools→Source Harmoniser opens a tool which splits an incoming extension into different parts (depending on the kind of statements that are contained within it), and checks particular elements of modelling style. The definition of such styles, and the extension of other aspects of this kind of activity, is expected to be developed further during 1998.

6.2.4 Perspective Editor

The GRAIL Common Reference Model is designed to be application-independent: to support different kinds of clinical applications, each of which may have their own requirements as to coverage, detail and complexity. This implies, however, that the common model provides support to the union of these requirements, such that for any particular individual application, there may be too much detail or complexity presented to it. To overcome this problem, there is the notion of *perspectives*. A perspective acts as a layer between the raw, underlying model and the application, providing an application-specific view, or perspective, on that model.

Some of this application-specific behaviour is provided by the application itself, but a great deal of it can be provided by the server itself:

- in the provision and support of a standard set of mechanisms to support perspectives;
- and by providing a standard set of perspectives to support particular kinds of applications



The menu option Tools→Perspective Editor opens a small sub-window from which different perspective editing tools may be opened, as shown to the right.

The Chain Editor provides ways of simplifying the often complex structures that may link two concepts using intermediate 'chains' of attribute←>concept←>attribute.

The Attribute Categoriser provides means to organise the hierarchy of attributes into groups which are useful points for indicating whether they should or should not, for example, be made visible to certain applications.

The Visibility Specifier is designed to provide more detailed filtering of what is, or is not, made visible to different kinds of applications, at the level of specific concepts that may form the focus of activity in different applications.

The Terminology Browser and Form Specifier are test applications, mimicking the requirements of an application that does not wish to see the full complexity of the underlying GRAIL model.

Full details of the use and implementation of the perspective mechanisms, and the tools to support them, are provided separately[22].

6.2.5 Internal Validation Tools

The use of GRAIL in the GALEN programme has a significant impact on the kinds of internal validation checks that need to be carried out, in comparison with their use in the more manual development of existing classification schemes.

For example, the subsumption (kind-of) relationships in the Common Reference Model are all maintained automatically by the underlying implementation of GRAIL inside the GALEN Terminology Server. This doesn't mean that an incorrect classification can never take place, but rather that the way of correcting such a problem requires changing other information, upon which the GRAIL classifier had made the link. Similarly, a missing subsumption link is not rectified by manually adding it, but rather by adding other information — describing concepts within the model further — that will cause the GRAIL classifier to add the link.

Reliance on the GRAIL classifier implies, however, that its implementation must be trusted. This is supported by a regression test suite, and also by a tool that automatically compares the results of two separate implementations of GRAIL. The menu option Tools→Server Tester opens an interface to connect to two servers, each with its own implementation, and checks and reports any discrepancies between them. The two servers might be different versions of the

[22] Contact galen-help@galen-organisation.com for up-to-date documentation details

same implementation, or they be different implementations (i.e. one may be the TeS, the other the PUPPI).

Much of the classification with the model is controlled by necessary statements: often elementary concepts are added with no (or very little organisation), and then they are annotated with necessary statements to achieve a full classification[23]. To check that such annotations have been completed across a whole section of the model (e.g. to answer questions such as *have all the muscles, nerves and bones been annotated to describe how they inter-relate?*) KnoME-pro provides a Subsumption Comparator. The menu option Tools→Subsumption Comparator opens a tool focused on a particular concept (all descendants of that concept are checked), which allows the specification of a set of subsumption links that you may wish to check, and then presents the results. This functionality is also available from the Knowledge Browser menu Highlighting→Subsumption check, which presents its results by highlighting them within the hierarchy display.

[23] See GALEN-IN-USE Deliverables 5.1 & 5.2 for full details of the modelling style used in the Common Reference Model

Appendix Two:
Tigger: Overview, Motivation and Interface

Tigger:
Motivation, Functionality & Interface

Danny Solomon (danny@cs.man.ac.uk)

January 1998

Summary

During 1996 & 1997, the GALEN-IN-USE project has developed an intermediate format for the representation of surgical procedures: a structured representation of plain textual rubrics or terms, but less complex and detailed than GRAIL. This is intended to make it possible to involve more clinicians in the project, without requiring each of them to have a deep knowledge of GRAIL and the GALEN Common Reference Model (CRM).

Tigger is a device for bridging the gap between that intermediate format, and GRAIL. This document describes:

- the motivation behind the intermediate format;
- the functionality that is needed in Tigger;
- an implementation of that functionality;
- the user interface that has been developed to support that functionality.

This document is intended to act as a User's Guide for the Tigger environment, and also as an informal specification of the dissection translation process.

The change history of this document is now included in the introduction

Contents

1 INTRODUCTION 84

1.1 GALEN, GRAIL AND THE COMMON REFERENCE MODEL 84
1.1.1 GRAIL 84
1.2 AN INTERMEDIATE FORMAT FOR GALEN-IN-USE 85
1.2.1 Dissections 85
1.3 TIGGER 86
1.3.1 Introduction 86
1.3.2 Kinds of Tigger functionality 87
1.3.3 Relationship to other work in GALEN-IN-USE 90
1.3.4 Background 92
1.4 AUDIENCE & PURPOSE: HOW TO READ THIS DOCUMENT 93
1.4.1 Colophon 93
1.5 FURTHER INFORMATION 94
1.6 CHANGE HISTORY 94

2 DISSECTION AUTHORING, IMPORT, VALIDATION & MANAGEMENT 95

2.1 DISSECTION IMPORT 95
2.1.1 Error handling 95
2.2 AUTHORING / AMENDING DISSECTIONS 95
2.3 VIEWING DISSECTIONS 96
2.3.1 Individual Dissections 96
2.3.2 Sets of dissections 96
2.4 DISSECTION VALIDATION 96
2.4.1 Manual annotation 97
2.5 DISSECTION MANAGEMENT 97
2.6 USER INTERFACE: THE TIGGER MAIN WINDOW 98
2.6.1 Menu bar 98
2.6.2 Dissection List 100
2.6.3 Dissection Editor 100
2.6.4 Tigger Preferences Tool 102
2.6.5 Importing Dissections from file 103
2.6.6 Tigger Global Export Tool 104
2.6.7 Tigger Dissection Management 104

3 DESCRIPTORS: THEIR MANAGEMENT AND THEIR MAPPING TO GRAIL CONCEPTS 109

3.1 INTRODUCTION 109
3.1.1 Descriptor Categories 109

3.1.2 Descriptor Interchange 109
3.2 DESCRIPTOR MAPPING 110
3.2.1 Contexts 110
3.2.2 Descriptor Mapping Rules 110
3.3 USER INTERFACE: THE DESCRIPTOR MANAGER 110
3.3.1 Descriptor List 111
3.3.2 Descriptor mapping 113
3.3.3 Descriptor usage 114
3.3.4 Lexical Variants 115
3.3.5 Specifying Descriptor Mapping Rules 115
3.3.6 Checking descriptor category discrepancies 116

4 LINKS: THEIR MANAGEMENT AND THEIR MAPPING TO GRAIL ATTRIBUTES 117

4.1 INTRODUCTION 117
4.2 LINK MAPPINGS 117
4.2.1 Expansion into GRAIL 'chains' 117
4.2.2 Multiple mappings 118
4.2.3 Contexts 118
4.3 LINK MANAGEMENT 118
4.4 USER INTERFACE: THE LINK MANAGER 118
4.4.1 Help information 120
4.4.2 Mapping area 120
4.4.3 Link usage 121

5 MANAGING COLLABORATIVE WORK 122

5.1 INTRODUCTION 122
5.2 TIGGER "PACKAGES" 122
5.3 GATHERING WORK INTO A PACKAGE 122
5.4 EXPORTING AND TRANSPORTING A PACKAGE 123
5.5 IMPORTING PACKAGES 123

6 DISSECTION TRANSLATION 125

6.1 INTRODUCTION 125
6.2 OVERALL TRANSLATION MECHANISM 125
6.3 SPECIAL LINKS 126
6.3.1 OR_MAIN 126
6.3.2 OR_WITH 126
6.4 BOOLEAN OPERATORS 127
6.4.1 Conjunction 127
6.4.2 Disjunction 127
6.5 TRANSLATING A SINGLE DISSECTION TO A SINGLE GRAIL CONCEPT 127
6.5.1 Start point 128

6.5.2 Translating the dissection body 129
6.5.3 Headers transformed into extrinsic annotations 130
6.6 OUTSTANDING ISSUES 130
6.6.1 Non-analytic links 130
6.6.2 OTHER 130
6.6.3 Suppressed links 131
6.7 USER INTERFACE 131
6.7.1 Interactive dissection translation 131
6.7.2 Dissection compilation 133
6.8 ABSTRACT GRAIL CONCEPTS 133

7 CRM EXTENSION & VALIDATION 135

7.1 INTRODUCTION 135
7.1.1 Specification of requirements for extending the CRM 135
7.1.2 Validation and extension of sanctioning knowledge within the CRM 135
7.2 SANCTION MANAGER 136
7.2.1 Organising and filtering the set of connections 137
7.2.2 Marking connections 138
7.3 SANCTION PROMOTER 140

8 DISCUSSION 143

8.1 LINK MAPPINGS AS A FORMALIZATION OF "MODELLING STYLE"? 143
8.1.1 Translation of automatically-generated dissections 143
8.1.2 Why not just use sanctioning information? 144
8.2 HOW MUCH SHOULD TIGGER DO? 144
8.3 MULTILINGUAL ASPECTS 144
8.4 OTHER INTERMEDIATE REPRESENTATIONS 144
8.5 CONCLUSION 145

APPENDIX ONE: DISSECTION INTERCHANGE FORMAT 146

APPENDIX TWO: GRAIL SOURCE TO SUPPORT EXTRINSIC INFORMATION 148

APPENDIX THREE: CONNECTION INTERCHANGE FORMAT 149

APPENDIX FOUR: REPORT DESCRIBING EXPERIMENTS WITH THE NHS CCC 150

1 INTRODUCTION 152

1.1 BACKGROUND 152

1.2 SUMMARY 152
1.3 OVERVIEW 153

2 READ CODES & TEMPLATES ® DISSECTIONS 155

2.1 READ TEMPLATES 155
2.2 GRAIL INTERMEDIATE REPRESENTATION DISSECTIONS 155
2.3 CONVERTING READ TEMPLATES TO DISSECTIONS 156
2.3.1 Efficiency / scaleability 156

3 DISSECTIONS ® GRAIL 157

3.1 EFFICIENCY / SCALEABILITY 157

4 COMPARISON OF READ & GRAIL HIERARCHIES 158

4.1 GRAPHICAL HIERARCHY COMPARISON 158
4.2 NON-GRAPHICAL HIERARCHY COMPARISON 158
4.3 EFFICIENCY / SCALEABILITY 160

5 RESULTS 161

5.1 BENEFIT TO GALEN 161
5.2 BENEFIT TO CCC 161
5.3 FUTURE WORK 161

List of Figures

Figure 1 : An example dissection 86
Figure 2 : GRAIL translation of the dissection shown in Figure 1 86
Figure 3 : Different tasks supported by Tigger 90
Figure 4 : The relationship between Tigger and the ClaM Workbench 91
Figure 5 : The relationship between Tigger and other GALEN-IN-USE activities 92
Figure 6 : The Tigger Main Window opened from the GRAIL KnoME 98
Figure 7 : Tigger Dissection Editor 101
Figure 8 : The Tigger Preferences Tool 102
Figure 9 : The Tigger Dissection Import Tool 103
Figure 10 : The Tigger Global Export tool 104
Figure 11 : The Tigger Transformation Tool 105
Figure 12 : Adding and editing a new transformation 106
Figure 13 : The Tigger Common Rubric Browser 108
Figure 14 : The Tigger Descriptor Manager 111
Figure 15 : Routes towards mapping of descriptors 114
Figure 16 : The Tigger Synonym Tool 115
Figure 17 : The Tigger Descriptor Mapping Rules Browser 116
Figure 18 : The Tigger Link Manager 119
Figure 19 : GRAIL mapping area showing multiple attributes 120
Figure 20 : GRAIL mapping area showing attribute chains 121
Figure 21 : Tigger Package Viewer showing your changes 123
Figure 22 : Examining elements of a package 124
Figure 23 : Specifying the GRAIL start point 128
Figure 24 : Mapping MAIN, treated as a link 128
Figure 26 : GRAIL Output window showing GRAIL translation 132
Figure 27 : CRM classification of dissections displayed in a Concept Browser 133
Figure 28 : The Tigger Sanction Manager 136
Figure 29 : Setting the topic filter in the Sanction Manager 138
Figure 30 : Marking statuses within the Sanction Manager 139
Figure 31 : The Sanction Promoter 140
Figure 32 : Finding common parents using the Sanction Promoter 141
Figure 33 : Exploring children of concepts from the Sanction Promoter 142

1 Introduction

1.1 GALEN, GRAIL and the Common Reference Model

The GALEN Project (1992-1995) has developed a language for representing clinical information in a way that makes it accessible and useable for both people and computer systems. Traditional methods for representing such information involve manually enumerating large collections of words and phrases ("terms" or "rubrics"), and manually asserting the relationships between these terms. This process is widely used, but has resulted in a large, disparate collection of "coding systems" or "classification systems" that are:

- incompatible: there are many coding & classification schemes in existence, which have typically been designed to support a particular use, but which cannot be easily extended or re-used for different purposes. Translating information between these formats is sometimes possible, but this may degrade the information content in transfer; even if it is possible, maintaining mapping tables between each pair of schemes is expensive and error-prone;
- difficult to use for data-entry: they are typically too large, in that it has proved impossible to use them to support data entry systems for routine use by clinicians; such data-entry systems typically involve picking from long, incoherent "picking-lists" of terms. Simultaneously, they are too small, in that they do not offer the requisite level of clinical detail that is required;
- inconsistent: coding & classification schemes are typically ordered into hierarchies, but the meaning of the relationships between different levels of the hierarchy is unclear, and often varies, even within a single scheme;
- incomplete: what subsumption relationships do exist are incomplete, both because they are maintained manually, and because the technical implementation of many schemes only allows a single parent for each code.

Despite all these shortcomings, these kinds of systems are still widely used (and are often mandatory for statistical returns to central governments). The production, maintenance and translation of these schemes demands increasing amounts of effort {ref}. Providing a more effective and useable representation of clinical information — letting the computer "know" more about the information it stores — is widely seen as key to improved health-care systems {ref}.

1.1.1 GRAIL

The GALEN Project has developed a language for representing clinical information in a structured form (or "model"), designed an ontology using that language, developed delivery mechanisms that enable clinical applications to make use of such models, and demonstrated ways of linking such models to existing coding and classification schemes so as to act as an interlingua.

The language developed in GALEN is known as GRAIL (the GALEN Representation and Integration Language). It has similarities to a range of description logics such as KL-ONE,

Loom or Classic {refs}, but it has been specifically designed and developed to support the requirements of a clinical terminology. Some of the design criteria involve formal decisions such as the balance between expressiveness and tractability; others involve usability for clinicians: for example the development of a syntax and a development environment suitable for use by clinicians to build a large and expressive model of medicine. Using such an environment, a small team of clinicians, working closely with the developers and implementors of GRAIL, have succeeded in producing a high-level model (or “ontology”) of medicine, along with detailed models in particular sub-domains to support particular clinical applications. Together, this model is known as the GALEN Common Reference Model (CRM).

However, experience has shown that GRAIL, its syntax and its development environment together form a kind of "assembly language" for producing a structured clinical model: it is hard to learn, and it includes many complexities and quirks. Some of those complexities are due to the need to represent the domain in a very formal form for the computer; some of the quirks are artefacts of the particular facilities that have been chosen (or deliberately left out) of the GRAIL language.

The GALEN-IN-USE Project aims to harness the power of a formal GRAIL model (the CRM) for the production and maintenance of existing kinds of coding and classification schemes. By mapping elements of such schemes to the CRM, the automatic classification mechanism can re-organise such schemes into multiple hierarchies and extract specific, coherent, sub-classifications; the language generation facilities can ameliorate the translation of such schemes into different languages (and can standardise the natural language used); and the code conversion facilities can be used to automatically map between different schemes.

The prerequisite to the demonstration of these facilities is the representation of large numbers of elements of existing (or new) coding and classification schemes in the language of the Common Reference Model.

1.2 An Intermediate Format for GALEN-IN-USE

It is possible to manually represent elements of classification schemes, and to map them to the CRM, using GRAIL as the “assembly language”. However, this is a time-consuming task, and it demands clinical expertise, as well as detailed knowledge of GRAIL as a modelling language and the particular form of the CRM. Taking into consideration that:

i. GALEN-IN-USE is intended to massively expand the scope and scale of such modelling;

ii. whilst the full power of GRAIL is needed when considering the overall scale, much of the individual mapping tasks are much simpler, and follow a regular pattern,

one approach to expanding the scale and coverage of the CRM is to introduce an intermediate form in which clinicians could more easily represent much of the structure of terms or rubrics from coding and classification schemes. Clinicians are asked to produce “dissections”: structured descriptions, using a very simple syntax, of terms and rubrics. These dissections are then automatically expanded into the more complex form of the CRM.

This has the advantage that, rather than having to introduce and train many clinicians across Europe into the vagaries and complexities of GRAIL and the CRM, much of which can be hidden for a large proportion of their work, they can use a higher-level abstraction which is both easier to explain, and easier to build tools to support. The disadvantage is that in order to provide more simplicity to the end-users, more complexity has to be developed, and hidden in the underlying infrastructure. That level of complexity is the subject of this document.

1.2.1 Dissections

An individual dissection is designed to represent the structure of a term or rubric from a coding and classification scheme. It is illustrated as an example here using its textual format; whilst this text can be written manually using a word-processor, dissections are normally produced

using a graphical tool specifically designed for the purpose known as SPET (Surgical Procedure Entry Tool) which is described as part of GALEN-IN_USE Deliverable 7.2.

A dissection has two parts: a *header* representing information such as the source of the term and the textual form of the term itself, and a structured *body* where the structure of the term is represented as a graph. Within the graph, nodes are 'descriptors' — representing clinical concepts, and the arcs are 'links' — representing semantic attributes or roles. As a deliberate simplification, the structure of the graph is represented using indentation within the body of the dissection. For example:

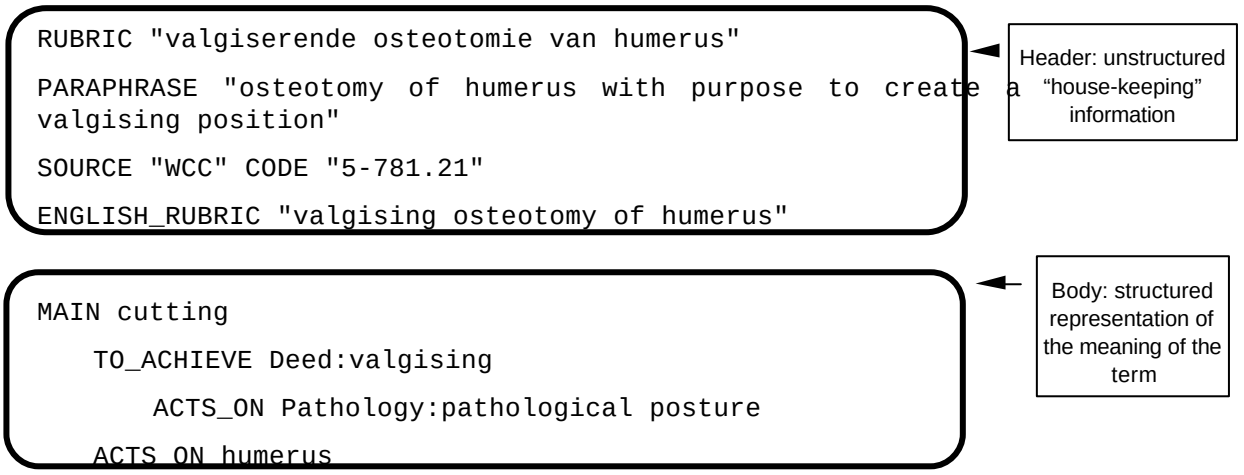


Figure 7 : An example dissection

A formal specification for the dissection syntax is provided as Appendix One on page 146, and an informal specification of the form of the links available is available in [VUM03].

1.3 Tigger

1.3.1 Introduction

Having designed and developed an interchange format (the "dissections") that would enable a large quantity of clinicians to represent large quantities of rubrics in a structured form, the task remains to develop a system to take those dissections, and, in the context of the CRM, produce GRAIL expansions of those dissections. For example, the GRAIL expansion of the example shown in Figure 7 above (which would, in the absence of the dissection mechanism, have had to be produced manually by the clinicians) is:

```
(SurgicalDeed which
        isMainlyCharacterisedBy (performance whichG
                isEnactmentOf ((Incising which playsClinicalRole SurgicalRole) whichG <
                        hasSpecificGoal (
                                (Valgising which playsClinicalRole SurgicalRole)
                                        whichG LocativeAttribute
PathologicalStandingPosture)
                        actsSpecificallyOn Humerus
        >)))
        hasProjection (('WCC' schemeVersion 'default') code '5-781.21' 'code');
        extrinsically hasDissectionRubric 'WCC 5-781.21 valgiserende osteotomie van
humerus'
```

Figure 8 : GRAIL translation of the dissection shown in Figure 7

The device to perform this task of translating dissections into GRAIL is known as TIGGER[24].

Tigger must perform this translation in the context of a particular GRAIL model. In the design of the CRM, there have been a number of modelling implementation choices: both in the choice of labels to give to concepts and attributes, and in the particular style in terms of the levels of complexity and detail that are represented, and how they are implemented. Therefore, we must provide a way for the implementors of the CRM to represent sufficient information within Tigger as to the CRM modelling style: enough so that the translation mechanism can produce valid GRAIL expressions in the context of the model.

This also demonstrates a subsidiary benefit of using the intermediate format: that of providing an extra level of indirection between the original source and the CRM. The material now being generated in bulk in GALEN-IN-USE offers a structured representation of terms and rubrics. The translation into GRAIL, being a separate process, can be changed or developed as necessary without having to re-do the basic structuring now being produced. The majority of the clinicians involved in the project are therefore isolated from any changes that may be made to the CRM. In turn, the implementors of the CRM are free to modify it, if required, knowing that to re-use the bulk material that is currently being generated, it will only be necessary to re-formulate the modelling style within Tigger.

Furthermore, this intermediate level of structure representation offers additional benefits: it offers a more easily attainable target for tasks such as analysing natural language, or of re-formulating existing structured or semi-structured forms. This aspect is described more fully in Section 8.

1.3.2 Kinds of Tigger functionality

This section outlines in more detail the different kinds of functionality that Tigger must support, and hence the different scenarios of use which it supports. They are split into four:

i. **Dissection import, validation & management**: loading dissections authored elsewhere, for validation, and so that further processing can be done; and integrating sets if dissections created separately into a coherent whole;

ii. **Dissection translation**: translating the dissections into GRAIL;

iii. **Authoring the specification of the style of the CRM**: providing the specification of the form of the GRAIL translations; this forms a essential input to the dissection translation process;

iv. **Using the dissections to extend and validate the CRM**: providing feedback to the CRM using the dissections as a corpus.

Dissection translation was initially the primary purpose for Tigger, but as that can now be done locally at dissection-author time using the Classification Workbench, the prime motivation for Tigger has developed into the central integration / management system.

1.3.2.1 Tigger as a dissection import, validation & management tool

Dissections are produced by clinicians at remote sites: either manually using word-processors, or (usually) by using the SPET data-entry tool which exports the dissections in the textual form. Tigger implements a parser to read these text files, and stores the dissections internally. Once they are loaded, they can be viewed and organised along various different criteria. Some of these are purely organisational (such as the source from where they have been loaded), whilst others are based on the dissections themselves: such as whether they satisfy various constraints that may be in force.

In addition, various transformations and reconciliations may be applied to ensure a common style.

---

[24] TIGGER stands for 'Template Interpreter and Grail Generator'. Dissections were initially, and for a short time, known as 'templates'.

Section 1.5 describes this aspect of Tigger in more detail, and illustrates the user interface that has been developed to support it.

webTigger

Tigger is a stand-alone program, normally used (in its prototype form) at the site responsible for the building and maintenance of the CRM in Manchester, UK. It is used to load dissections produced at other sites in Italy, France, the Netherlands and Sweden, which are transferred via ftp to Manchester. During the early stages of the process, the SPET data-entry tool was not available, and the dissection syntax was being revised and refined to meet the needs of the centres. It was important to demonstrate that the material being produced matched the dissection syntax, and that the Tigger parser could read that material. So as to allow rapid syntactic validation, a WWW-based interface to Tigger was produced so that Tigger as a dissection import and validation tool could be used remotely from multiple sites. This interface is known as webTigger, and is available at:

`http://mighp0.cs.man.ac.uk:8008/launch/GALENTemplateInterface`

1.3.2.2 Tigger as a dissection translation tool

Once Tigger has loaded a set of dissections, it can translate each into one or more GRAIL expressions. This process is described briefly here, and in more detail in section 5.

Basics of dissection translation

Dissections are a simplification of GRAIL expressions in several different ways:

- "descriptors" are a simplified way of referring to arbitrarily complex GRAIL concepts, using language that is recognisable to clinicians;
- a small number of "links" are used to connect descriptors together, rather than the more numerous GRAIL attributes: context is used to disambiguate in the translation process from link to GRAIL attribute;
- link-concept-link chains (such as the Feature/State mechanism which is the way modifiers are implemented in the CRM [GALEN-IN-USE Deliverable 5.1]) are hidden in a single link;
- syntax is simpler: bracketing is limited, to ensure that the majority of dissections are kept as simple as possible;
- more than one GRAIL concept can be implicitly represented in a single dissection, to mirror the usage within existing coding and classification schemes.

However, both dissections and GRAIL expressions represent structured graphs; the process of translation is to take the dissection body, expand the simplifications described above, and then re-attach information provided in the dissection header as extrinsic statements {ref} and as mappings to the original coding scheme.

Reviewing classifications

Once GRAIL translations are produced, they can be compiled: Tigger is a client application that communicates with a GALEN Terminology Server {ref}. The resulting classification can be viewed using the existing environment for viewing the model: the GRAIL KnoME (Knowledge Modelling Environment) {ref}. This is suitable for an overview of the resulting classification; however, the KnoME is intended to build and reconcile the CRM itself; different facilities are required for classification centres —these are to be found in the Classification Workbench (ClaW) which is a separate development taking place within GALEN-IN-USE.

1.3.2.3 Tigger as a development environment

There is much information (outlined in the previous section) that is required by Tigger to expand the dissections into GRAIL expressions. The specification of this information demands both clinical knowledge, and knowledge of the implementation of the CRM. Thus it is

necessary to provide an environment where this information can be entered and managed — it cannot be simply hard-wired into the implementation.

Descriptor & link mappings

Tigger provides facilities to:

- enter, amend and interchange mappings between descriptors and their corresponding GRAIL concepts; this is described in detail in section 3;
- enter, amend and interchange mappings between links and their corresponding GRAIL attributes, together with the means to specify how context is taken into account when translation takes place; these are described in detail in section 3.3.2.1.

Whilst preferred descriptors and links are specified {ref} and available via the SPET, there will always be the requirement for sites, while producing dissections, to introduce new descriptors and links. To map these requires the Tigger authoring environment to provide facilities for viewing the contexts in which these new links and descriptors are used.

Managing collaborative work using Tigger

Tigger is designed to be used by a small group of clinicians: to specify to Tigger how to translate dissections, and perhaps to modify some dissections. This activity produces a central resource which is then used to support the activities of a wider group of clinicians.

However, it has proved to be important to manage collaboration between the small group of clinicians using Tigger directly: work by different clinicians mapping descriptors and links, and amending dissections needs to be integrated and harmonised. Tigger provides tools to assist this process; they are described in section 5.

1.3.2.4 Tigger used to extend and validate the CRM

The prime focus of the production of dissections is to represent elements of existing coding and classification schemes in the CRM. However, this activity also forms a way of directing the development of the CRM, and of validating it:

- directing development of the CRM: descriptors might be used for which the corresponding clinical concepts have not been modelled, or links might be used which indicate a semantic relationship that has not been modelled;
- validating the CRM: much of the activity in translating dissections into GRAIL involves searching for and using sanctions in the CRM; the use of particular constructions in dissections may indicate where sanctions are missing.

Tigger provides facilities to support both these kinds of activity: the first to support the translation of new dissections, the second as a separate off-line activity. These facilities are described in detail in section 7.

A graphical view of the different kinds of activity that Tigger supports is shown in Figure 9.

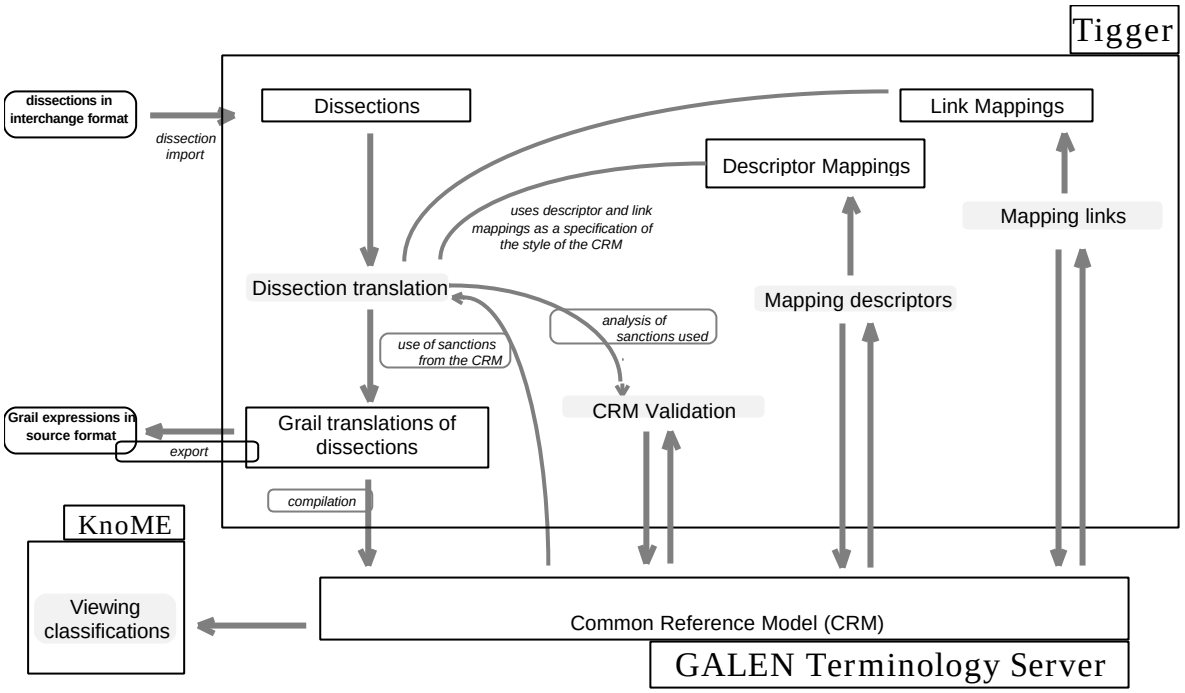


Figure 9 : Different tasks supported by Tigger

1.3.3 Relationship to other work in GALEN-IN-USE

The production of dissections is designed to support the use of a Classification Workbench (ClaW); the overall flow of which is shown in Figure 10.

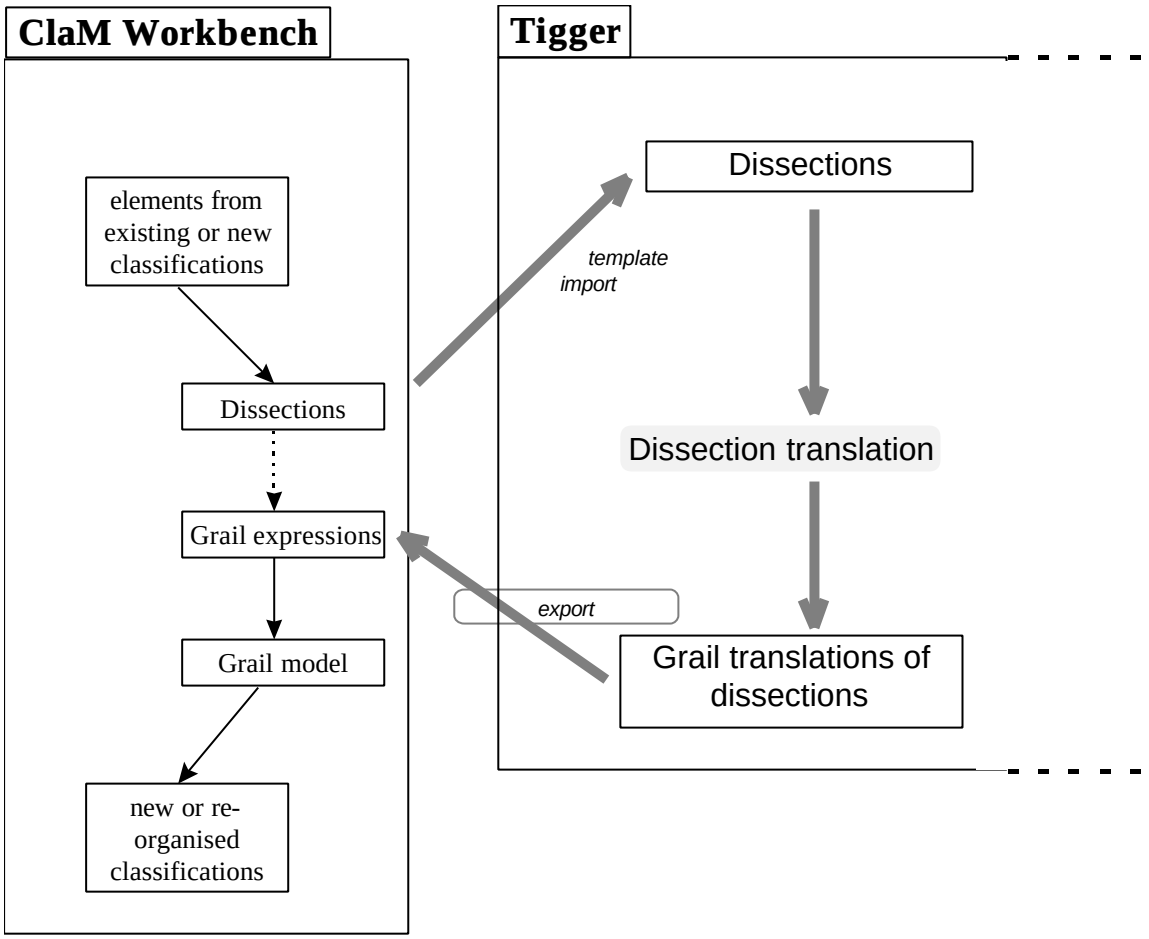


Figure 10 : The relationship between Tigger and the ClaM Workbench

The ClaW has for some time supported most of the functionality required for a single environment to be able to be delivered to classification centres. In the development phase, however, the process of translating dissections to GRAIL expressions was only supported by Tigger, and hence there had to be transfer between these two environments to achieve the overall flow within the ClaW. Currently, the ClaW also implements the dissection translation functionality as indicated by the dashed line in Figure 10. Now that functionality is complete, Tigger will continue to be used for dissection management, authoring descriptor and link mappings, and for validating and extending the CRM. An overview of the current position in which Tigger sits in relation to other work in the project is shown in Figure 11.

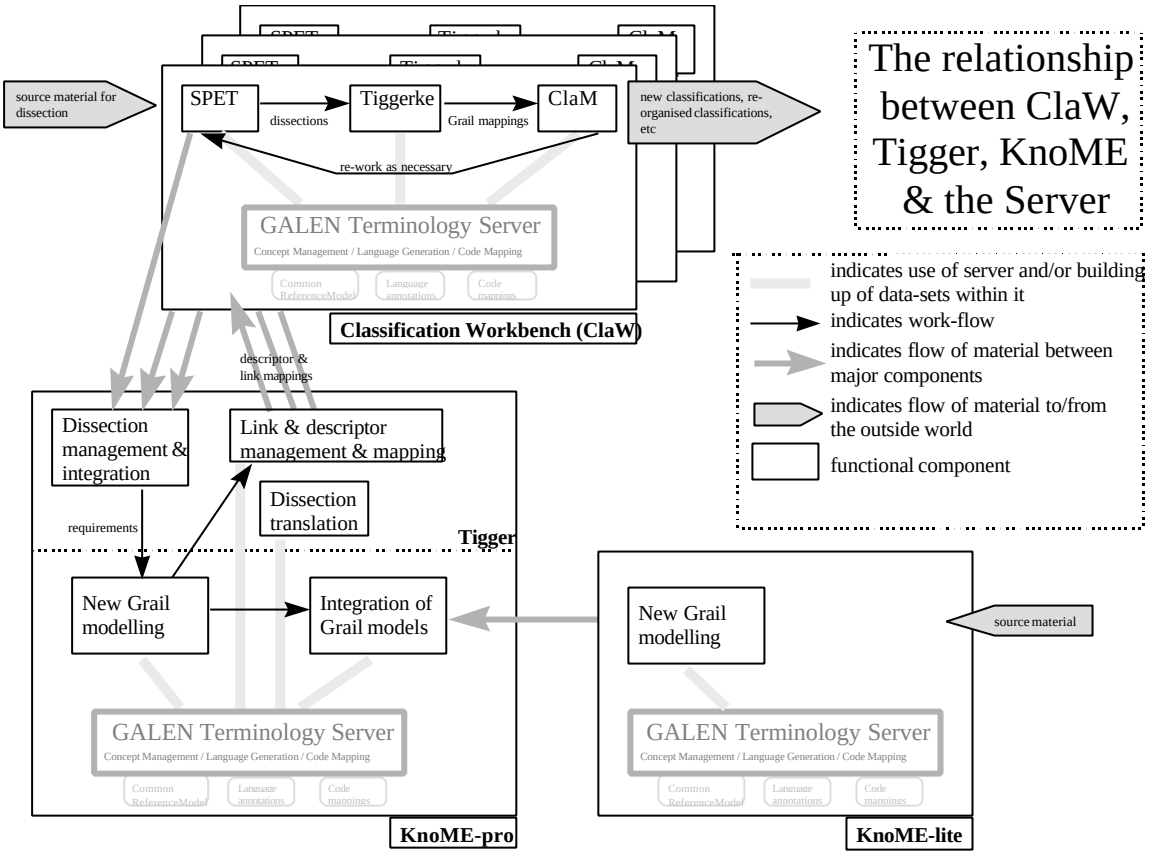


Figure 11 : The relationship between Tigger and other GALEN-IN-USE activities

This shows Tigger's position as part of KnoME-pro - the Grail harmonisation and integration environment.

To summarise, the two main tasks for Tigger at present are:

i) to maintain the set of descriptors and links (the 'controlled vocabulary' for writing dissections) and their mappings to GRAIL, and supplying that as a single resource[25] to users of the Classification Workbench;

ii) to maintain, integrate and harmonise the sets of dissections as they are produced concurrently by multiple users of the Classification Workbench.

1.3.4 Background

Structured representations of clinical concepts take many forms; a different form from that developed in GALEN-IN-USE was used before the project started by one of the project participants. The form used was a more constrained dissection consisting of a fixed-size two dimensional table. Early experiments to translate those tables into GRAIL were a useful introduction to the process of transforming an intermediate representation, and are described in {ref 2D-to-GRAIL doc}.

[25] this resource is colloquially known as the 'common.txt file'

1.4 Audience & purpose: how to read this document

This document is intended to serve three distinct purposes, for three different audiences:

- as a user guide to the Tigger environment for knowledge engineers harmonising and managing sets of dissections, and maintaining the set of descriptors and links;
- as an informal specification of the dissection-translation functionality within Tigger for implementors;
- as an introduction to the dissection-harmonisation mechanism for users of the Classification Workbench.

There is much overlap between these audiences: knowledge engineers mapping descriptors and links need to know the overall strategy of how dissections get translated into GRAIL, whilst implementors of translation functionality need to know how to interpret those mappings. At the same time, all this information may be of (background) interest to dissection authors. In order to try and make this document as readable as possible for all kinds of audience, the document has been laid out as follows:

- Section 2 describes the facilities within Tigger for reading, examining and managing dissections, and introduces the user interface facilities provided;
- Section 3 describes what descriptors are, and how they are mapped, in some detail, as well as describing the tools provided to support this mapping;
- Section 4 describes the mechanisms and tools for mapping links.

Together with the basics of dissection translation included in this introductory section, these three sections should be enough for knowledge engineers reviewing and translating dissections, and extending or amending link & descriptor mappings.

- Section 5 describes facilities provided by Tigger for assisting the integration and harmonisation of descriptor & link mappings, and of amended dissections.
- Section 6 provides a more detailed overview of the translation process.

This will be needed by those seeking to re-implement Tigger's translation functionality, but it may also be of interest to knowledge engineers using Tigger for more than bulk translations; for example: specifying any changes or extensions in the translation mechanism that may be required in the future, or debugging its output.

- Section 7 provides an initial view of additional Tigger functionality that is still being developed: that of using the dissections as a knowledge source to (semi-) automatically extend and/or validate the Common reference Model.
- Section 8 provides some discussion of Tigger as a whole, and describes some other tasks that may be possible given such an environment.

Several appendices are included, which provide detailed technical information on interchange formats, and which are only of interest to those seeking to re-implement parts of Tigger, or to re-use some of the information that it uses or generates.

1.4.1 Colophon

Elements of dissections (descriptors, links, or dissection analyses themselves) are presented `in this font`.

Elements of GRAIL are presented **in this font**.

Menu options are described **in this font**, with → indicating a sub-menu; for example: **File→Close**.

1.5 Further information

Other documents that cover (or overlap with) topics discussed here include:

- **GALEN-IN-USE Deliverable 9.2** (of which this document forms a part) describes, at a higher level, the task of managing the process of 'modelling' within GALEN-IN-USE. It includes details of the process of modelling in the GRAIL language directly, and the environments developed to support that;
- **GALEN-IN-USE Deliverable 7.2** describes the Classification Workbench in detail. This is the toolset which is in use by several classification centres across Europe. The Classification Workbench provides a mechanism for clinicians to enter dissections as a means of analysing existing coding or classification schemes (or creating new ones), for translating these into elements in the GRAIL model, and for using the resulting GRAIL classification to validate and re-organise these schemes;
- **GALEN-IN-USE Deliverables 5.1 & 5.2** describe the structure and form of the GRAIL Common Reference Model which supports the Classification Workbench (and other applications) and is the target model into which dissections are translated.

For more background information, see

http://www.galen-organisation.com/mig/GALEN-IN-USE

1.6 Change History

Date	Summary of changes
November 1996	first draft distributed to Nijmegen & Manchester
13 December 1996	added change management section, some parts of discussion
18 December 1996	revised Figure 3 & 4
23 January 1997	general tidying, added description of OTHER implementation, revised example translation based on extrinsics wrapper DissectionDetails, added more on generating abstractions
25 April 1997	simplified extrinsics link between GRAIL concept and dissection
10 June 1997	amended details of when to attach playsClinicalRole SurgicalRole to GRAIL mappings of descriptors (section 6.5.2)
January 1998	Major re-vamp to cover user-interface changes. All references to 'template' is now of 'dissection'. New sections on dissection transformation, and various other additional tools and features that have been developed over the past 6 months

2 Dissection authoring, import, validation & management

This section describes the facilities within Tigger for authoring, importing, viewing and managing sets of dissections. This is the first step, and allows processes of, for example, translating the dissections into GRAIL expressions, validating dissections, re-organising or transforming dissections.

An outline of the different kinds of facilities, and the motivation for them, is introduced first, before describing the user interface that has been developed to support them.

2.1 Dissection import

Dissections are read from text files containing one or more dissections. These files can be produced manually, but are more often the result of an export from the SPET. A grammar for the dissection syntax is presented as Appendix One.

An important requirement on Tigger is to manage the set of dissections as they are submitted by multiple authoring sites.

2.1.1 Error handling

The intention is that all textual input files for Tigger will be syntactically correct, as they will usually have been automatically produced by SPET. To support those cases where syntax is incorrect, due to manual creation of input files, to different interpretation or implementation of the interchange format, or because the dissections may have been generated (semi-)automatically from other structured sources, any syntactic errors found are presented for correction to the user.

2.2 Authoring/amending dissections

As well as reading dissections from external files, Tigger provides an interface (described in section 2.6.3) for adding new dissections or amending existing ones. This is not intended to be the primary way in which dissections are authored, but is provided for the following reasons:

- for development and experimentation with translation strategies, it is useful to add new test examples, or see the effect of changing a dissection;
- some problems with dissections only come to light when the translation process is attempted: being in a position to react to the feedback provided by the translation process speeds up the process of correcting the dissections (a discussion of the kinds of problems that do come to light is included in section 2.4).

2.3 Viewing dissections

2.3.1 Individual Dissections

Having been imported and parsed, it is often important to be able to see the original dissection in the form that it was written. Rather than storing the original textual form, Tigger re-generates a textual form of each dissection on demand; because the form of a dissection is constrained, the only difference between the re-generated textual form and the original is in the order of any optional header fields that may be present.

This mechanism is also used when exporting dissections from Tigger after any amendments that may have been made to the originals.

2.3.2 Sets of dissections

It is important, both for validation, and for seeing where and how particular links and descriptors are used, to be able to see filtered selections of dissections. The filtering criteria may be purely organisational (such as *show me all the dissections that were originally in the file wcc.txt*); they may depend on the content of the dissections (*e.g. show me all the dissections that use the descriptor 'excising'*), or they may depend on further interpretation of the dissection (e.g. *show me all the dissections that have been marked as broken in some way*).

2.4 Dissection Validation

It is relatively simple to produce dissections that are syntactically correct; however, there are a number of ways that dissections may be broken, and Tigger must check these states and be able to filter a set of dissections based on any one of these criteria. These criteria include:

- incorrect indentation level: the syntactic dissection interchange format included as Appendix One does not specify how levels of indentation should behave — in fact indentation can only increase by one level with each new link;
- broken constraints: there are some defined constraints [VUM03] which Tigger must check, for example that every main deed must have, as one of its describing links, the link ACTS_ON;
- broken sanctions: descriptors can only be combined if the categories to which they belong have been sanctioned to be connectable using a particular link;
- unknown links or descriptors: any dissection might introduce new links or descriptors; these have to be mapped (if possible) to GRAIL before any dissection using them can be translated;
- limitations on Boolean constructs: for example, there is a limit to the number of disjunctions between descriptors that can be used within a single dissection;
- use of special constructs: whilst there is informal understanding of the use and intention of the `NON_ANALYTICALLY` flag on links, there has been insufficient usage to be able to determine the formal rules for translating such links into GRAIL.
- errors in translation: it may be that, when the translation process is attempted, a valid translation for a link into a GRAIL attribute is not found. This may be an indication of a dissection that is broken, or some missing information in the link mappings; either way, until either the dissection or the mappings are amended, translation cannot be performed. Doing the translation is, individually, a fairly quick activity (up to 1 second per dissection, depending on the amount of information from the Terminology Server cached in Tigger); however, performing this check automatically when importing files (which may contain many hundreds of dissections) is liable to interrupt work flow.

- errors in compilation: even when a GRAIL expression can be produced from a dissection, further validation takes place when the GRAIL expression is presented to the Terminology Server for compilation: classification of the GRAIL concepts, and its annotation with extrinsic information from the dissection header. Errors which may arise in this phase include cardinality conflicts — the dissection translation process treats each link in isolation.

Some of these criteria are easy to check; whenever a dissection is imported, or amended within Tigger, it is reasonable that these criteria are re-checked. However, some of the later ways in which dissections may be broken are harder to find — they are more expensive in terms of processing time required and thus they are "harder" in the sense that they may compromise user-level responsiveness. It is possible, via the Tigger Preferences Tool, to arrange to what level these checks are made, at what time (either on import from file, or on any individual dissection change made within Tigger).

2.4.1 Manual annotation

As well as the formal criteria described so far, some other aspects of validation can only be performed manually. For example:

- mis-representation or incomplete representation of the rubric;
- incorrect style: there can be alternative ways of representing a rubric. For consistency, often a particular style is specified and agreed.

Tigger provides facilities to manually annotate any dissection with these or other notes as specified by the user.

2.5 Dissection management

During the first 18 months or so of the GALEN-IN-USE project, the main motivation for Tigger was to 'bridge the gap' of the Classification Workbench: to provide Grail translations of defined sets of dissections. As the project has continued, and the Classification Workbench has itself implemented such functionality, another requirement has become more important: that of *dissection management*. This involves, for example:

- ensuring a common style amongst the disparate dissection authors, by manually or automatically applying transformations;
- comparing dissections - between authors, and across the change history;
- exporting sets of mapped links and descriptors for use Classification Workbench users;
- providing an overview of the combined set of dissections, in terms of coverage & statistical measures such as numbers at a particular level of validation;
- maintaining a central resource of all received dissections.

2.6 User Interface: the Tigger Main Window

When started, Tigger provides a "Main Window" which displays a list of dissections that have been loaded. The Tigger Main Window also provides access to all the other tools which are described in subsequent chapters. The Tigger Main Window is started from the GRAIL KnoME Main Window[26]: either by using the menu option **Applications → Open Tigger**, or by using the short-cut button illustrated in Figure 12. Note that re-opening the Tigger Main Window will cause it to be de-iconified (if necessary) and brought to the front of the screen.

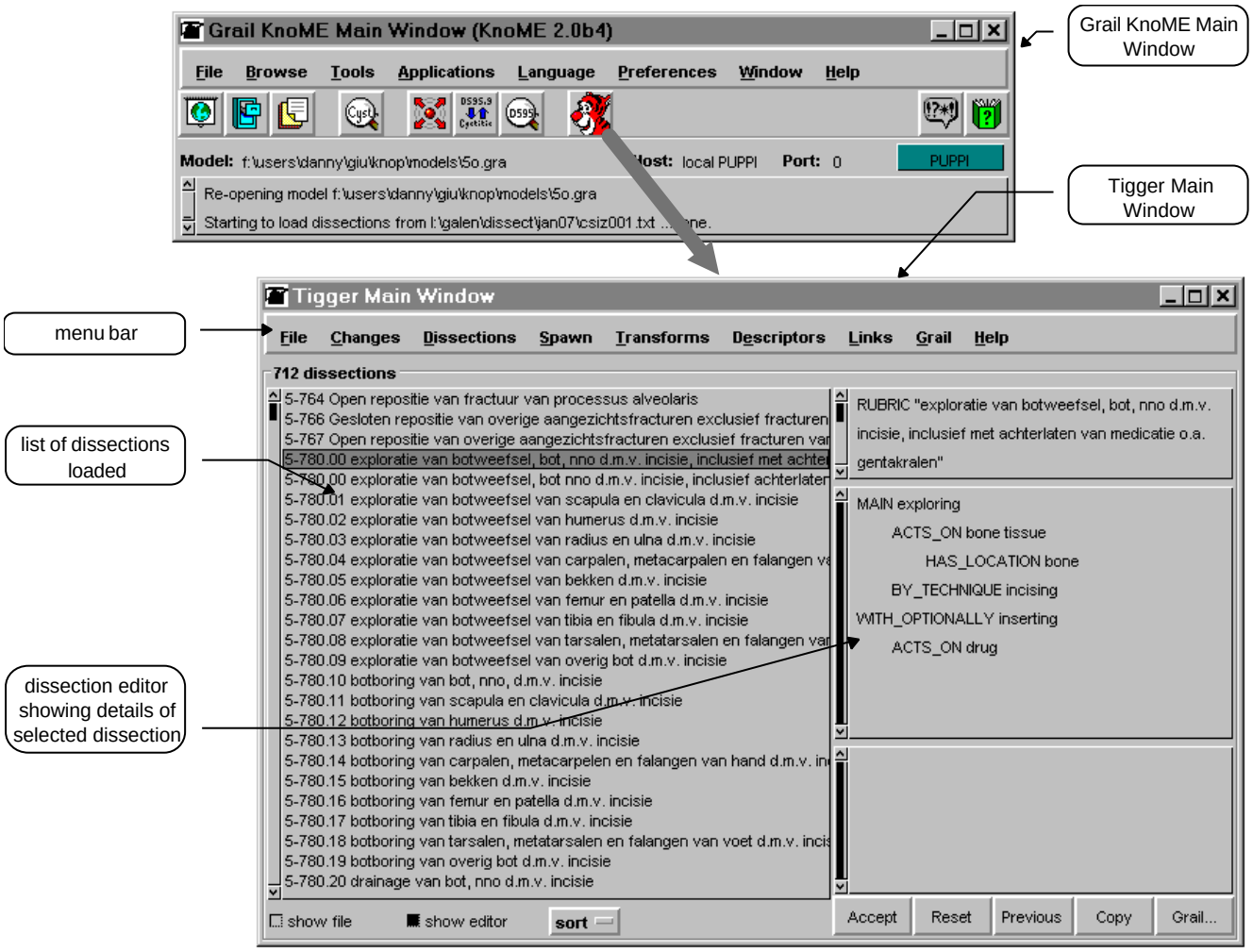


Figure 12 : The Tigger Main Window opened from the GRAIL KnoME

The Tigger Main Window has three parts: a menu bar for manipulation of the loaded dissections and for access to all the other Tigger functionality, a list of dissections that are currently loaded, and a "dissection editor" view of an individual dissection.

2.6.1 Menu bar

The **File** menu choices provide access to the Preferences Tool (see Section 2.6.4), for importing and exporting the global 'common.txt' file (see Section 2.6.6), for managing various statistical features (see Section 2.6.7) and for managing control to the GALEN Terminology Server (of which Tigger is a client application). Tigger is closed using **File→Exit**. The option

[26] for details of how to open the KnoME Main Window, see "The GRAIL KnoME: Overview and User Guide"

File→Close down clears all loaded dissections, descriptors and links before closing the tool, and is only infrequently used.

The **Changes** menu provides access to some facilities that Tigger provides for managing and integrating work; these are described in Section 5 (and include facilities for showing who is currently logged in, comparing changes to descriptors, links or dissections, etc).

The **Dissections** menu provide facilities to import dissections from file (see Section 2.6.5), amend or add individual dissections, manually mark dissections as broken in some user-defined way, and to change the way in which individual dissections are displayed in the list. The duplicate rubric tool (see Section 2.6.7.2) is opened from this menu. Finally, the sub-menu **Dissections→Check up to and including** provides a list of possible error types; selecting one will cause all the loaded dissections to be checked to that level. For example, to compile all dissections to Grail, choose **Dissections→Check up to and including→Compilation of Grail expressions**. By default, all error types are re-checked from the start; this is overridden by holding the shift-key down, in which case only those error-checking actions that have not previously been done are checked. The **Dissections→Next** and **Dissections→Previous** options (with associated hot-keys) allow you to easily browse through the list of dissections.

The **Spawn** menu provides facilities to pull out subsets of the loaded dissections according to various different criteria. They are presented in a tool (known as a 'Dissection Viewer') with the same structure as the Tigger Main Window, but with a slightly restricted set of menus. This allows multiple sub-selections to be made if required. The selection criteria are split into five different categories:

i) by house-keeping information: e.g. by source file, or by the source, author or status as represented in each dissection's header information;

ii) by rubric/code: acts as a 'find by code/rubric' option;

iii) by use: i.e. by whether a dissection's structure includes various different constructions. A submenu offers several of the more useful ones;

iv) ok: i.e. whether the dissection is still (after whatever error-checking operations have been applied) still known to be OK;

v) broken: this options pulls out dissections that have been found to be broken in one of several different ways; a submenu provides a set of options which roughly correspond to the error-checking levels themselves.

In addition, the option **Spawn→into all categories** applies all these selections and presents multiple separate sets of dissections.

The **Descriptors** and **Links** menu are the points from which different views of the descriptors and links, respectively, can be opened. All these options are covered in Sections 3 & 4 respectively.

The **GRAIL** menu includes facilities for exporting sets of GRAIL translations, or to examine individual translations (the option **Grail→Show Grail Translation** will open a new floating window that will reflect the Grail translation of the currently selected dissection). The option **Grail→Open Sanction Manager** opens a tool for using the dissections as a corpus from which to validate and extend the sanctions within the Common Reference Model; this is described in Section 7.2.

The **Help** menu provides details of any constraints that are in force, and indicates any known bugs. Help→About shows version information (which should be included in any bug reports or feature requests[27]), and details of the currently loaded 'common.txt' file.

[27] to be sent to: **galen-help@galen-organisation.com**

2.6.2 Dissection List

Each dissection that has been loaded is displayed, one per line, in the list that forms the major part of the Tigger Main Window. Dissections are shown in normal type if no problem has been detected, or in bold type if a problem (e.g. one of those as described in section 2.4) is known to exist. Note that they may not have yet been found, as some of the more computationally expensive problems are only tested in response to specific user prompting.

Options to change the format of the display are available using **Dissections → Display using…**; for example, it is possible to include the code associated with each dissection in the display. Use the menu button **sort** at the bottom of the tool to sort the listed dissections by either code or rubric.

Various operations are available on individual dissections; for example there is a facility to mark individual dissections as broken using the menu option **Dissections → mark broken….** This and other operations on individual dissections are also available from a "context menu" (middle or right mouse button menu) which is available when the mouse cursor is over the dissection list.

To open a view on a subset of all the loaded dissections, use the **Spawn** submenu. These options open tools (Dissection Viewers) that offers similar facilities to the Tigger Main Window for manipulating and viewing dissections, but only display dissections that match particular criteria. These filtering options apply to the set of dissections visible, so it is possible to use these options on successive views to produce more complex queries (e.g. *show me all the dissections from the file wcc.txt that use disjunctions, don't break any constraints, and that display some problem when trying to produce GRAIL translations*).

2.6.3 Dissection Editor

Normally, the right-hand part of the Tigger Main Window shows a 'Dissection Editor' view of the selected dissection (it can be removed by de-selecting the check-box **show editor** which appears at the bottom of the window). A similar view — in its own window — of a selected dissection can be opened by using the menu option **Dissections→Edit…**, or by double-clicking the selection.

The dissection editor shows details of the selected dissection in the dissection list. To easily move through the list, hot-keys are available, associated with the menu options **Dissections→Next/Previous**. If the **shift** key is held down whilst the selected dissection is changed, extended details are shown: any unmapped descriptors or links are underlined, and any descriptors with category discrepancies are shown highlighted in red (see Section 3.3.6).

A dissection editor has three main parts, as shown in Figure 13: the upper pane shows the header information of the dissection, the middle pane show the structural part of the dissection, and the lower pane shows details of any errors or problems that (have been found to) exist. The relative sizes of these panes can be adjusted by clicking and dragging the boundary area between two panes.

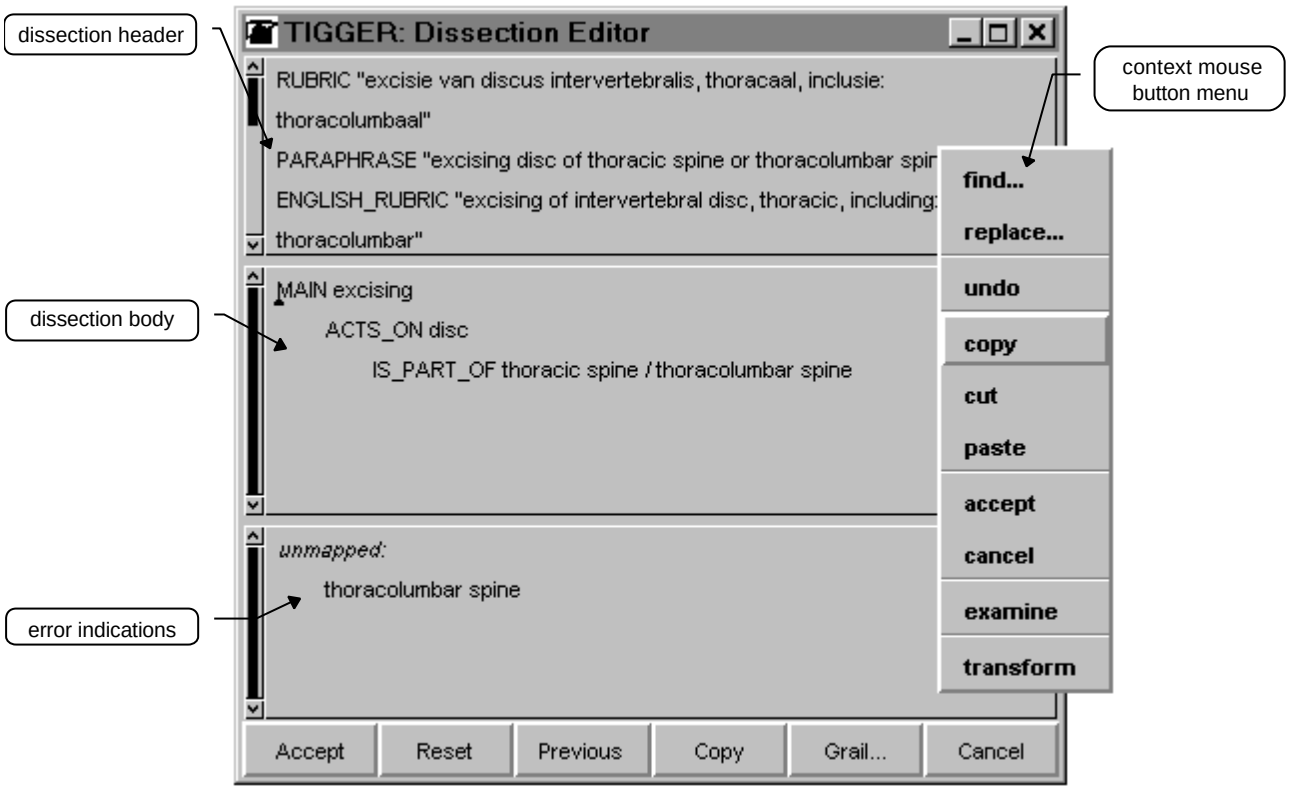


Figure 13 : Tigger Dissection Editor

To amend the dissection, type any corrections into the centre pane, and save the result by either using the context menu option **accept**, as shown in Figure 13, or by pressing the **Accept** button at the bottom of the tool. This mechanism is only intended to support small amounts of editing, perhaps to correct any typographical errors in manually-produced dissections. Note that, if any changes to dissections are made in this way, an updated set of dissections in textual form can be produced by using the menu option **Dissections → Export...** from the Tigger Main Window or a spawned set of dissections.

The **examine** option (also available in the lower pane) can be used to show more details of the selected descriptor or link: the GRAIL mappings (and other information) associated with the selected descriptor or link are presented using the Descriptor Manager or Link Manager tools. These tools are described fully in sections 3.3 and 4.4 respectively.

The transform option from the menu is used to apply a matching pre-defined transformation to the dissection: to globally replace links or descriptors, or to effect a more complex structural change. This is described fully in Section 2.6.7.1.

The **Reset** button at the bottom of the tool restores the dissection to the state it was at the previous accept; i.e. any changes are removed in the tool.

The **Previous** button provides the ability to view previous versions of this dissection, as a result of any changes that are accepted, or any transformations applied. Pressing the button once will undo the last change made. To *see* the last state but without changing it, hold the **Ctrl** key down whilst pressing the button; this will spawn a new editor with the previous state visible.

The **Copy** button copies the contents of the top and middle panes to the clipboard. Holding the **shift** key down when performing this operation will include the HISTORY header field. Holding the **Ctrl** key down will include any local OTHER fields (rarely needed). The **Grail...** button opens the Tigger GRAIL Output window (described fully in Section 6.7.1) which shows the Grail translation of a dissection, and provides facilities to browse the resulting GRAIL concept.

2.6.4 Tigger Preferences Tool

Tigger is configurable in several different ways. Various switches are available from the Tigger Preferences Tool, which is opened by choosing the menu option **File->Preferences...**. The tool is shown in Figure 14.

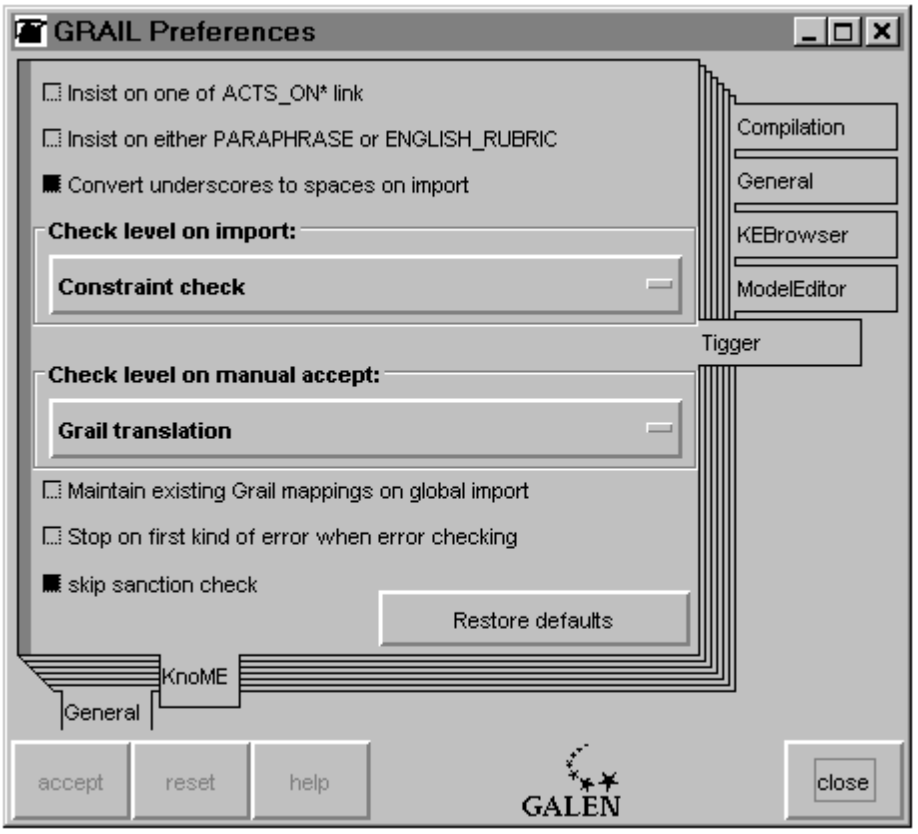


Figure 14 : The Tigger Preferences Tool

A description and motivation of each of these options follows:

- **Insist on one of ACTS_ON* link** and **Insist on either PARAPHRASE or ENGLISH_RUBRIC** The normal style rules for manually-authored dissections include the rules that each main deed should be described by at least an ACTS_ON link, and that a dissection (for efficient validation) must include either a paraphrase or an English translation of its rubric. For experiments and other work that result in large numbers of automatically-generated dissections, these restrictions may be removed;
- **Convert underscores to spaces on import** It is a overall design goal that descriptors be as close to clinician's preferred language as possible. However, at the beginning of the project, many descriptors were created with underscores separating words, rather than spaces. This option controls whether such descriptors are automatically converted on import;
- **Check level on import / manual accept** As described earlier in this chapter, there are several different ways in which dissections may be broken, and it takes significantly different processing time to determine some of these. It is thus useful to be able to specify to what level dissection should be checked, both on import (i.e. when typically many hundred or thousand dissections are being imported) or on manual accept (i.e. when a change is made to an individual dissection within Tigger);
- **Maintain existing Grail mappings on global import** Tigger maintains the set of descriptor and link mappings (colloquially known as "common.txt") To aid merging of different versions of this resource, this option controls whether existing Grail mappings are maintained or discarded during the import of common.txt;

- **Stop on first kind of error when error checking** This option controls whether or not to stop when one kind of error is detected (for example that unmapped descriptors are used). Often it is only possible to stop, in which case Tigger will, but sometimes, it is possible to carry on and check other kinds of errors (for example, if a manually-asserted constraint were found);
- **Skip sanction check** This option suppresses the checking of sanctions (between descriptor categories) when error-checking. This is useful when importing automatically-generated dissections.

2.6.5 Importing Dissections from file

Before Tigger can do anything of what it does with dissections, it must load them[28]. To import a set of dissections from a single file, use the menu option **Dissections->Import→Checking overlaps** or **Dissections->Import→no check; just add**. In either case, any dissections found will be added to the set already present. The first option checks incoming dissections for overlap with any already loaded, and offers choices for including or keeping any overlaps.

To load a set of files, each of which may well contain multiple dissections, use the Tigger Dissection Importer (as shown in Figure 15), by choosing Dissections→Import→Open Import Tool.

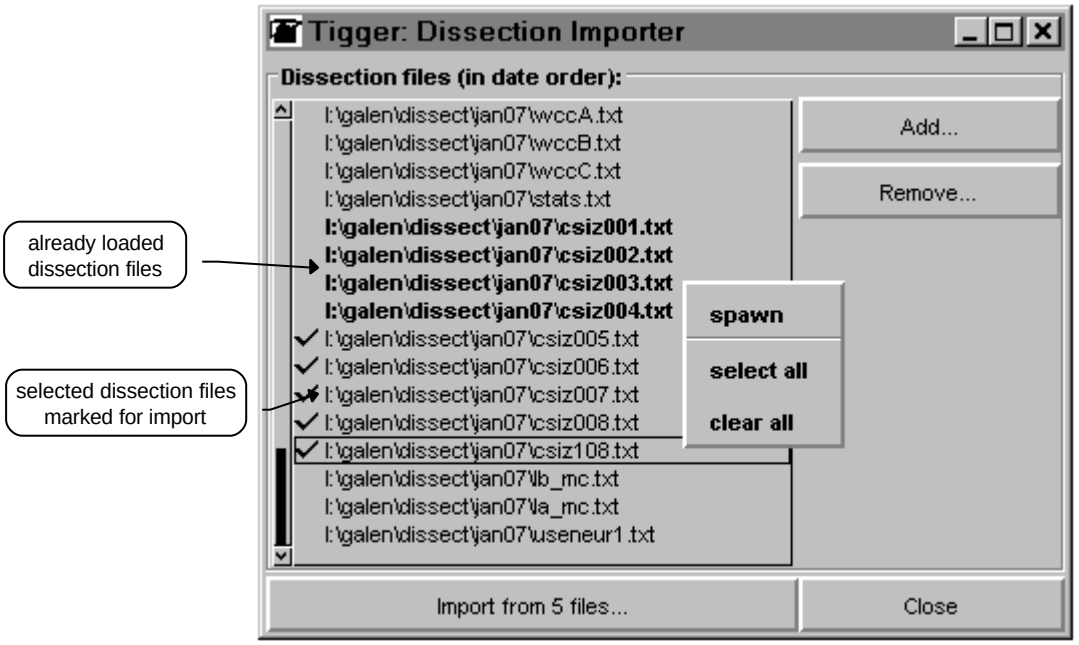


Figure 15 : The Tigger Dissection Import Tool

The tool presents a list of dissection files; to add files to this list, select the **Add...** button. Either enter a single file-name, or a directory (in which case all .txt files found there will be added to the list). Files already loaded are shown in bold face. One or more files may be selected, for subsequent **Import** or **Remove**al from the list. To make more than one selection, use the standard modifier-key combinations[29]. A context-menu is available (as shown) to select or clear all the files in the list. The **spawn** option opens a file editor on a specific file.

[28] At present, dissections are maintained persistently in the Tigger environment only by using the features of the Smalltalk environment in which Tigger is implemented: if the 'image is saved' - i.e. choose **File→Save as...** from the KnoME Main Window; then the next time Tigger is started all the dissections will still be present. In future, Tigger may well be connected to an external database.

[29] **Ctrl-select** to add / remove individual files; **Shift-select** to extend a selection

2.6.6 Tigger Global Export Tool

An important part of Tigger's functionality is managing the preferred set of links and descriptors (and other information) that is required to support users of the Classification Workbench. Within Tigger, these tasks are supported by tools described in Sections 3 & 4. There is an agreed interchange format within GALEN-IN-USE to pass this information around, described in GALEN-IN-USE Deliverable 7.2. To create a file in this format, colloquially known as "common.txt", use the Tigger Global Export Tool (shown in Figure 16), opened using **File→Global export…**.



Figure 16 : The Tigger Global Export tool

The top part of the pane allows you to enter (mandatory) version and descriptive information, whilst the series of check-boxes let you specify exactly what information you wish **Export**ed to the file. For descriptor categories and sanctions, you have the further option of specifying exporting the current internal state, or from another form of the information (either by interrogating the model, or inserting the contents of an external file, respectively).

The check-box **include translation rules** toggles the inclusion of the GRAIL start point (Section 6.5.1.1) and the descriptor mapping rules (Section 3.2.2) in common.txt.

The common.txt file can also be imported into Tigger (often useful when upgrading between versions of Tigger, for example) by using the menu option **File→Global import…**.

2.6.7 Tigger Dissection Management

Overall dissection management is an important task, especially as the number of dissections increases: both as a direct result of manual dissection authoring using the Classification Workbench, and as a result of (more experimental) techniques in automatically generating dissections (described in Section 8). This section describes several aspects of this process.

2.6.7.1 Dissection Transformation

[VUM03] describes a set of patterns, or templates, for the preferred style of manually-authored dissections. However, it still may be the case that some dissections need transformation to bring them to a common style:

- they might be in a new area which hasn't been extensively explored, and for which a template does not yet exist (or no agreement on a common style has been reached);

- to modify dissections if the preferred style changes;
- to modify dissections if the preferred set of links and / or descriptors changes;
- to help manage automatically-generated dissections that are more easily targeted towards a slightly different template.

Tigger provides two means of transforming dissections: for one-off changes to a single, or small number of dissections, the Tigger Dissection Editor can be used manually (as described in Section 2.6.3). However, this is unsuitable when larger numbers of dissections need to be transformed, or if the transformation is complex, or when the transformation may need to be repeated in well-defined circumstances in the future. In this case, Tigger provides facilities to build and maintain a library of different transformations and to apply these *en mass* to large numbers of dissections.

Defining a transformation

To define a transformation, open the Tigger Transformation Tool (**Transforms→Open Transformation Tool**) as shown in Figure 17. The tool shows a list of transformations (the *transformation library*) on the left, with the right-hand pane showing a description of the selected transformation. Many features are available from the context-button menu over the transformation list, as illustrated. A few of the transformations are hard-wired in the implementation, whilst the rest (and all user-defined ones) are specific instances of a *transformation type*.

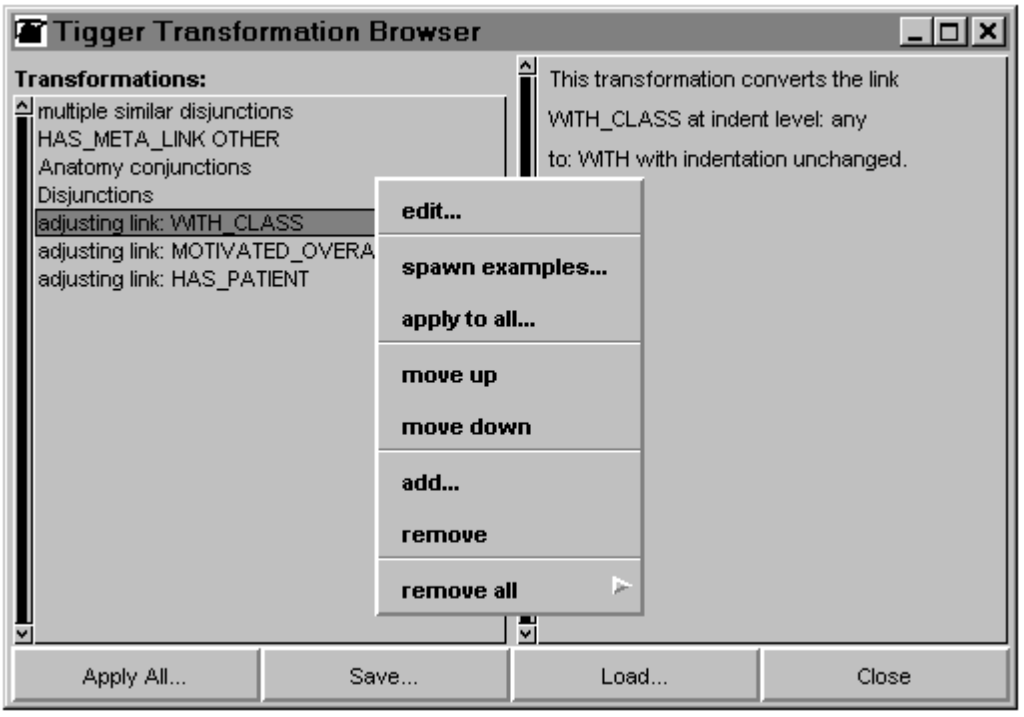


Figure 17 : The Tigger Transformation Tool

To add a new transformation, choose the option from the context-button menu **add**. A list of available transformation types is presented; a new transformation of your selected type is created and an editor opened for you to specialise the transformation. This sequence is illustrated in Figure 18.

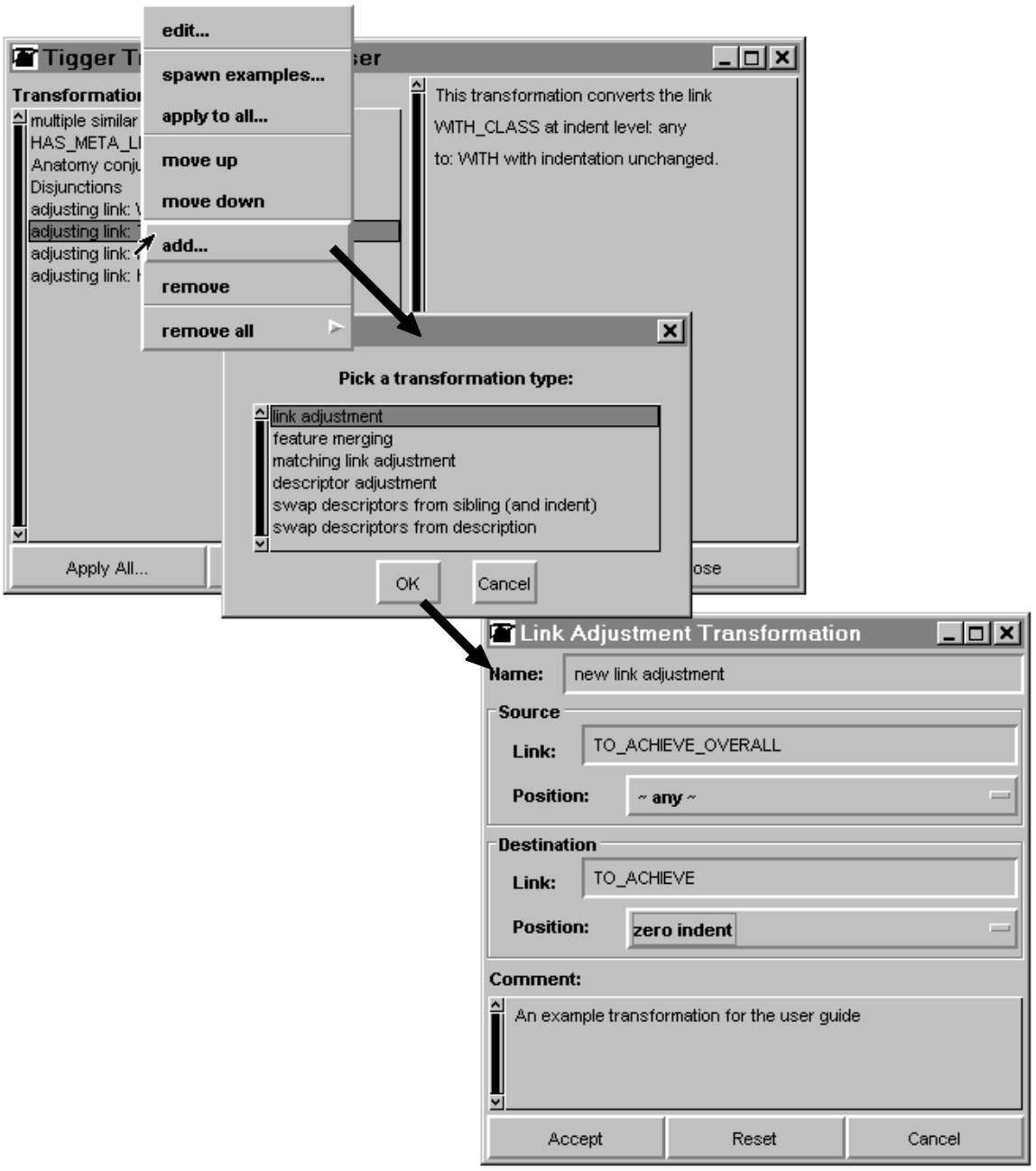


Figure 18 : Adding and editing a new transformation

As seen, there are six different transformation types, each of which has a specific editor - to see details of the specifics of each transformation, create a new one and then have a look at the resulting description in the transformation browser itself. In the editor illustrated, you can specify a 'link adjustment': a named link (if it appears in the position as specified in the upper pull-down menu) is transformed to another named link, and it may have its indentation within the dissection adjusted (controlled by the pull-down menu). Wildcards may be entered in the **Link** entry fields.

Note that the transformation is *structural*; i.e. any description of a link that is moved will move with it.

Applying a transformation

There are a number of ways to apply one or more of the transformations in the transformation library:

i) The dissection editor lower pane (see Section 2.6.3) will indicate if any transformation is applicable to the dissection it is displaying. To apply the transformation, use the context-button menu option **transform**. Holding the **shift** key down will cause any further transformations to be applied, until no more are applicable;

ii) From the Tigger Main Window, choose **Transforms→Spawn Transformable Dissections**, which opens a dissection viewer on all dissections for which at least one of the transformations in the transformation library is applicable. These may then be transformed individually using the method described in i);

iii) From the dissection browser, choose the context-button menu **apply to all...**. This will apply the selected transformation on any of the loaded dissections to which it applies. A dissection viewer will be then opened showing those dissections that have been transformed. To see the set of dissections for which the selected dissection is applicable (which could then be transformed individually using the method described in i), choose the option **spawn examples...**;

iv) Use the button **Apply All...** on the bottom of the Transformation Browser, which effectively calls method ii) sequentially on all the dissections in the dissection library.

Especially in the latter case, the order of the transformations is sometimes important. Use the context-button menu options **move up** or **move down** to promote or demote the selected transformation.

Saving the transformation library

The transformation library is specific to Tigger, and does not appear, for example, in the common.txt file. To save or load the library from file, use the buttons (**Save...** / **Load...**) at the bottom of the Transformation Browser, or the menu options from the Tigger Main Window **Transforms→Save/Load Transformation Configuration**.

2.6.7.2 Checking for duplicate rubrics

The Tigger Main Window option **Dissections→Search for duplicate rubrics** analyses all the loaded dissections, searching for duplicate rubrics that may be present. This is important for detecting overlap between work done at different centres, and for checking the work done at a single centre.

A tool (the Tigger Common Rubric Browser) is opened, as shown in Figure 19, to enable any duplicates to be examined more closely. The top of the tool provides options to select one of three different kinds of rubric duplication (taking into account the dissection's code and paraphrase); further details of each of these can be seen by choosing the **?** button associated with each option. The list at the left of the tool shows a number of sets of duplicate rubrics (the display shows one dissection from each set). Selecting one of these sets causes the right-hand part of the tool to show all the dissections in that set. In the example shown, it is clear that different codes have been assigned to the dissections; to examine an individual dissection in more detail, double click it to open a dissection editor, or use the **Spawn all** button to open a view on all the dissections. Comparison of the structure of the dissections can then be made, if required.

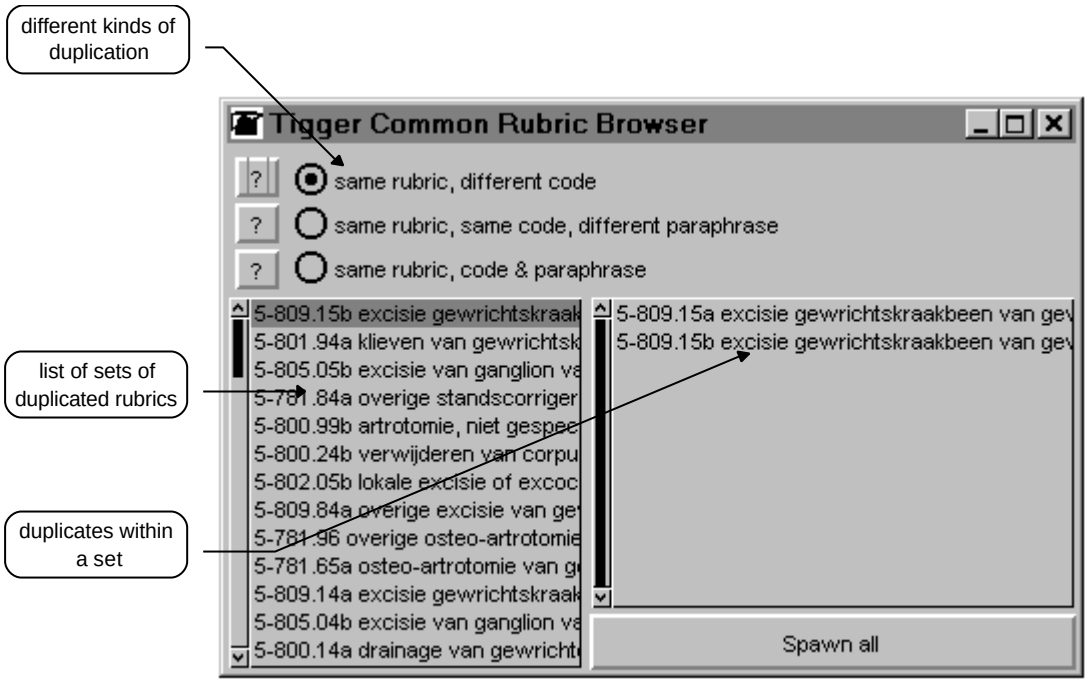


Figure 19 : The Tigger Common Rubric Browser

2.6.7.3 Reports and statistical information

To obtain overview statistics of the numbers of dissections loaded, organised by source-file and according to their error-checked state, use the menu option **File→Save statistics to…** which produces a textual report.

To export all loaded dissections into a new directory, together with a statistics report, use the menu option **File→Full report to…**.

The current status of all dissections received is maintained on the WWW at:

http://www.galen-organisation.com/mig/giu/private/status.htm

3 Descriptors: their management and their mapping to GRAIL concepts

3.1 Introduction

An important aspect of the GALEN-IN-USE interchange format is that it allows reference to clinical concepts by using agreed, well-known, clinical words and phrases (known as 'descriptors'), rather than insisting on GRAIL names or complex descriptions. To support this, Tigger provides a mechanism, and tools, to support the mapping between GRAIL concepts and descriptors.

Further aspects of descriptor management are also supported: for example marking some descriptors as being lexical variants of others.

3.1.1 Descriptor Categories

To support navigation within the descriptor list, descriptors belong to a (single) *category*: one of a small number of named types, such as: *Anatomy*, *Deed*.

There are two ways in which particular descriptors may acquire category information:

i. it may be assigned automatically, based on the GRAIL mapping of the descriptor. The set of descriptor categories are modelled in GRAIL and associated (using extrinsics information) with parts of the GRAIL model. More details of the categorisation algorithm are given in section 3.3.2.1;

ii. descriptors may have been annotated with their category within dissections in which they are used (using the syntax <category>:<descriptor>).

Note that a particular descriptor can only have a single category; Tigger checks on dissection import for descriptors with explicit category information that they match with any pre-existing category information.

3.1.1.1 Category sanctions

Tigger also maintains the set of *sanctions* that control the use of particular links between descriptors of a particular category.

3.1.2 Descriptor Interchange

Tigger itself does not insist that all descriptors must appear on a pre-determined list: if a new descriptor is used in an imported dissection, Tigger will notice it and make it available for mapping (described further in this section). However, the SPET uses a pre-determined

descriptor list, which is provided as part of the common.txt file, and it is an important role for Tigger to manage the set of descriptors (which acts as part of the controlled vocabulary that is provided to users of the Classification Workbench).

3.2 Descriptor Mapping

Normally, each descriptor is mapped to a single GRAIL concept (section 3.2.1 describes an unusual extension), whilst there may be many descriptors which all map to the same GRAIL concept. Before any dissection can be translated into GRAIL expressions, all the individual descriptors must be mapped into GRAIL concepts — either elementary or complex (named or un-named) concepts. Some mappings are trivial, in that the descriptor label is the same, or very similar, to a named GRAIL concept — however, manual validation must still take place to ensure that the labels mean the same. Other mappings may be more complex:

- the labels might have no lexical similarity;
- the appropriate GRAIL concept may not be reified;
- there may be additional modelling needed before the descriptor can be mapped.

Tigger supports all these kinds of mapping.

3.2.1 Contexts

Although it is normally the case that a particular descriptor always maps to the same GRAIL concept, there are occasions where it is useful to be able to have more than one mapping, disambiguating during the translation process using context. The context used is the link with which the descriptor is used. For example, the descriptor *bone* may have different mappings depending whether it is used to describe a piece of anatomy, or a material used in a graft.

3.2.2 Descriptor Mapping Rules

It is sometimes necessary to augment the provided GRAIL translation of a dissection, so that the resulting classification is as expected. For example, the word 'cutting' is, in itself, not specific to medicine, and may be mapped to the (also non-medical-specific) GRAIL concept **Cutting**. However, in order to ensure, for example, that a dissection in the context of surgical procedures using the descriptor 'cutting' is classified as a surgical deed, the descriptor more properly needs to be expanded during dissection translation to **Cutting which playsClinicalRole SurgicalRole**[30]. Tigger provides opportunity to specify these kinds of translation rules, which are part of configuring Tigger to operate within a particular domain (for example, the rules would be different if Tigger were dealing with dissection involving non-surgical parts of medicine, or indeed totally non-medical activities). The tools to support this are described in Section 3.3.5.

3.3 User Interface: the Descriptor Manager

The tool within Tigger for viewing and manipulating descriptors is known as the 'Tigger Descriptor Manager'. It may be opened in several different ways:

- to open a Descriptor Manager on the entire set of descriptors known, use the menu option **Descriptors → Open Descriptor Manager** from the Tigger Main Window;
- to open a Descriptor Manager showing only those descriptors that are used an in a particular set of dissections, use the menu option **Descriptors → Spawn descriptors used here** from any dissection list viewer;

[30] This description is necessarily brief; see GALEN-IN-USE Deliverable 5.1 for further details

- to open a Descriptor Manager on a particular descriptor (or set of descriptors) described by their name, use the menu option **Descriptors → Find**;
- to open a Descriptor Manager on a particular descriptor used in a dissection, use the **examine** option from a dissection editor's context menu (see section 2.6.3).

However it is opened, a Descriptor Manager has the same form, as illustrated in Figure 20.

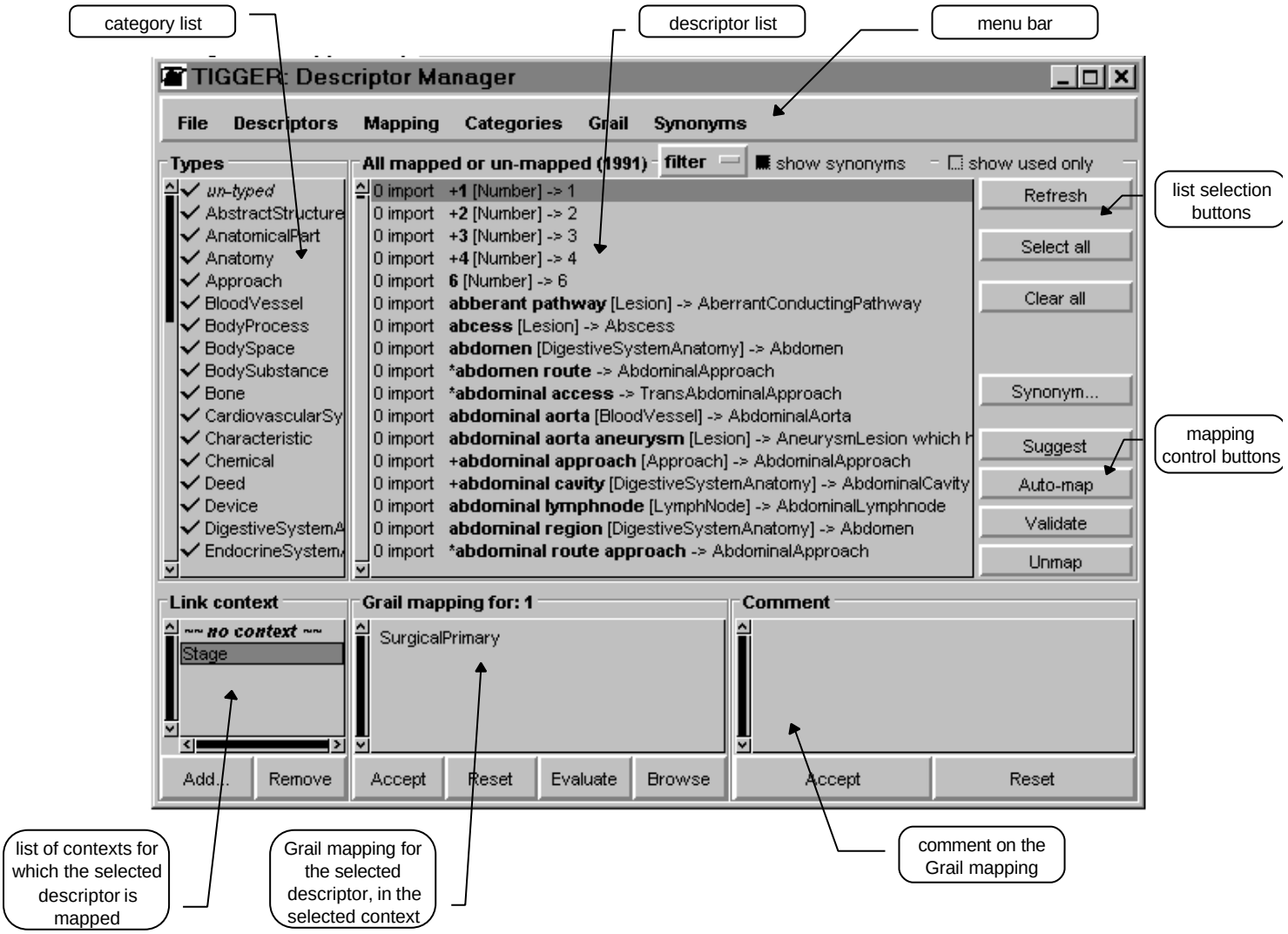


Figure 20 : The Tigger Descriptor Manager

A Descriptor Manager has three main parts:

- a menu bar giving access to all mapping, interchange and viewing functions;
- a descriptor list showing brief details of each descriptor: the list may have anything from one to several thousand entries, depending on how the Descriptor Manager was opened, and how the list has subsequently been filtered;
- a mapping area where GRAIL mappings for a particular descriptor can be viewed, entered or amended

3.3.1 Descriptor List

Each line of the descriptor list represents a single descriptor: various pieces of information are presented in each entry:

- usage: the number of times the descriptor has been used in the set of dissections currently loaded into Tigger (to open a view on those dissections, use the menu option **Descriptors → Show usage…**; this option is also available from the context button menu over the descriptor list);
- origin: whether the descriptor was first seen in an imported descriptor list (as part of the common.txt file), in a dissection that was loaded, or whether it was manually added;
- mapping kind: the source for the mapping (if not from an imported interchange file);
- label: how the descriptor is used in a dissection (shown in bold);
- category: the descriptor category is shown within square brackets, if there is one;
- mapping: the GRAIL concept to which it is mapped (only the standard mapping is shown even if more context-specific mappings are present);
- lexical variants: if the selection is marked as a lexical variant of another descriptor (see section 3.3.4) it is preceded by an asterisk (*****); if other descriptors have been marked as a lexical variant of it, it is preceded by a plus sign (**+**).

For example, the information being presented for the highlighted descriptor in Figure 20 is: *the descriptor '**1**' is not used in the current set of loaded dissections; it appeared in the imported common.txt file, it is of category anatomy, the standard mapping is to the GRAIL concept 1, and other descriptors have been marked as a lexical variant of it.* Note that the mapping shown at the bottom of the tool is to a different GRAIL concept (**SurgicalPrimary**), which is its mapping in the context of its use after the link `Stage`. This is described further in the next section.

3.3.1.1 Multiple link context usage

Some of the lines in the descriptor list are shown in italic; this indicates that the specified descriptor has been used in the context of more than one link; to see which ones, use the context menu option **link usage…**. Note that you may only add a new context-sensitive link mapping if the descriptor has already been used in the context of more than one link; the **Add…** option presents a list of those links.

To get a textual report of those descriptors that are used in more than one context, use the menu option **File→Suspect usage report…**; this report can be useful for validation.

3.3.1.2 Filtering the descriptor list

There are several different ways in which the list of descriptors in a Descriptor Manager can be filtered:

- the check box **show used only** removes from the list any descriptors that are not used; note that if the **shift** key is held down whilst switching this check box on, it will only display descriptors that are *not* used;
- the check box **show synonyms** toggles the inclusion of any descriptors that are marked as synonyms or lexical variants of other descriptors;
- the menu options under **Descriptors→Show** allow you to show descriptors according to their mapping status: whether they are mapped, un-mapped or mapped but un-validated (more details of these statuses is given in section 3.3.2); this function is also available from the pull-down menu labelled **filter** at the top of the descriptor list;
- filtering descriptors according to their category: the descriptor list only contains descriptors whose category is selected from the category list — there are context menu options to **clear all** or **select all** the categories, or individual categories can be selected or de-selected with the mouse. The first category in the category list is always *"un-typed"*, representing those descriptors which have no category assignment.

3.3.1.3 Sorting the descriptor list

The menu option **Descriptors→Sort** presents options to allow the list of descriptors to be sorted alphabetically (the default); according to use (descriptors used in the most dissections appear first), or grouped into categories.

3.3.2 Descriptor mapping

All the functionality described thus far in this section provides means to view and organise sets of descriptors — all to support the higher-level task of mapping those descriptors to GRAIL concepts. This section describes the facilities within Tigger designed to support that task.

There are several different routes by which a mapping might be produced:

- manual: given knowledge of the model, and the intended meaning of the descriptor (possibly obtained by viewing the dissections in which it is used), a GRAIL expression (a single GRAIL knowledge name, or a complex GRAIL concept) can be manually entered into the GRAIL mapping area in the Descriptor Manager (shown in Figure 20), and **Accept**ed using the button or the context button menu option. The **Evaluate** button evaluates the contents of the mapping area, to check that the GRAIL concept exists in the model; the **Browse** button does that, but also opens a KnoME Concept Browser on the result;
- auto-map: many descriptors have the same, or lexically similar labels to the GRAIL concepts to which they ought to be mapped. To find those cases, Tigger provides an 'auto-map' mechanism which uses lexical techniques to search the CRM for possible mappings for a descriptor;
- suggest: in the case where an auto-map is not found and where additional modelling is required, it is often useful for Tigger to suggest a lexical form for the mapping, which can then be added to the CRM. Tigger has an (in-built) knowledge of the style of GRAIL naming conventions, which it uses to produce these suggestions.

  To support the additional modelling that is required in these cases, the menu option **GRAIL→export GRAIL source** provides two ways to export GRAIL source files. The first provides fully compilable GRAIL source that adds new elementary concepts for each suggestion (using as a parent the GRAIL concept mapped to the descriptor category — see section 3.3.2.1). The second provides non-compilable source files that are in a form useful for supporting manual additions to the CRM using the standard KnoME tools (this may be useful for the cases where an appropriate concept for the mapping can be reified, or when more detailed modelling is required than that available from the stop-gap method described above of adding new elementary concepts).

The **auto-map** and **suggest** options are available:

- from the menu-bar option **Mapping**;
- via hot-keys;
- from the context mouse button menu;
- by using the mapping control buttons on the right of the Descriptor Manager window.

Before any mappings added using the **auto-map** or **suggest** methods become available to the dissection translation process, they must be validated (using the **validate** option) as a separate step; any mappings added manually are considered to have been implicitly validated. It may well be the case that this model of validation will need to be extended when mapping is done in many different sites. Figure 21 illustrates the different ways mappings may be added.

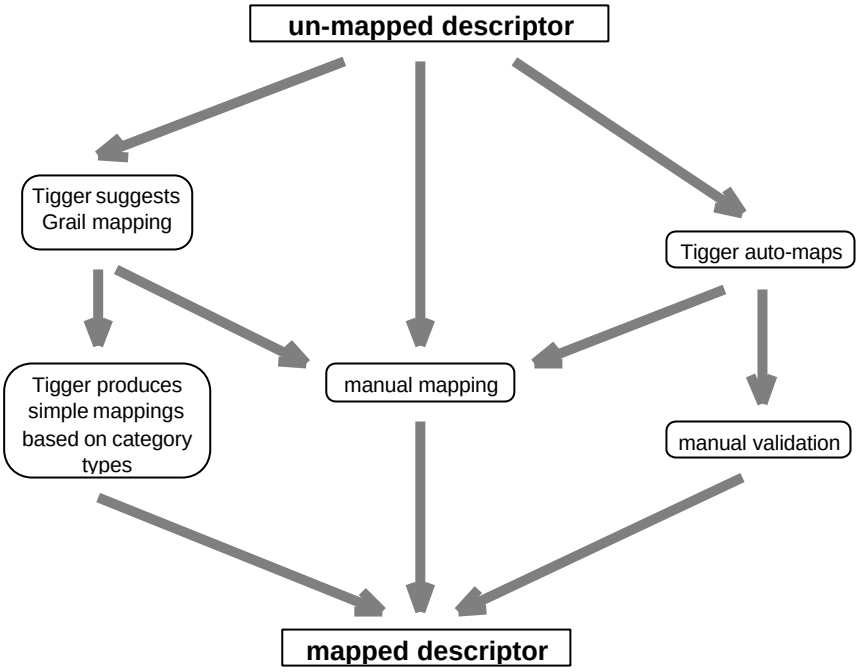


Figure 21 : Routes towards mapping of descriptors

The **auto-map** and **suggest** options apply to all the descriptors in the visible list that have been selected (using the mouse to add and remove individual selections, or by using the list selection buttons on the Descriptor Manager window).

3.3.2.1 Descriptor categorisation

As mentioned in the introduction to this section, descriptors are assigned a category, based upon their GRAIL mapping. The option **Categories→Recategorize descriptors…** assigns such a category for each of the selected descriptors in the Descriptor Manager. It works by searching for the lowest categories (represented as GRAIL concepts) that are attached using extrinsic statements, starting at the GRAIL mapping of the descriptor. If more than one are found, they are disambiguated by using numerical ranks that are also attached to the categories using extrinsic statements.

3.3.3 Descriptor usage

As part of the process of managing and mapping one or more descriptors, it is important to see the context in which they are used in the set of loaded dissections. The Descriptor Manager supports this by providing the following ways of seeing where a particular descriptor is used:

- the menu-bar option **Descriptors->Show usage**;
- the context-button menu option **dissection usage…**;
- double clicking the selected descriptor.

All of these methods produce a Dissection Viewer (as described in Section 2.6) containing only those dissections that use the descriptor.

Note that this mechanism is refined if the Descriptor Manager is opened as a result of choosing the menu option from the Dissection Viewer **Descriptors→Spawn descriptors used here**. In this case, the descriptor usage options in the resulting Descriptor Manager presents dissections that use the descriptor *and* are also present in the original list of dissections from which the Descriptor Manager was opened. This refining behaviour can be overridden by holding the shift key down when applying any of the three ways of showing descriptor usage, as described above.

3.3.4 Lexical Variants

Tigger is used to create a set of preferred descriptors; this is not done is isolation, but it takes into account those descriptors which have been first introduced by users of the Classification Workbench in their dissections. Part of the task of managing the descriptor list is marking some descriptors as lexical variants or synonyms of each other.

This can be done manually, for example by using the button **Synonym...** on the right of the Descriptor Manager, to add or remove synonym information for a single descriptor. Alternatively, Tigger can examine the GRAIL mappings of descriptors and present those descriptors that have been mapped to the same GRAIL mapping, as candidates for being marked as lexical variants or synonyms. The menu option **Synonyms→Open Synonym Tool** opens the tool, as shown in Figure 22.

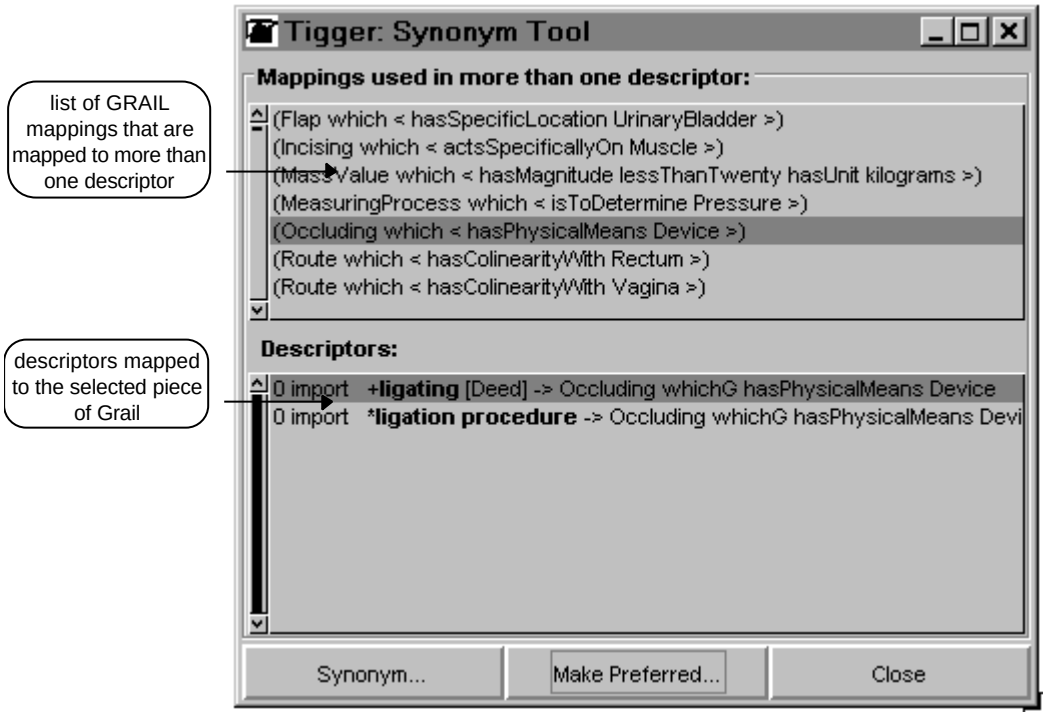


Figure 22 : The Tigger Synonym Tool

The button **Make Preferred...** marks all the non-selected descriptors as synonyms or lexical variants of the selected descriptor. The button **Synonym...** can be used to adjust synonym information for an individual descriptor.

Once any descriptors have been marked as synonyms, they can be replaced in all the dissections in which they are used, by choosing the menu option from the Descriptor Manager **Synonyms→Globally replace all synonyms in all dissections**. A textual report of all synonym information held (for validation) can be obtained by using the option **Synonyms→Export textual synonym report...**.

3.3.5 Specifying Descriptor Mapping Rules

As described in Section 3.2.2, Tigger maintains a set of descriptor mapping rules that form part of defining the context in which Tigger operates. To set up these mapping rules, choose the menu option from the Tigger Main Window **Descriptors→Mapping Rules→Open Mapping Rules Tool** to open the tool as shown in Figure 23.

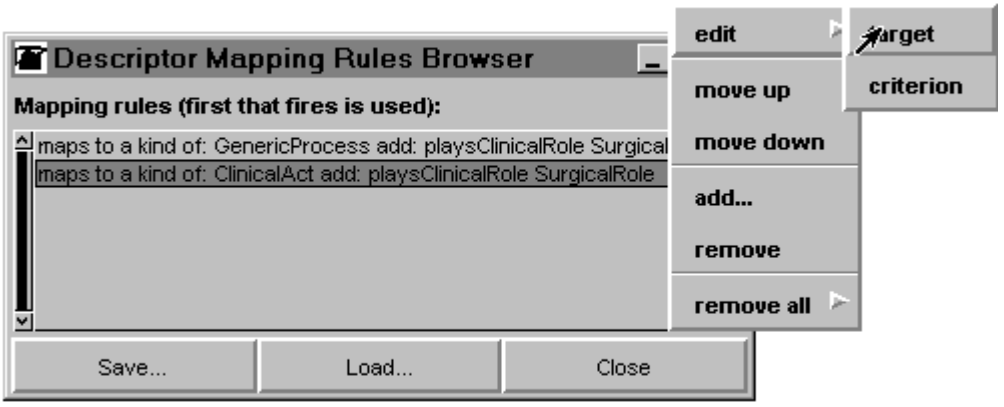


Figure 23 : The Tigger Descriptor Mapping Rules Browser

The tool shows a list of mapping rules, one per line, together with the context-button menu from which most functions are available. The illustration shows the option used for editing the *target* of the selected rule; each rule has the same form[31]: if a descriptor is mapped to a GRAIL concept which is subsumed by the *target*, then the *criterion* specified in the rule is added to the descriptor's GRAIL mapping during dissection translation.

3.3.5.1 Saving descriptor mapping rules

The set of descriptor mapping rules can be imported to, and exported from, Tigger as part of the common.txt file (as described in Section 2.6.6), or separately: by using the (**Save...** / **Load...**) buttons at the bottom of the Descriptor Mapping Rules Browser, or by using the menu options from the Tigger Main Window **Descriptors→Mapping Rules→Save/Load Descriptor Mapping Rules**.

3.3.6 Checking descriptor category discrepancies

Options are available from the Tigger Main Window menu **Descriptors→Category Discrepancies** to view all those dissections that have been loaded that contain different categories from those already known to Tigger (option **view dissections...**), or to extract a summary report of such discrepancies (option **view report...**). Tigger maintains and adds to this information as new dissections are imported; to clear the stored set of discrepancies, choose the option **clear→confirm**.

Note that when using a Dissection Viewer (perhaps after choosing the option **view dissections...** as described above) to view dissections, holding the **shift** key down whilst selecting a dissection will cause the display in the associated dissection to show any category discrepancies to be highlighted in red.

[31] experience so far has not suggested any other rules, though other kinds may emerge in the future

4 Links: their management and their mapping to GRAIL attributes

4.1 Introduction

The previous section has described one major simplification between GRAIL as a modelling language, and the dissection format used in GALEN-IN-USE: the distinction between descriptors and GRAIL concepts. The second simplification, and arguably the more important one, is the distinction between dissection links, and GRAIL attributes. That is the subject described in this section.

However, the relationship between links and GRAIL attributes is more complex than that between descriptors and concepts, in the following ways:

- cardinality: the vast majority of descriptors are mapped to a single GRAIL concept, and hence the translation process from dissections to GRAIL expressions is trivial in that regard; in contrast, virtually all links map to more than one (and often to many more than one) GRAIL attribute. To perform translation, there is much use of context, and of querying the CRM, to determine the appropriate link mapping in a particular dissection;
- structure: often a single link needs to be expanded into a more complex structure, consisting of an **attribute—concept—attribute** 'chain' (or a more complex chain with additional intervening concepts).

This complex relationship between links and attributes is partly a consequence of the fact that in the CRM, there are a large number of attributes, some of which are an artefact to cope with the limitations of cardinality control in GRAIL. Partly, however, we wish to make the intermediate representation as simple as possible, given the constraints that the dissections must be able to be transformed into GRAIL. The deliberate decision has been taken to provide simplicity in terms of the links, by hiding complexity within Tigger.

4.2 Link Mappings

The simplest form of link mapping is the case of a one-to-one mapping between a single GRAIL attribute and a link. In this case, the transformation during dissection translation is trivial.

4.2.1 Expansion into GRAIL ‘chains’

The next stage of complication is when a link is mapped into a single attribute chain; in this case transformation is still simple, but the chain is expanded into an embedded complex concept.

4.2.2 Multiple mappings

It is often the case that there may be more than one potential GRAIL attribute (or attribute chain), that might be applicable: in this case, the choice between them is made by checking sanctioning information in the CRM: only attributes that result in sanctioned criteria in the resulting GRAIL expression can be used.

For example, the link `ACTS_ON` may be appropriately linked to the GRAIL attributes **LocativeAttribute**, **actsMultiplyOn**, **actsSpecificallyOn**, or **actsOn**, depending on the context. The mechanism for performing this disambiguation is explained more fully in section 6.

4.2.3 Contexts

The technique for disambiguating between multiple attributes or attribute chains (as described in the previous section) goes some way to providing Tigger with the means for choosing the right attribute (or attribute chain); however, additional contextual dependency is sometimes also required. In this scenario, the descriptor to which the link is attached, and/or the descriptor which follows the link, may play a part in choosing the appropriate attribute or attribute chain.

For example, the link `HAS_LOCATION` is mapped to the GRAIL attribute **hasSurfaceVisibility** if the descriptor following it is mapped to a concept which is subsumed by the GRAIL concept **SurfaceVisibilityStatus**; otherwise, it is mapped to one of the attributes **LocativeAttribute, hasMultipleLocation**, **hasSpecificLocation**, **hasLocation**, **hasSpecificSublocation** or **hasSublocation**

As part of the overall mapping for a single link, Tigger first checks these kinds of context to select a set of attribute or attribute chains, from which disambiguation then takes place using the sanctioning mechanism described.

4.3 Link Management

Tigger provides a graphical tool (described in section 4.4) for authoring, amending and viewing the link mappings in the system, and for managing them in terms of their part of the 'controlled language' that is supplied to users of the Classification Workbench in the common.txt file.

Harmonising the use of different links across sets of dissections can be achieved using the dissection transformation mechanism described in section 2.6.7.1.

4.4 User Interface: the Link Manager

The Tigger Link Manager is a tool for viewing the link mappings in the system, adding to them or amending them, and for producing views on how particular links are used in dissections. The Link manager is opened either by using the Tigger Main Window menu option **Links → Open Link Manager**, or by using the **examine** option from a Dissection Editor. It is illustrated below in Figure 24:

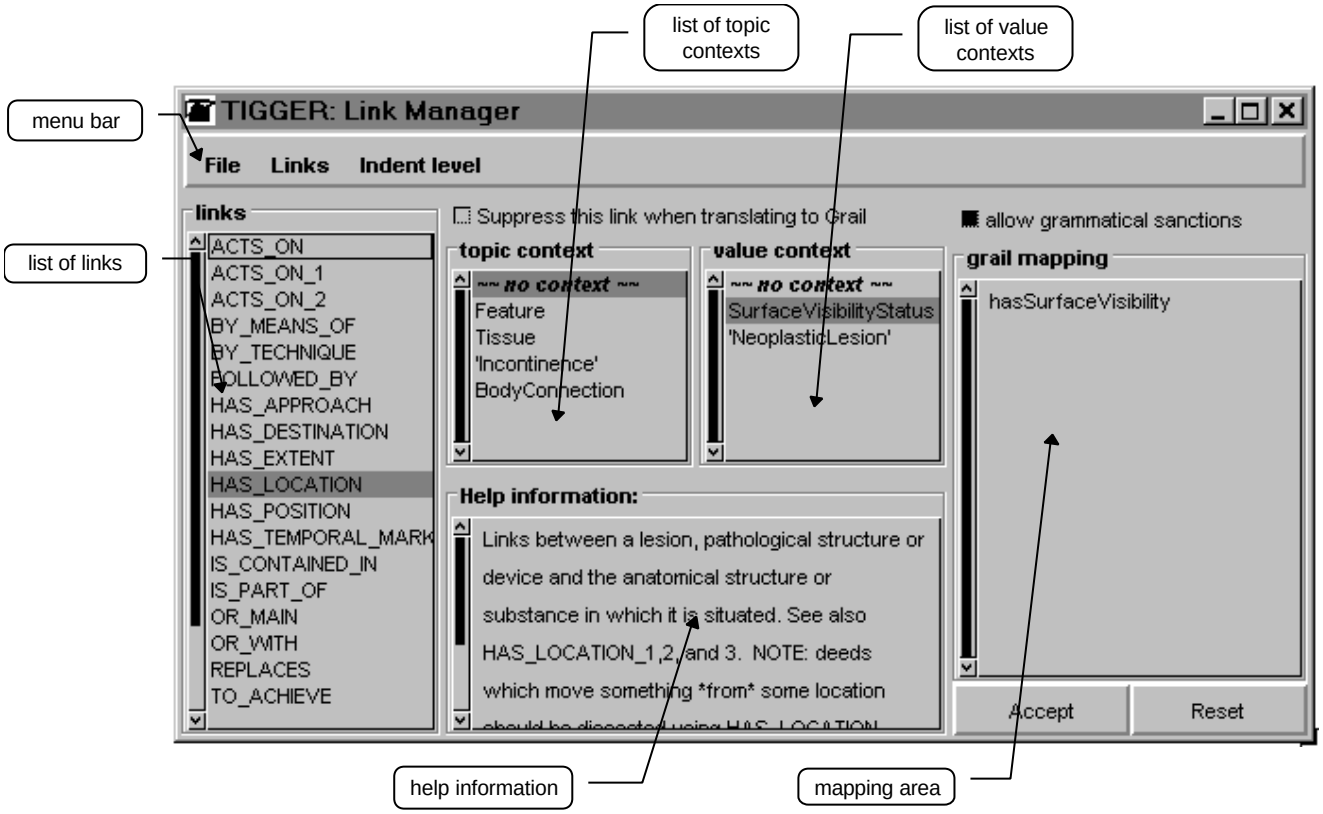


Figure 24 : The Tigger Link Manager

The link list to the left of the Link Manager shows all the links that are used in the dissections currently loaded; the menu option **Links→ Show** provides options to filter the list, e.g. to include those links that are not currently used. As there are far fewer links than descriptors, and that number grows only slowly, there are fewer filtering and sorting facilities than are available in the Descriptor Manager.

The list of topic contexts refers to the GRAIL concepts used to control the translation to GRAIL attribute chains, depending on which descriptor the link is describing; the list of value contexts controls the translation according to the kind of descriptor that appears after a link. The use of these contexts is described in Section 6. The check-box **Suppress this link when translating to GRAIL** is used to indicate to Tigger any links that should not take part in the dissection translation process[32].

Links which are introduced using the `HAS_OTHER_FEATURE` mechanism are shown within square brackets, and appear together at the end of the list. Links marked with an asterisk are not mapped, but they appear to be in the mapping area: however, the mapping display is suggested by Tigger based on an in-built knowledge of the default mechanism for mapping such links. To accept such a suggestion, use the **Accept** button.

Note that the various panes of the Link Manager can be resized by selecting and dragging the area between two panes.

[32] whilst this information is contained in common.txt, it is not, at the time of writing, interpreted by the Classification Workbench when it translates dissections; instead, it is considered that only the well-known link `WITH_OPTIONALLY` is treated this way

4.4.1 Help information

The area marked **Help information** is used to enter information about the link that is intended to be useful for users of the Classification Workbench; it is passed through in the common.txt file. To enter or amend any help information, simply type in the area and accept it using the **accept** option on the context-button menu.

4.4.2 Mapping area

With one topic context and one value context selected (note that there may be different lists of value contexts associated with each topic context), the mapping area is enabled. It shows the set of attributes (or attribute chains) mapped to the link, in the selected contexts.

The check-box in the mapping area allow grammatical sanctions is used to toggle the translation process' use of grammatical sanction information in the model (described in Section 6.5.2)[33].

The main part of the mapping area is a text editor: the mappings are shown one attribute or attribute chain per line (shown in Figure 25).

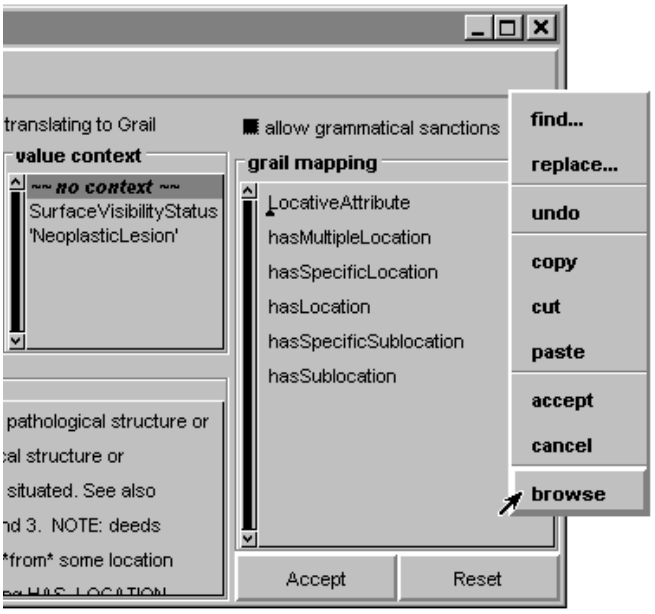


Figure 25 : GRAIL mapping area showing multiple attributes

The **browse** option opens a KnoME Concept or Attribute Browser (as appropriate) on the selection within the mapping area (to confirm presence in the model, or to explore the model).

An attribute chain is shown as a space-separated list starting with an attribute, followed by any number of concept—attribute pairs (an example is shown in Figure 26). To amend or add to a mapping, type into this area, in that format, and use the **Accept** button or the **accept** option from the mouse context button menu.

[33] whilst this information is passed out of, and into, Tigger in common.txt, it is not, at the time of writing, used by the Classification Workbench dissection translation process, which operates as if the check-box were always checked

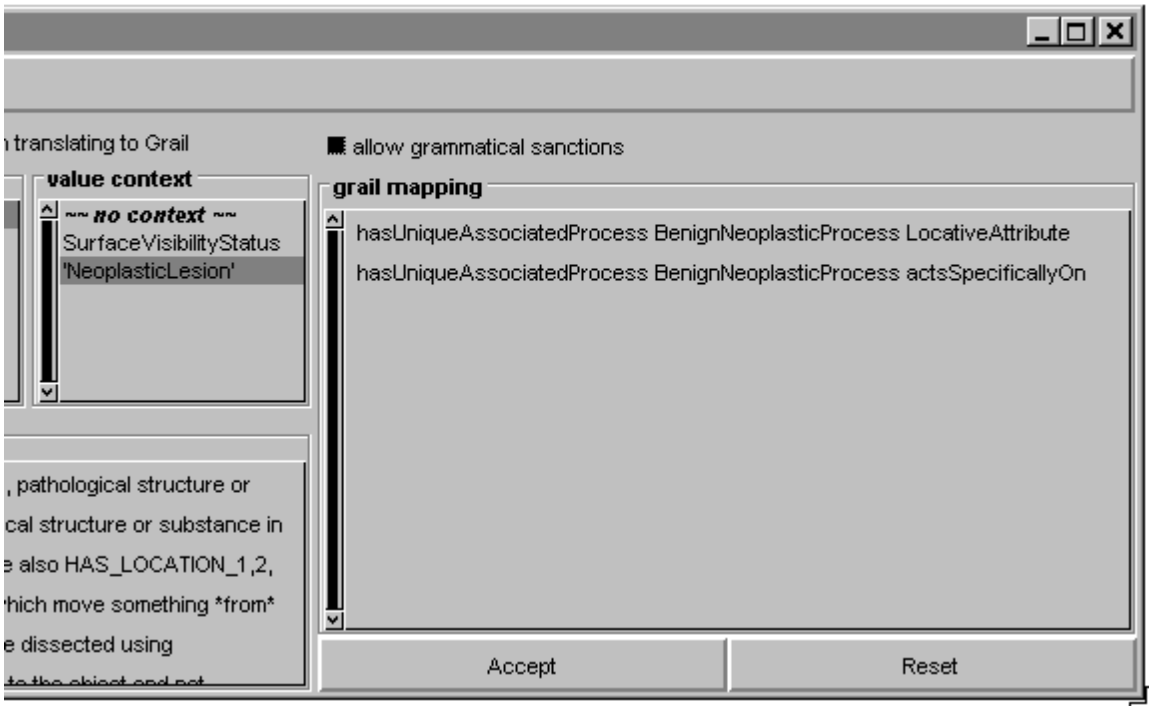


Figure 26 : GRAIL mapping area showing attribute chains

Note that any attributes or concepts entered into the mapping area must exist in the model, before a mapping is accepted. Links imported from common.txt that include attributes or contexts that are not in the model will cause the appropriate link to be shown in bold type in the link list.

4.4.3 Link usage

It is important whilst mapping links to be aware of the context in which they have been used (this is especially important for links introduced for the first time by users of the Classification Workbench). Double click on a selected link, or choose **Links->Show usage...** to spawn a dissection viewer on all the loaded dissections that use the link.

5 Managing Collaborative Work

5.1 Introduction

The previous three sections have described how to manage dissections and to map links and descriptors. As described in those sections, Tigger provides methods to export the entire set of dissections, link mappings and descriptor mappings, for interchange between other parts of the project, or to transfer between different version of the Tigger software.

Whilst sufficient for the purposes described, these facilities to not well support the scenario of more than one clinician using Tigger to manage or amend different parts of the overall set of dissections or mappings. This section describes specific facilities to support that task, based on the notion of "packages".

5.2 Tigger "packages"

A "package" is a set of changes made to dissections, links or descriptors: new, amended or annotated dissections, new or amended link & descriptor mappings. A package is a logical piece of work that is typically the result of an individual clinician working with Tigger, based on an existing set of dissections and link & descriptor mappings. It is often the case that more than one clinician might be working with Tigger — producing such packages — based on the same underlying set of dissections and mappings.

Tigger provides facilities to:

- identify particular changes as part of a particular package;
- produce transportable export versions of packages;
- load a package into Tigger;
- examine the contents of a package without loading its contents;
- highlight any potential conflicts or overlaps between work done in different packages.

The "Tigger Package Viewer" is a tool for working with packages; options are available under the Tigger Main Window menu **Changes** to identify packages and open a Package Viewer.

5.3 Gathering work into a package

Work within Tigger is identified as belonging to a particular package by "check-pointing" at a particular point in time, and identifying yourself by name. Any subsequent changes are gathered together as a package. To perform a checkpoint, use the menu option **Changes→Reset**

changes... from the Tigger Main Window. You will be asked to enter your name to act as an identification for any subsequent work.

Any subsequent changes to the dissections, links or descriptors within Tigger will be managed as being part of a package. To see what changes have been made since the last check-point, use the menu option **Changes→View my changes...** which opens a Tigger Package Viewer as shown in Figure 27:

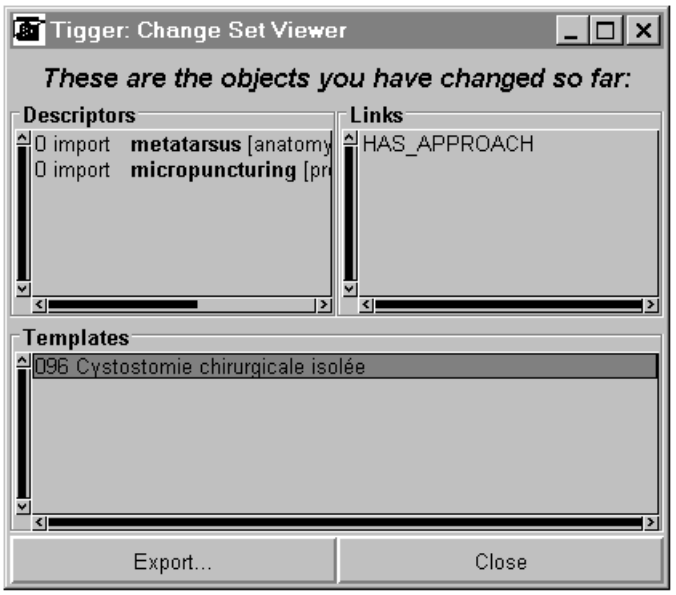


Figure 27 : Tigger Package Viewer showing your changes

This example shows that two descriptors, a single dissection and a single link have been changed since the last check-point; context mouse button menu options are available to examine any element of a package.

5.4 Exporting and transporting a package

To export the contents of a package, use the menu option **Changes→Export my changes...**. You are prompted for a name, which acts as the name for a folder (directory) where textual versions of your changes are kept. The advantage over using the standard Tigger export functions is that only the dissections, links or descriptors that have changed are exported, rather than the entire set. This makes for physically smaller material to transport, as well as being easier to integrate subsequently).

To transport a package, simply move the folder and all its contents. A folder can be combined and compressed into a single file for email or electronic transfer; the ZIP format is recommended as a common multi-platform utility.

5.5 Importing packages

There are two ways of importing work contained in a package. The first is to directly add the contents of a package to Tigger: dissections, descriptors or links that might be contained in the package may over-write any that might be already present. This option is useful if you are confident that no conflicts are liable to occur. To load a package in this way, use the menu option **Changes→Load changes...**; specify the (folder) name of the package when requested.

A more cautious approach is to "examine" a package before applying any of its contents. Rather than applying the contents of a package directly, the package is read, and a Tigger Package Viewer is opened on its contents. The package can be then selectively tested for

conflicts and applied as necessary. To examine a package, use the menu option **Changes→ Examine changes…**.

The entire contents of the package can be applied by using the **Apply…** button; another option is to examine individual elements of the package and reject them if required. There are similar context mouse button menus over each area of the tool, illustrated below in Figure 28 for the set of descriptors in the package.

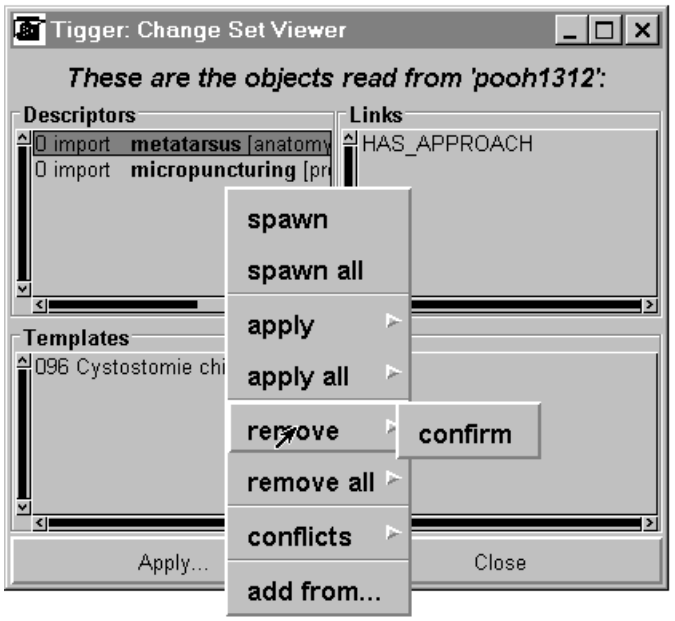


Figure 28 : Examining elements of a package

The **spawn** options open a Descriptor Manager on the selected descriptor or the set of descriptors (appropriate tools are opened from the menus over the link and dissection lists).

The **apply** options (with confirmation) will add the specific elements of the package to Tigger.

The **remove** options remove elements from the package: this is useful either to remove changes if examining your own package (rather than an imported one), or if conflicts have been identified.

To help identify conflicts, the **conflicts->report to...** option produces a textual description of differences between the work contained in the package, and the elements currently loaded into Tigger.

To use the Package Viewer to investigate conflicts contained in standard Tigger exchange format files, the option **add from…** adds to the package by reading the contents of standard dissection, link or description interchange files.

6 Dissection Translation

6.1 Introduction

The previous sections have hinted at the process of translating dissections into GRAIL expressions: the graph structure represented by the dissections is expanded into an analogous graph structure in GRAIL; descriptors are mapped to GRAIL concepts, whilst links are mapped to a single attribute, or to a attribute chain which is expanded into a more complex, nested, graph structure in GRAIL.

However, this is something of a simplification: it is often the case that a single dissection is expanded into more than one GRAIL concept expression. There are two main reasons for this:

- disjunctions between descriptors are translated by duplicating the dissections in which they appear;
- there are some links that are specified to have "special" behaviour, to cope with some of the idiosyncrasies of existing coding schemes which are being represented.

In addition, there are some parts of the dissection that do not get transformed into parts of the resulting GRAIL expression; they are logically redundant, but are in the original coding schemes to act as indications to human users.

This section describes how these mechanisms are implemented, describes in more detail the process introduced so far, and also introduces some additional features of the translation process. Finally, the ways in which the resulting GRAIL expressions may be compiled, and the resulting classifications viewed, are described.

6.2 Overall translation mechanism

The sequence of operations that takes place to translate a dissection (**T**) is as follows:

- any part of **T** that is redundant is removed to yield **T1**; this is any sub-tree of the dissection that uses a link marked as being suppressed (as described in Section 4.4)[34];
- **T1** is expanded due to the use of any "special" links to produce a dissection set **T1set1**; this is described in section 6.3;
- each dissection in **T1set1** is expanded due to the use of disjunctions within descriptors, to yield a dissection set **T1set2**; this is described in section 6.4.2;
- each dissection in **T1set2** is translated into a single GRAIL expression, and annotated with extrinsic information originating from **T**; this is described in section 6.5.

[34] at present the Classification Workbench dissection translator only treats the well-known link `WITH_OPTIONALLY` in this way

6.3 Special Links

There are two special links (specified in [VUM03]) which are treated differently from other links. Their effect is not to contribute to the structure of an individual GRAIL concept; rather, they act to split up and expand an individual dissection into a dissection set. The behaviour of these links is built into Tigger, and so they do not need to be mapped using the Link Manager. These two links are `OR_MAIN` and `OR_WITH`; they are both "zero-level indent" links.

Details of this and other knowledge that is built into Tigger can be obtained using the Tigger Main Window's **Help** option.

6.3.1 OR_MAIN

The use of the link `OR_MAIN` splits a dissection into two, each of which has the same dissection header. One dissection is formed of all the links in the original up to and not including the `OR_MAIN` branch, whilst the second is formed with `OR_MAIN` transformed to `MAIN`, and including all the links from `OR_MAIN` from the original. For example:

```
<dissection header>
MAIN x
OR_MAIN y
```

is transformed to:

```
<dissection header>
MAIN x
<dissection header>
MAIN y
```

The link `OR_MAIN` can be used in a dissection more than once, in which case the dissection is split up in an analogous way.

6.3.2 OR_WITH

The link `OR_WITH` splits a dissection into more than one in a slightly more complex way than the previous example. Each resulting dissection retains the original `MAIN` branch; one is just the `MAIN` branch, whilst the others are formed from the `MAIN` branch, and one `OR_WITH` branch changed to `WITH`. For example:

```
<dissection header>
MAIN x
OR_WITH y
OR_WITH z
```

is transformed to:

```
<dissection header>
MAIN x
<dissection header>
MAIN x
WITH y
<dissection header>
MAIN x
```

```
WITH z
```

6.4 Boolean operators

6.4.1 Conjunction

Conjunction within the structure of a dissection uses the `WITH` link, which is mapped as normal using the Link Manager. However, conjunctions between descriptors are treated specially. As described in [VUM03], conjunctions between descriptors are only permitted between categories of type anatomy; they are expanded as follows:

```
descriptor1 & descriptor2
```

is transformed into:

ArbitraryBodyConstruct whichG <

hasStructuralComponent grail1

hasStructuralComponent grail2 >

where **grail1** is the GRAIL mapping for `descriptor1`, and **grail2** is the GRAIL mapping for `descriptor2`.

6.4.2 Disjunction

Disjunction within the structure of a dissection is treated by expanding a dissection into a set, as described in section 6.3.2; disjunction between descriptors is also treated by expanding the dissection, but in a different manner:

```
<dissection header>
MAIN x
 LINK a / b / c
```

is expanded into:

```
<dissection header>
MAIN x
 LINK a
<dissection header>
MAIN x
 LINK b
<dissection header>
MAIN x
 LINK c
```

6.5 Translating a single dissection to a single GRAIL concept

The previous section has indicated how a dissection may be expanded into a set of dissections, if particular special links, or disjunctions within descriptors, are used. This section describes how each dissection in that set is then individually translated into a GRAIL expression representing a single complex GRAIL concept.

The overall process is that the dissection body is translated into a GRAIL expression (described in sections 6.5.1 and 6.5.2), which is then annotated with information from the dissection header (described in section 6.5.3).

6.5.1 Start point

The starting point for each dissection is the keyword `MAIN`, followed by a descriptor that is always of category 'Deed'. The equivalent GRAIL starting point is a more complex "wrapper", enabling the uniform representation of procedures that have any number of sub-processes that may or may not actually take place. This is represented in two parts: a concept to act as a starting point, and the treatment of `MAIN` as a link in its own right.

6.5.1.1 GRAIL start point

The GRAIL start point is a concept; it is specified to Tigger using the option **GRAIL→Start point...** from the Tigger Main Window, illustrated in Figure 29:

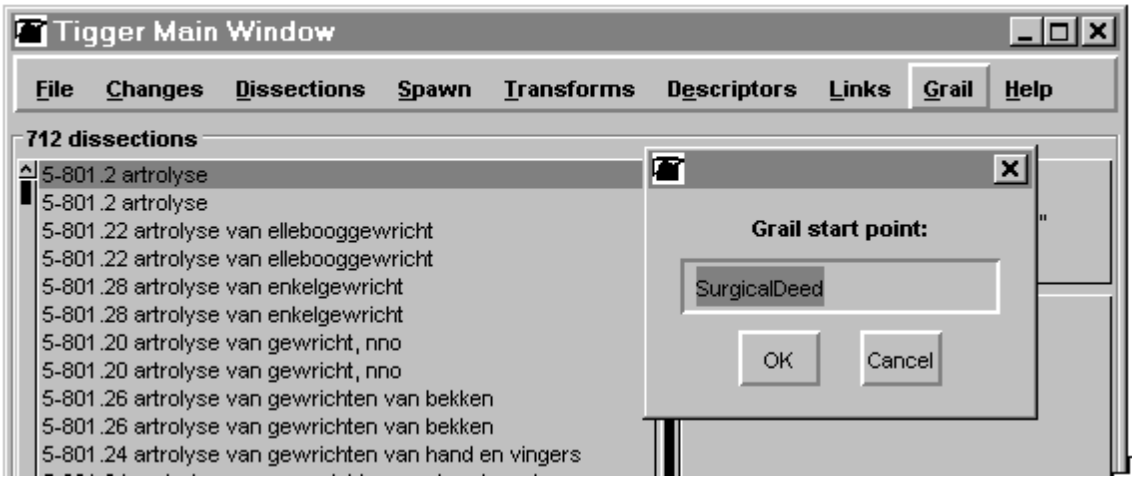


Figure 29 : Specifying the GRAIL start point

Alternatively, the start point can be included (with the descriptor mapping rules) in common.txt if the **include translation rules** check box is used (as described in Section 2.6.6).

6.5.1.2 MAIN treated as a link

MAIN is treated as a link: linking the invisible descriptor that is mapped to the GRAIL start point to the first descriptor that appears in a dissection. This is so it can be mapped to GRAIL using the Tigger Link Manager in the normal way, for example as shown in Figure 30.

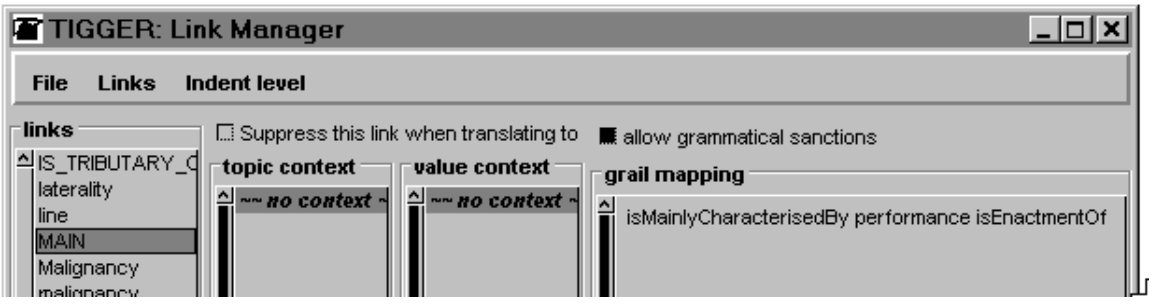


Figure 30 : Mapping MAIN, treated as a link

Given this mapping, together with the GRAIL start point, every dissection translation starts as:

((SurgicalDeed which

isMainlyCharacterisedBy (performance whichG

isEnactmentOf (

Note that the attribute chain **isCharacterisedBy performance isEnactmentOf** is expanded into a nested complex concept: the operator **whichG** rather than **which** is used (representing a complex concept sanctioned at the grammatical-level, rather than the sensible-level) to link the second and third parts of the attribute chain. This is because using the grammatical qualifier level is the more general case, and we have not implemented the mechanism to specify, for each element of an attribute chain, what the qualifier level "ought" to be. In practice, this limitation does not cause any problems.

6.5.2 Translating the dissection body

The rest of the dissection graph structure is then recursively translated into GRAIL constructs; as an example, consider the dissection fragment:

```
descriptor1
      LINK descriptor2
```

First, the GRAIL concepts representing the descriptors must be found; often this is a straightforward one-to-one mapping, but where contextual mappings (described in section 3.2.1) are associated with a descriptor, the surrounding link context must be used: in this example if `descriptor2` has a contextual mapping for the case where it is used after `LINK`, that mapping must be used, otherwise the standard mapping is used. Note that the criterion **playsClinicalRole SurgicalRole** is applied to any descriptor mappings that have as a parent either of the concepts **GenericProcess** or **ClinicalAct**.

Once GRAIL concepts (say, $\mathbf{grail_1}$ and $\mathbf{grail_2}$) are found in this way, an appropriate attribute or attribute chain must be found. First, the topic context ($\mathbf{grail_1}$) is checked in the mappings for `LINK`; from the resulting set of value contexts, the appropriate context is found for the value ($\mathbf{grail_2}$), which results in an appropriate collection of potential attribute or attribute chains. A "matching context" is indicated if the GRAIL concept which has been found as a translation for a descriptor is a kind of (in the strict GRAIL sense) the GRAIL concept used as the key to a contextual link mapping.

To disambiguate from the set of attribute or attribute chains, each one is checked in turn to see whether a suitable sanction exists in the CRM at the sensible qualifier level: the first one that matches is used. If none match, then the test is relaxed to use the grammatical qualifier level; again, the first that matches is used; if none, then an error is raised, indicating a problem with the dissection, missing link mappings, or missing sanctions in the CRM[35]. Section 7 discusses how information gleaned from this process can be used to augment or validate the CRM.

The definition of "suitable sanction" is as follows: in the case where the potential link mapping is a single attribute $\mathbf{a_1}$, a single sanctioning test is performed: $\mathbf{grail_1\ a_1\ grail_2}$. For an attribute chain $\mathbf{a\ c_1\ a_1\ c_2 \ldots c_{n-1}\ a_{n-1}\ c_n\ a_n}$, two tests must be performed, both of which must pass: $\mathbf{grail_1\ a\ c_1}$ and $\mathbf{c_n\ a_n\ grail_2}$.

[35] as described in Section 4.4.2, it is now selectable (within Tigger) whether or not to try grammatical sanctions if no sensible sanction exists

6.5.3 Headers transformed into extrinsic annotations

The code (if present) and the rubric of the original dissection is combined into a single string and attached to the generated GRAIL concept using an extrinsic statement with the attribute **hasDissectionRubric**.

Some dissections originate from terms in existing coding systems, and have both a SOURCE and a CODE header field; in this case, Tigger also adds a **hasProjection** operation to the GRAIL translation to add the mapping from the GRAIL concept back to the original code. This can be seen in the example of Figure 8 on page 86.

6.6 Outstanding issues

The mechanism described so far to translate dissections to GRAIL has proved to be sufficient to translate the overwhelming majority of the dissections we have seen (as of January 1998). There are, however, several outstanding issues that are known about but not yet implemented; they are introduced here.

6.6.1 Non-analytic links

[VUM03] describes a mechanism for annotating links with the NON_ANALYTICALLY keyword, to indicate that any links so annotated represent "empirical" rather than "analytical" information: i.e., information that comes from the modeller's clinical knowledge of the situation, rather than from the strict analysis of the input term. The expected mechanism to translate such links into GRAIL is not to use them as part of the resulting GRAIL concept, but to attach such information using **topicNecessary** statements.

However, of the set of dissections received as of January 1998, less than 0.1% used this mechanism, and so the translation of non-analytic links has not been implemented: we require substantial examples of use to determine the details of how to translate such constructions.

6.6.2 OTHER

A relatively recent addition to the dissection specification is the keyword OTHER. This reflects common usage in existing coding schemes, but needs to be handled differently when transforming dissections into the more formal GRAIL notation. OTHER is used to indicate that the descriptor which it describes is treated in the original source rubric as being exclusive of a set of other codes that happen to appear in the original scheme. An interim implementation in the dissection translation is described here.

In the translation, OTHER is treated by further specialising the GRAIL mapping of the descriptor by a scheme-specific and application-specific criterion; for example the descriptor:

```
OTHER bone
```

is translated into the GRAIL fragment:

```
(Bone which hasDissectionCodingOtherness
        (CodingOtherness which hasContextOfSchemeNamed 'WCC'))
```

It is thus mandatory that any dissection that uses OTHER contains a SOURCE header field[36].

[36] an extension to this which is planned but not yet implemented is to further specialise the GRAIL mapping with the relevant code as well as scheme, so that the result would be:

```
(Bone which hasDissectionCodingOtherness
        (CodingOtherness which < hasContextOfSchemeNamed 'WCC'
                hasContextOfCodeNamed '5-780.19' >))
```

6.6.3 Suppressed links

As described in section 6.2, any part of the dissection structure under suppressed link is not included in the dissection structure that is translated to GRAIL; it should, however, be added as additional extrinsic information to the resulting GRAIL concept(s).

6.7 User Interface

This section describes how to access Tigger's dissection translation functionality: it can be invoked in two ways:

- on a dissection-by-dissection basis to examine how individual dissections are translated; this is described in section 6.7.1;
- on a collection of dissections: use the menu option **Dissections→Check up to and including→ Grail translation** from the Tigger Main Window; this option attempts to produce a GRAIL translation for all the visible dissections that do not have any sort of problem (such as using un-mapped links or descriptors). The translations are associated with each dissection within Tigger; subsequently, they can be viewed interactively, compiled, or exported as GRAIL source files.

However the translations are generated, they can be exported in textual GRAIL source form by using the menu option **GRAIL → Export GRAIL Translations...** from the Tigger Main Window. Alternatively, they may be compiled, as described in section 6.7.2.

6.7.1 Interactive dissection translation

Individual GRAIL representations for dissections are viewed within a GRAIL Output tool; there are two ways to open such a tool. The first is to use the menu option **GRAIL→Show GRAIL Translation** from the Tigger Main Window (or a spawned dissection list view); this action opens a GRAIL Output tool which is then automatically re-focused to whichever dissection is selected in the dissection list. The second is to use the GRAIL button at the bottom of a Dissection Editor (see section 2.6.3). Whichever method is used, the same tool is opened; it is illustrated in Figure 31.

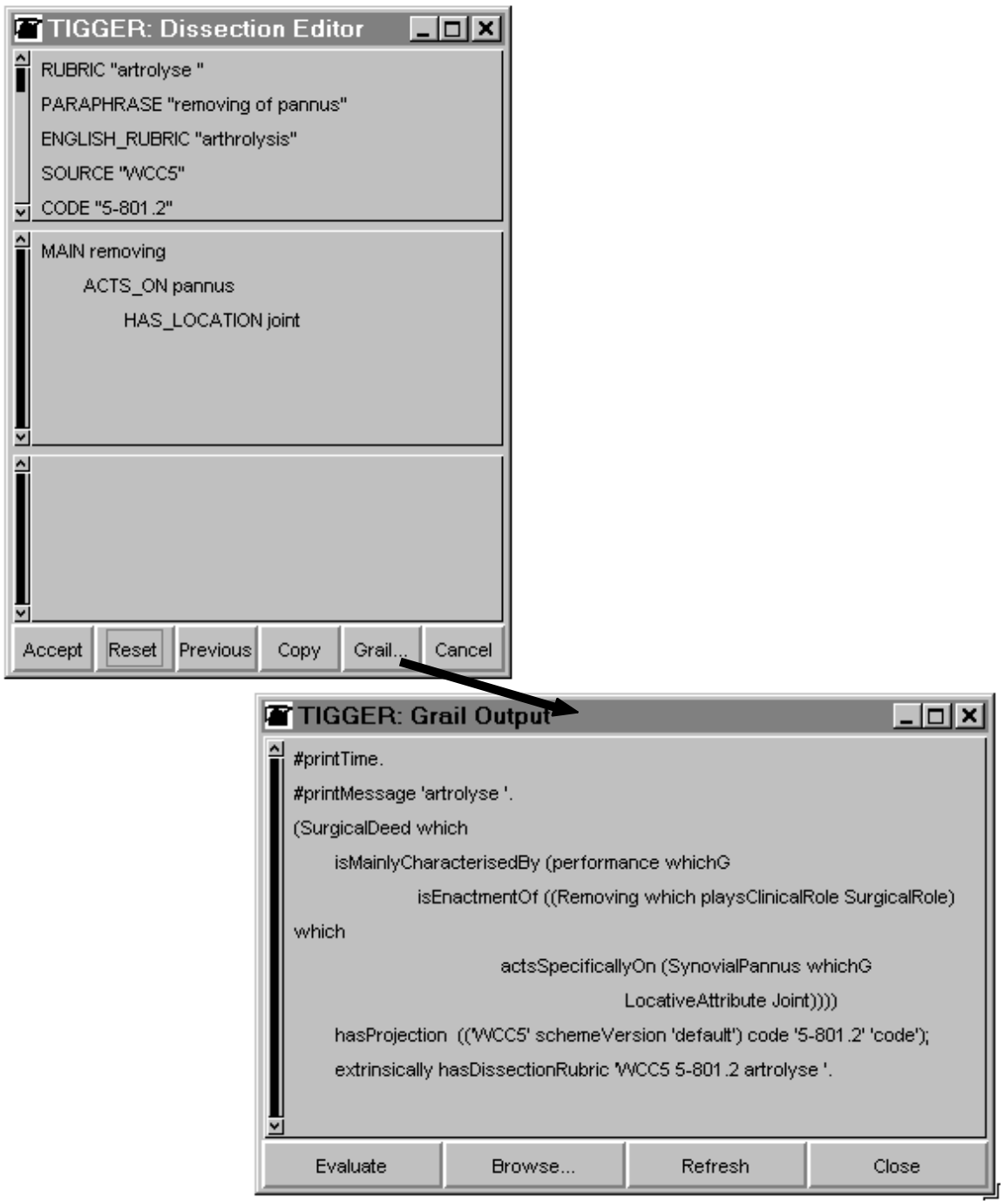


Figure 31 : GRAIL Output window showing GRAIL translation

The main part of the tool is a KnoME Workspace: offering browsing and exploration facilities for part or all of the GRAIL expression. Some commonly used functions are available via the buttons at the bottom of the tool.

Tigger is an interactive environment: links mappings or descriptor mappings may change, or dissections may be amended; any of these actions will invalidate any translations that may have been made. Tigger ensures that the translations associated with any dissection are reset if any operation that affects it takes place within the environment. If a GRAIL output window is open, the translation will be automatically re-generated and shown in the window; otherwise, the translation is removed.

6.7.2 Dissection compilation

Dissections that have been translated to GRAIL may then be compiled: either individually via the GRAIL Output tool's menu options or buttons, or in bulk by using the menu option **Dissections→Check up to and including→Compilation of Grail expressions** from the Tigger Main Window. Any dissections with compilation errors can then be spawned off and examined: most compilation errors arise from cardinality conflicts which Tigger does not attempt to forestall in the dissection translation process (this is discussed further in section 8.2).

Once compiled, the KnoME Concept Browser can be used to explore the resulting classification. The display of GRAIL concepts can be set to use the originating dissection's rubrics and codes by using the menu **option language→extrinsics** from the context button menu over the hierarchy view of the Concept Browser, as shown in Figure 32:

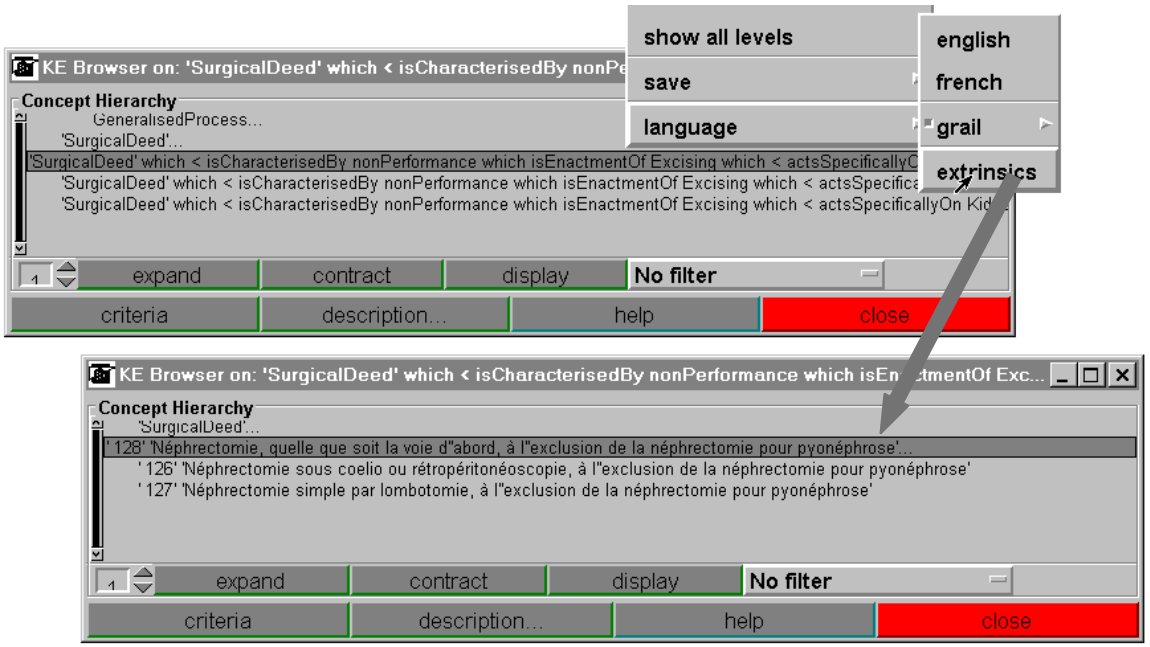


Figure 32 : CRM classification of dissections displayed in a Concept Browser

6.8 Abstract GRAIL concepts

So far, this section has described the process of translating individual dissections into one or more GRAIL concepts, and has illustrates how to compile these expressions and view the resulting classifications. However, it is often the case that just producing GRAIL translations for the dissections supplied is not enough to produce the expected levels in the resulting hierarchies — as GRAIL is generative, it does not need to reify all potential concepts that may subsume the translated dissection. Normally, this is beneficial, but for the point of view of viewing or exporting hierarchies, it can be counter-productive. So, in order to support this specific need, Tigger can produce a selection of abstract GRAIL concepts that subsume GRAIL expressions that derive from the dissections themselves. The menu option **GRAIL→Export GRAIL abstractions...** produces a GRAIL source file that can be manually added to the CRM using the KnoME compiler.

There are several ways of producing these GRAIL abstractions:

- pruning the structure of the generated GRAIL expression;
- substituting more abstract GRAIL concepts for parts of the GRAIL expression representing specific descriptors;

- substituting more abstract GRAIL attributes for parts of the GRAIL expression representing specific links;
- some combination of these.

At present, Tigger produces abstractions based on the first mechanism; it is anticipated that an important part of the ClaM workbench will be facilities for enabling the user to control the style and area in which abstractions are required.

7 CRM Extension & Validation

7.1 Introduction

So far, this document has described how to transform dissections into GRAIL expressions; these expressions contain elements of the CRM, and conform to its style. Using the intermediate representation of the dissections, many clinicians are able to map elements of existing (or new) coding and classification schemes into the CRM, without having to know much of its internal detail and complexity. This process supports the classification management functionality which GALEN-IN-USE is developing, and is the primary purpose for the techniques described in this document.

However, the act of representing existing coding and classification schemes in an intermediate representation is also beneficial for the development and maintenance of the CRM itself; this section describes the ways that this may take place, and introduces some tools that are being developed to fully exploit the current activity within the project.

There are two significantly different ways in which the dissection analysis benefits the development of the CRM: as a way of specifying the direction in which the CRM may usefully be extended in terms of its coverage of clinical concepts, and as a way of validating the sanctioning information contained within the CRM.

7.1.1 Specification of requirements for extending the CRM

The CRM does not claim to cover all of medicine, and may well never cover all of medicine to the level of detail that may be required by certain applications. The development of the CRM has always been guided by its users; at present the dissection analyses being performed in GALEN-IN-USE provide specific information about which parts of medicine are required: the set of descriptors that are used in dissections but which are not easily mapped provide a rich corpus directing the extension of the CRM in terms of additional concepts. Section 3.2 provides details of support that Tigger offers to extend the CRM, so as to be able to model descriptors that appear in dissections.

7.1.2 Validation and extension of sanctioning knowledge within the CRM

Tigger relies on the sanctioning information in the model to disambiguate the mapping of links into GRAIL attributes or attribute chains, as described in section 6.2. Normally, Tigger expects an appropriate sanction to exist at the sensible qualifier level to enable an attribute or attribute chain to link two descriptors in a dissection. It may be the case that no sanction is found, or that only a grammatical-level one is found. At present, Tigger produces translations based on either sensible-level or grammatical-level sanctions: this is a pragmatic choice based on the knowledge that:

- it is acknowledged that some sensible-level sanctioning information is not yet present in the CRM;

- the organisation of the resulting GRAIL classification does not depend on the qualifier-level of any of the criteria within the expression.

If Tigger fails to find a sensible-level sanction, this indicates:

- either a dissection that uses a link in an incorrect way (assuming the CRM is complete and correct);
- or some missing sanctioning information in the CRM (assuming the dissections are correct).

Tigger provides facilities for using information about the sanctions it uses (or fails to find) in the translation process to extend and validate the CRM sanctioning knowledge, and to further validate the dissections. Two tools are described in this section: the *Sanction Manager*, which provides facilities for exploring and examining the sanctions from the CRM that are used when translating dissections, and the *Sanction Promoter*, which is designed to make it easy to produce a minimal set of additional sanctions when extending the CRM.

Note that these tools are at an earlier stage of development and iterative prototyping than the rest of the Tigger environment.

7.2 Sanction Manager

The Tigger Sanction Manager (illustrated in Figure 33) is a tool for exploring the different connections between GRAIL concepts that have been used during the dissection translation process to map links to attributes. It is opened by using the Tigger Main Window option **GRAIL→ Open Sanction Manager**.

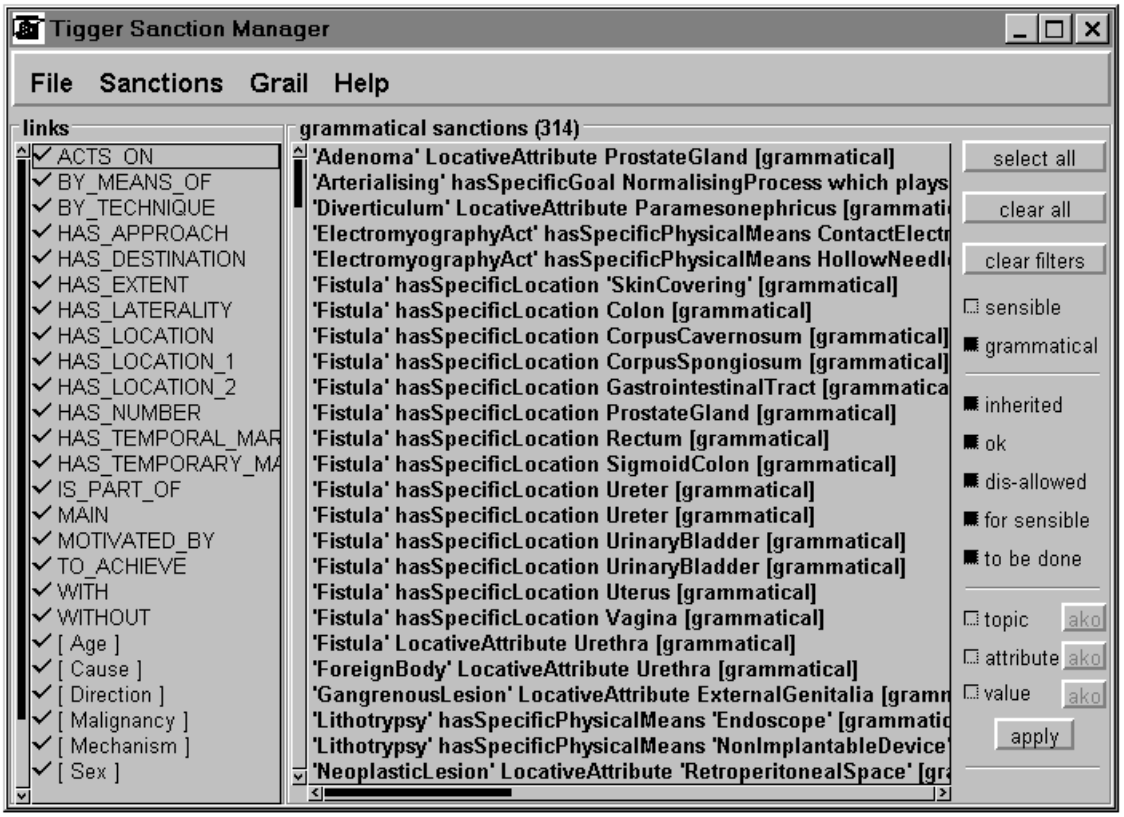


Figure 33 : The Tigger Sanction Manager

Each "connection" is a 4-tuple, consisting of:

- topic: GRAIL translation of `descriptor1` (given the dissection fragment

```
descriptor1
      LINK1 descriptor2);
```

- attribute: GRAIL attribute that is the translation for `LINK1;`
- value: GRAIL translation of `descriptor2`;
- qualifier: the qualifier level of the lowest GRAIL sanction that exists in the model which sanctions the GRAIL criterion **attribute - value** for the **topic**.

Each element of the connection can be viewed or displayed (using the GRAIL KnoME Concept Browser or GRAIL Display tool) using options from the context mouse button menu over the connection list.

There are two steps to using the Sanction Manager: organising and filtering the set of connections that exist, and marking specific selections of connections as valid or invalid. This process may result in a set of additional invalid dissections, and/or a set of additional sanctions that may be found to be missing from the CRM. The Sanction Promoter is a separate tool for organising, minimising and adding these sanctions to the CRM; it is described in section 7.3.

7.2.1 Organising and filtering the set of connections

When the Sanction Manager is opened, it displays a list of all the connections that have been found in the dissections that have been translated; to reset the list of connections, use the menu option **File→ Forget→ confirm**. Together with the ability to produce GRAIL translations of individual dissections (as described in section 6.7.1), this enables the examination of the connections that are used for individual dissections.

It may be the case, however, that there are several hundred or thousand connections that may be visible in the connection list; the Sanction Manager provides various facilities for filtering and for sorting the connection list.

7.2.1.1 Filtering the connection list

The connection list can be filtered in several different ways:

- by link: each connection exists due to the translation of a link within a dissection; the connection list shows only those connections which originate from links which are selected in the link list on the left-hand side of the tool;
- by qualifier-level: the **sensible** and **grammatical** check-boxes are used to show connections of a particular qualifier level. This reflects the different tasks that may be involved: for sensible-level connections, the task is to validate that the sanctions in the CRM that gave rise to such connections are appropriate, whilst for grammatical-level connections, the task is to either disallow such connections, or suggest that they should be allowed at the sensible-level;
- by status: section 7.2.2 describes the different steps in marking connections that are supported: the status-control check-boxes to the right of the Sanction Manager are used to display only those connections of a particular status;
- by inheritance status: the menu switch **Sanctions → Use inheritance** can be used to control whether inheritance is taken into account when displaying sanctions, and when marking sanctions with particular statuses. If inheritance is used, then the **inheritance** check-box specifies that that the connection list should only contain those sanctions which are not subsumed by any other connection in the list;
- by GRAIL type: the connection list may be filtered by the GRAIL type of any or all of the topic, attribute or value in a particular connection. For example, to show only those connections whose topic is a kind of **Fistula**, check the **topic** check-box which enables the associated **AKO** button: pressing this button allow the specification of the GRAIL type to be entered. Then **apply** the filter; this is shown in Figure 34:

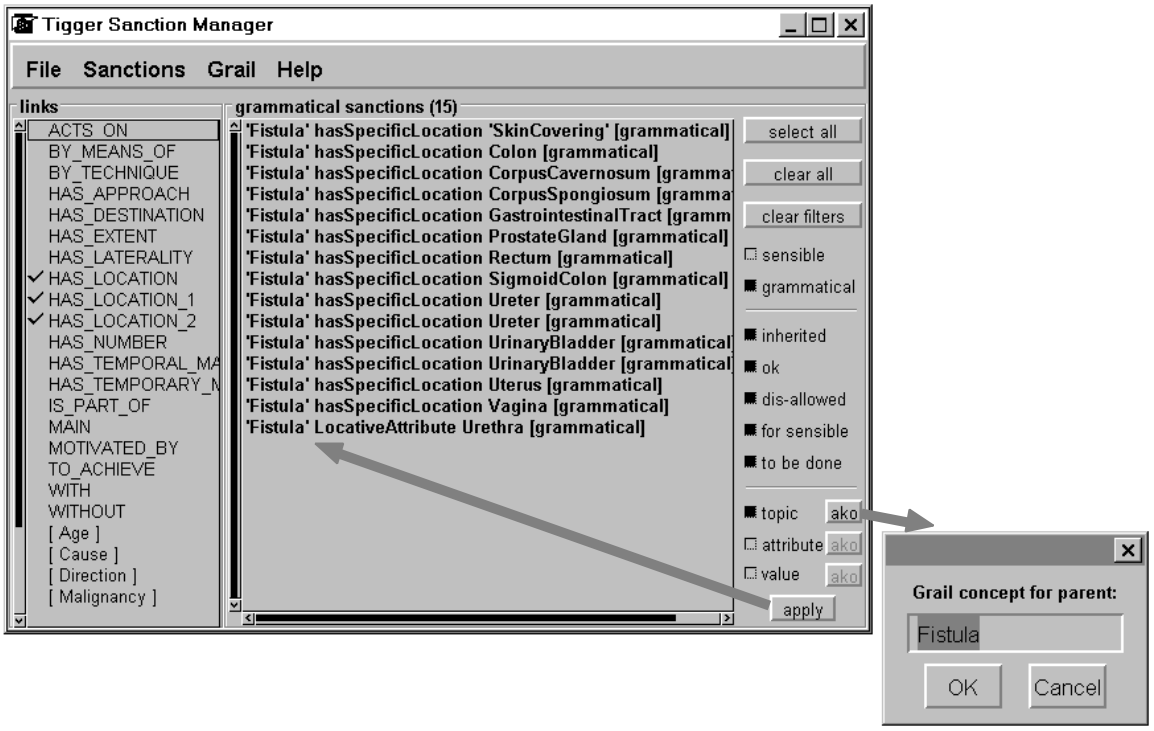


Figure 34 : Setting the topic filter in the Sanction Manager

Note that the link list now shows only those links that were the source of the connections shown; to see the particular dissections that were the source of a particular connection, use the context mouse button menu option **used in...**.

7.2.1.2 Sorting the connection list

The connection list can be sorted according to the components (topic, attribute or value) of any connection using the menu option **Sanctions → Sort by...**.

7.2.2 Marking connections

The facilities described so far provide a view of the connections used in the dissection translation process; subsequently, the Sanction Manager permits an individual connection, or a set of connections, to be marked in various ways. Options to perform this are available from the mouse context button menu **mark**, as shown in Figure 35:

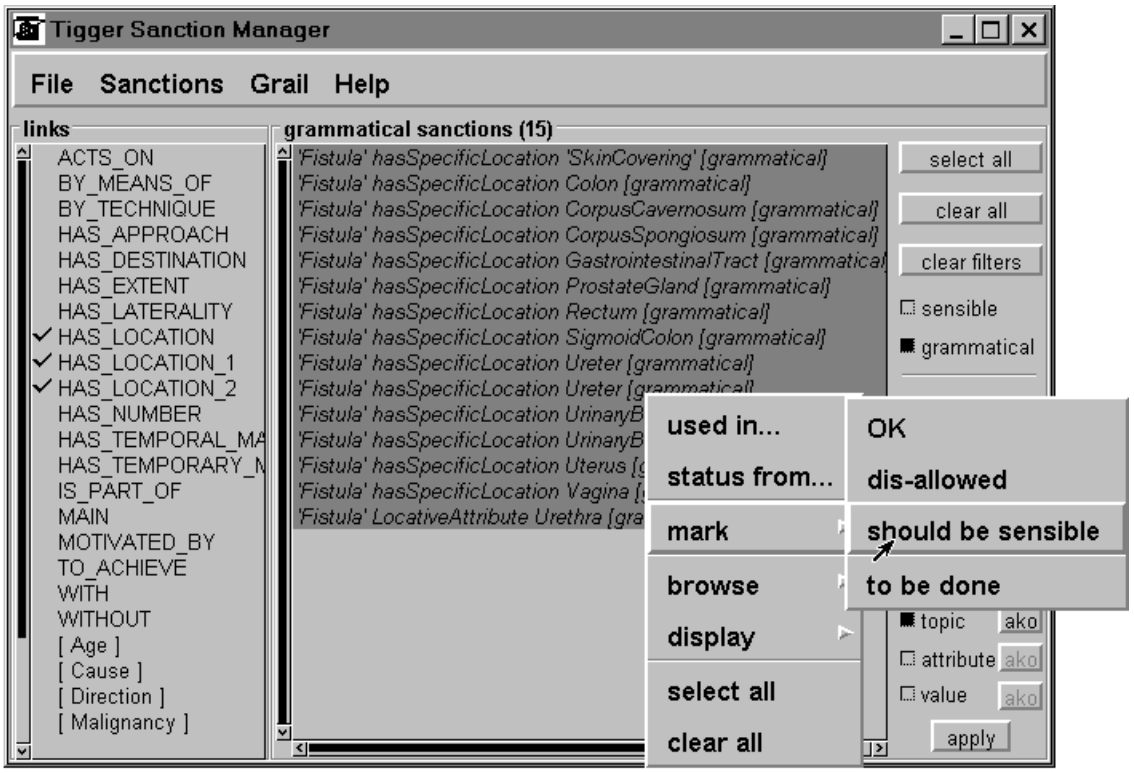


Figure 35 : Marking statuses within the Sanction Manager

Connections within the connection list are displayed using different highlighting conventions according to their status; the **Help** menu shows details of the convention in use. As described in the previous section, the connection list can also be filtered according to status using the check-boxes to the right of the tool. The different available statuses are:

- **OK**: used for sensible-level connections, indicating that this is a valid connection, i.e. that the sensible-level sanction in the CRM is validated.
- **dis-allowed**: used for either grammatical-level or sensible-level connections, indicating that the connection should not be made. If used on a sensible-level connection, this might indicate that a sensible-level sanction is incorrect; if used on a grammatical-level connection, it might indicate that the dissection that gave rise to this connection is incorrect. To propagate any dis-allowed connections back into the appropriate dissection translations, use the menu option **Sanctions → Propagate…**.

  It is possible to save the set of connections marked as **dis-allowed**, to act as an input to the dissection translation process, so that the next time such grammatical-level connections are found, they are immediately skipped; this would allow immediate further checking of alternative link mappings, and may provide earlier feedback of problems in the dissections. These options are available from the menu options **File → Import…/Export…**. The syntax for the interchange format is given in Appendix Three; it is used in two ways: to pass around sets of "raw" connections (an output from the dissection translation process), and to specify sets of dis-allowed grammatical-level connections as an input into the translation process. Notionally, if the CRM was known to be "complete", all grammatical-level sanctions might be dis-allowed, but that state may well never exist.
- **should be sensible**: used for grammatical-level sanctions to indicate that the CRM is missing a sensible-level sanction that would be expected. The set of suggested sanctions can be exported in GRAIL source form using the menu option **GRAIL → Export new sensible sanctions…**. However, it may well be the case that the sanctions ought to be added into the CRM at a higher level, both to minimise the number of additional sanctions, and to make

sure that their coverage is maximal. Managing the set of suggested CRM sanctions in this way is the task of the Sanction Promoter (described in the next section);

- **to be done**: used to remove any of the statuses so far described.

7.3 Sanction Promoter

One outcome from the use of the Sanction Manager is a set of connections which ought to be sanctioned at the sensible level by the CRM, but which are not. The Sanction Manager can export this set in a textual GRAIL source form, suitable for immediate inclusion in the CRM, but simply adding these sanctions has a drawback. Adding sanctions in this way will cover the cases already met, but there may well be more general cases that are not covered; the model may come to be populated with a multitude of sensible-level sanctions, each of which covers just one particular case.

To overcome these problems, Tigger provides an interactive tool for exploring and managing any potential sanctions: to maximise the coverage of the sanctions, and to minimise their number. The tool is known as the Sanction Promoter. It is opened from the Sanction Manager using the menu option **GRAIL → Open Sanction Promoter**, and will open with the set of connections that have been marked as **should be sensible**; it is illustrated in Figure 36:

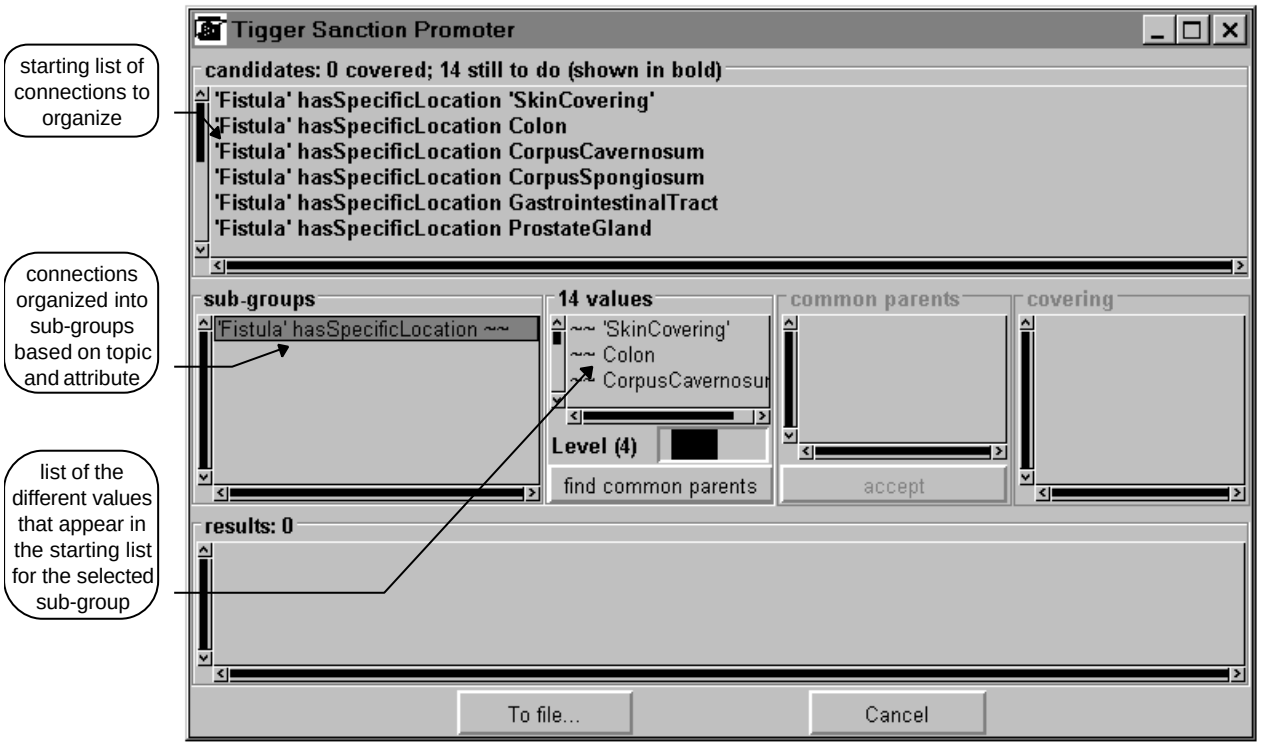


Figure 36 : The Sanction Promoter

The Sanction Promoter first organises the set of connections into "sub-groups": groups of connections with the same topic and attribute. This definition is limited, and may well be extended in the future. In the example shown, there is a single sub-group. The tool shows this sub-group, together with a list of values that form the differences between the input set of connections. The task is to explore the model, and possibly to raise the level of any sanctions added, so that fewer sanctions are added, but they have more coverage.

To explore the common parents of the values shown, the button **find common parents** searches the model and presents any candidates in the next pane. The extent of the search is controlled by the slider above the button, which controls the number of subsumption (parent-child) links that are traversed in the search. This is shown in Figure 37:

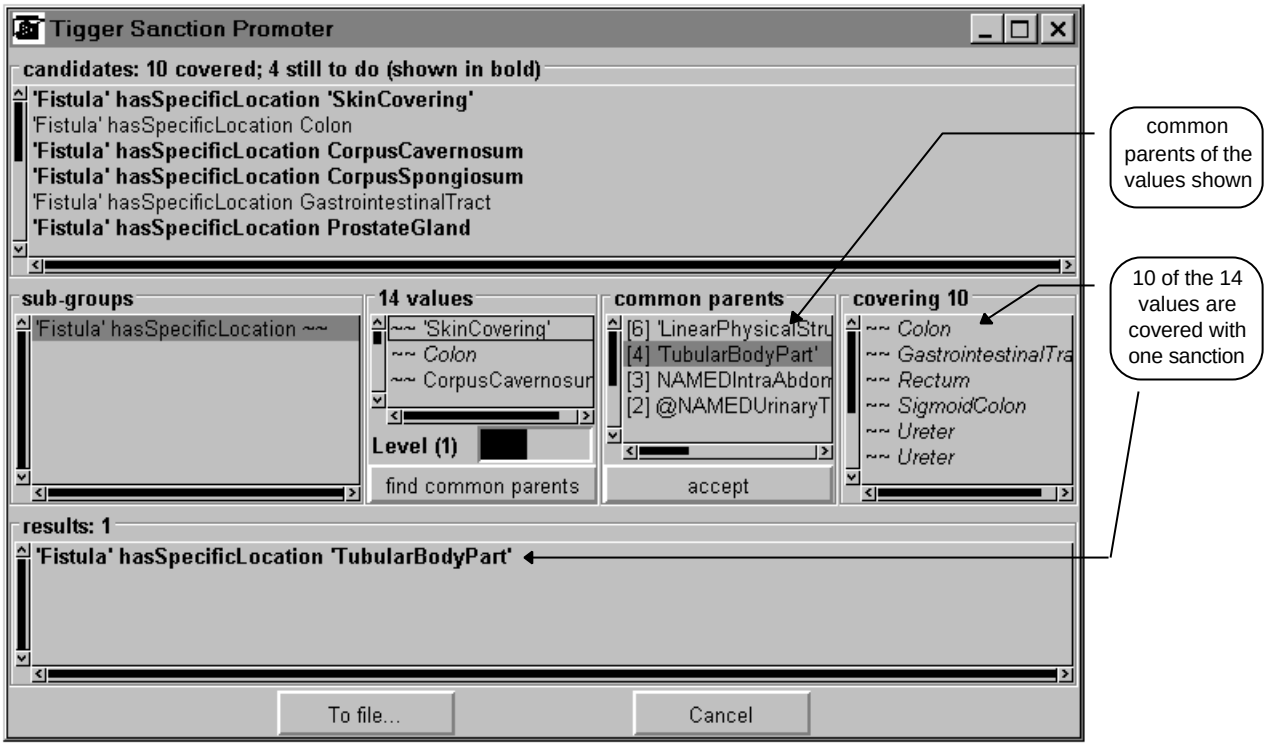


Figure 37 : Finding common parents using the Sanction Promoter

Several candidate common parents are found; using knowledge about the CRM, and exploring it directly (any concept appearing at any part of the tool has an associated browse and display option), a choice may be made that the concept **TubularBodyPart** is an appropriate point at which to apply a sensible sanction. The **accept** button adds this new sanction, which appears in the result list at the bottom of the tool; the candidate list at the top reflects those input connections that are now covered with the new sanction.

The new sanction will have wider effect, however. To check what other concepts are affected by the new sanction, it is necessary to explore the children of the value of the new sanction. This can be done directly, as shown in Figure 38:

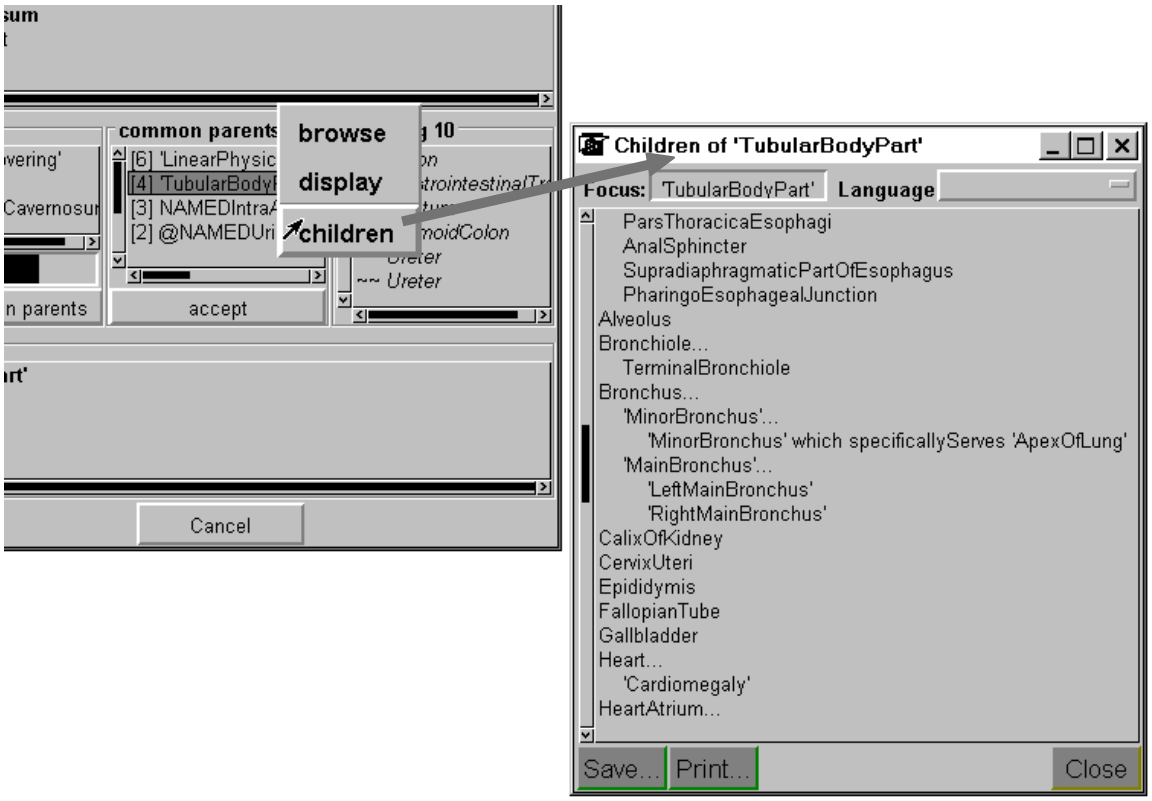


Figure 38 : Exploring children of concepts from the Sanction Promoter

If satisfactory, the set of sanctions in the result list can be exported in GRAIL source form using the button **To file...**. Whilst possible to add the new sanctions directly into the CRM, this is not done so as to ensure that the source form of these sanctions is maintained.

8 Discussion

So far, this document has concentrated on describing Tigger as it has been developed to support the activities taking place in GALEN-IN-USE. This section discusses some wider issues:

- internal to Tigger: for example some aspects of the translation mechanism;
- in a wider view: how it relates to other activities taking place elsewhere.

8.1 Link mappings as a formalization of "modelling style"?

Link mappings — including contextual information — may be thought of as being a first step towards the task of formalizing the CRM modelling style. There are several different kinds of knowledge and style in which GRAIL modellers and maintainers need to become proficient. The first is the syntax and the operation of the GRAIL language: this can be described and documented. As GRAIL is designed for implementation by computer, tools and authoring environments can be tuned to support that language. However, different modellers or groups of modellers, given the same domain to model for a particular purpose, using a particular modelling language, will come to build different models in different styles. Frequently, there is a choice to be made as, for example, whether to model concepts compositionally, or to introduce them as primitives, and then describe them further using necessary statements. It has been difficult to lay down these choices in a form that can be used to build tools to support a particular modelling style, so as to guide modellers adding, in a consistent way, to an existing model.

An implication is that the knowledge contained in the link mappings (particularly the information pertaining to attribute chains) may be useful for other tasks, notably the task of translating complex GRAIL expressions into natural language. Much of the task of generating natural language is to do with simplifying the structures of GRAIL concepts into a form that is more easily readable and understandable in language: a task that is, in many ways, the inverse process from translating dissections into GRAIL, which is supported by the link mappings.

8.1.1 Translation of automatically-generated dissections

The majority of the dissections that have so far been analysed and translated using Tigger have been produced manually by clinicians: data entry tools may have been used to enter the analysis, but the clinical task of interpreting the rubric into its structure has been done by hand.

Another approach being investigated in GALEN-IN-USE is to adapt the automatic parsing tools developed in multiTALE {ref} to rubrics from coding and classification schemes, to produce dissections that can be translated using Tigger. This process is being developed, but it already seems that having a stable intermediate format provides a much easier target to attain than the underlying GRAIL model.

Experience with this process has emphasised the importance of dissection transformations (as described in Section 2.6.7.1). This mechanism helps bridge the gap between the *linguistic-oriented* structures that automatic parsing tools are set up to generate, and the *concept-oriented*

structures that the intermediate representation, and its translation into GRAIL, is designed to support.

8.1.2 Why not just use sanctioning information?

The question arises: why bother with explicit link mappings? Why not just use the sanctioning information already in the model to find out how different concepts may be linked? There are different reasons for this:

- the sanctioning information in the CRM is not complete; as described in section 6.8, an important goal of the translation process is to extend and validate the sanctioning information within the CRM;
- even if the sanctioning information were complete, there are often many different ways to connect two concepts using sensible sanctions, especially when attribute chains are permitted; this operation is also relatively expensive.

So, the link mappings provide a series of hints and suggestions for Tigger to guide its search: it uses sanctioning information, but not alone; to make the process tractable, the link mappings act as an additional resource.

8.2 How much should Tigger do?

In the process of translating dissections into GRAIL expressions, Tigger queries the CRM to help it disambiguate, for example, between different link mappings. This takes time, but enables better translations to be produced. In contrast, Tigger only treats each link within a dissection in isolation, so may not detect some conflicts in cardinality until this is pointed out by errors from the classification process: this takes longer, but does not necessitate extra functionality to be implemented within Tigger. An important choice, therefore, in designing and implementing Tigger has been in deciding where to draw the line between implementing functionality within Tigger to provide early feedback, and relying on the underlying model which may take longer. An extreme version of this is to replicate large parts of the classification process (or even all of it) within Tigger, which is clearly a VBII (very bad idea indeed).

In practice, it appears to be the case that of the dissections seen so far, less than 5% of GRAIL translations which are produced by Tigger lead to subsequent classification errors.

8.3 Multilingual aspects

Tigger is monolingual; descriptors and links have English labels, and the processes and tools for mapping use the underlying assumption that the dissections are produced in the same language as that used by the modellers producing the CRM. This is not to say that dissections and descriptors cannot be usefully be managed in different languages: indeed this is mandatory. As the GALEN-IN-USE intermediate format becomes more widespread, it becomes more important that we can support such multilingual aspects. Rather, the task of providing such support is layered on top of Tigger: in practice by the tools provided within the Classification Workbench (for example the SPET) which is delivered to end users across Europe.

8.4 Other intermediate representations

The use of an intermediate structured representation in medicine is not unique to GALEN-IN-USE; CEN has developed the ENV 1828 specification {ref} and the NHS CCC has used intermediate representations as part of the process of using requirements from clinicians to direct and populate a coding system. However, the aspect that remains different to GALEN-IN-USE is the existence of a route from the intermediate representation to an automatically

generated and maintained classification. Appendix Four describes some experiments to merge the results of the work at the NHS CCC with the GALEN-IN-USE Intermediate Representation.

8.5 Domain area

Tigger, and associated dissection-authoring software such as the Classification Workbench, were first developed in the context of the priorities of the GALEN-IN-USE project itself: that of representing surgical procedures.

However, analogously to the Common Reference Model's application independence, there is nothing specific in the technology that limits its use to this domain. These are the tasks required (all supported by Tigger) to move to a different domain:

- create and map a new set of descriptors and links, many of which, especially the links, will be able to be re-used (as described in Sections 3.3 & 4.4);
- modify any descriptor mapping rules (as described in Section 3.3.5);
- modify the GRAIL start point (as described in Section 6.5.1.1);
- modify the set of descriptor category sanctions (as described in Section 3.1.1.1).

All these parameters are exported as part of the common.txt file so that the Classification Workbench dissection-authoring software can be used directly in this new domain.

The only outstanding areas where Tigger is not soft are:

- the treatment well-known 'special' links (as described in Section 6.3);
- the treatment of descriptor conjunctions (as described in Section 6.4.1).

8.6 Conclusion

Tigger, when considered overall, is complex: it supports different kinds of functionality to support several different tasks. The task, for example, of refining link and descriptor mappings is not trivial, and demands knowledge of the ways in which Tigger does its job, as well as of the CRM. However, it is important to remember that the major task for which Tigger was developed is to simplify the analysis of clinical information by many different clinicians in different places. For their task to be eased, some complexity has to be concentrated into Tigger.

Appendix One: Dissection Interchange Format

This appendix presents a BNF-style grammar for the dissection interchange format that is read by Tigger. Note that the grammar does not specify rules as to the level of indenting, or any constraints between kinds of links or descriptors that may be used.

```
DissectionList          :   DissectionList Dissection
                            |   "nothing"
                        ;
Dissection              :   DissectionHeader
                            OptionalDissectionProperties
                            DissectionBody;
DissectionHeader        :   'RUBRIC' RubricString
                        |   <string>;
RubricString            :   <string>;
OptionalDissectionProperties :  OptionalDissectionProperties
OptionalDissectionProperty
                        |   "nothing";
OptionalDissectionProperty :    'ENGLISH_RUBRIC' OptionalPropertyString
                        |   'PARAPHRASE' OptionalPropertyString
                        |   'SOURCE' OptionalPropertyString
                        |   'AUTHOR' OptionalPropertyString
                        |   'STATUS' OptionalPropertyString
                        |   'HISTORY' OptionalPropertyString
                        |   'COMMENT' OptionalPropertyString
                        |   'CODING_METACOMMENT'
OptionalPropertyString
                        |   'CODE' OptionalPropertyString  ;
OptionalPropertyString  :   <string>;
LinkAttribute           :   ExistingLinkWord
                        |   NewLinkDescription;
ExistingLinkWord        :   <upperidentifier>;
NewLinkDescription      :   ExistingLinkWord ':' NewLinkId;
NewLinkId               :   <loweridentifier>;
DescriptorWord      :   DescriptorId
                        |   DescriptorType ':' DescriptorId ;
Descriptor              :   DescriptorWord
                        |   DescriptorWord Boolean Descriptor    ;
Boolean                 :   '&'
```

	|	'/';
DescriptorId	:	<upperorlower>;
DescriptorType	:	<capitalisedorloweridentifier>;
Tabs	:	Tabs Tab
	|	"nothing";
Tab	:	<tab>;
DissectionBody	:	'MAIN' Descriptor Links
	|	'NO_MAIN' Descriptor Links;
Links	:	Links Link
	|	"nothing";
Link	:	Tabs LinkAttribute LinkFlag DescriptorFlag Descriptor
	|	Tabs 'HAS_OTHER_FEATURE' NewLinkId LinkFlag 'VALUE' Descriptor;
LinkFlag	:	'NON-ANALYTICALLY'
	|	'EMPIRICALLY'
	|	"nothing";
DescriptorFlag	:	‘OTHER’
	|	“nothing”;

Appendix Two: GRAIL source to support extrinsic information

The grail source included here is generated by Tigger based on the GRAIL start point (described in section 6.5.1.1). It provides the infrastructure for attaching extrinsic information, and for implementing the OTHER mechanism.

```
"This file was exported from Tigger version 3.7 on January 13, 1998
15:31:54.000"


TopCategory newSub ApplicationCategory.
           ApplicationCategory newSub DissectionCategory.
              DissectionCategory newSub [CodingArtefact].

CodingArtefact newSub [CodingOtherness CodingNECness].

ApplicationAttribute newAttribute DissectionAttribute
InverseDissectionAttribute manyMany.

DissectionAttribute newAttribute hasContextOfSchemeNamed isNameOfSchemeContext
manyMany.
           CodingArtefact grammaticallyAndSensibly hasContextOfSchemeNamed
StringValueType.

DissectionAttribute newAttribute hasDissectionCodingArtefact
isDissectionCodingArtefactOf manyMany.
           SurgicalDeed grammaticallyAndSensibly hasDissectionCodingArtefact
CodingArtefact.

DissectionAttribute newAttribute DissectionDetailsAttribute
InverseDissectionDetailsAttribute manyMany.

DissectionDetailsAttribute newAttribute hasDissectionRubric
isDissectionRubricOf manyMany.
           SurgicalDeed grammaticallyAndSensibly hasDissectionRubric
StringValueType.
```

Appendix Three: Connection Interchange format

This appendix contains details of the interchange format for exchanging sets of connections: to and from the dissection translation process (as described in section 7.2.2). Informally, it is a set of connections, one per line; each connection is a tab-separated list of topic, attribute, value, qualifier-level and (optionally) status. The topic, attribute and value are in the form suitable for evaluation by a GRAIL compiler. A BNF-style grammar is:

Connections	:	Connections Connection
		| "nothing";
Connection	:	Topic Tab Attribute Tab Value Tab QualifierLevel
OptionalStatus;		
Topic	:	<compilableGRAIL>;
Attribute	:	<compilableGRAIL>;
Value	:	<compilableGRAIL>;
QualifierLevel	:	'sensible'
	|	'grammatical';
Tab	:	<tab>;
OptionalStatus	:	Tab Status
	|	"nothing";
Status	:	'dis-allowed'
	|	'should be sensible'
	|	'OK';

Appendix Four: Report describing experiments with the NHS CCC

Comparing Read / Grail hierarchies

Danny Solomon (danny@cs.man.ac.uk)
Jeremy Rogers (jeremy@cs.man.ac.uk)

August 1, 1997

Summary

This document describes some experimental work to investigate the technical relationship between Read codes & templates, and the GALEN model, in an attempt to discover how we might be able to mutually benefit.

Read codes and templates have been automatically transformed into dissections in the GALEN Intermediate Representation, which are then translated into Grail expressions, and compiled into the GALEN Common Reference Model. Elements within the Grail model are mapped to the Read codes from which they were generated. Given this, we can compare the manually asserted Read hierarchy with the Grail hierarchy which the classifier has inferred from the structure of the Read codes & templates.

This seems to act as a quality assurance / validation exercise for both parties: each can compare their results with the other; furthermore, initial experiments suggest that this mechanism may be useful for more specific purposes for each party:

- to suggest hierarchical relationships between Read codes for the CCC;
- to map Read codes to the GALEN model.

This document is very much in draft, specifically to allow first sight to the techniques and tools to the CCC before the meeting in Manchester on August 7th 1997.

Contents

1 INTRODUCTION 152

1.1 BACKGROUND 152
1.2 SUMMARY 152
1.3 OVERVIEW 153

2 READ CODES & TEMPLATES ® DISSECTIONS 155

2.1 READ TEMPLATES 155
2.2 GRAIL INTERMEDIATE REPRESENTATION DISSECTIONS 155
2.3 CONVERTING READ TEMPLATES TO DISSECTIONS 156
2.3.1 Efficiency / scaleability 156

3 DISSECTIONS ® GRAIL 157

3.1 EFFICIENCY / SCALEABILITY 157

4 COMPARISON OF READ & GRAIL HIERARCHIES 158

4.1 GRAPHICAL HIERARCHY COMPARISON 158
4.2 NON-GRAPHICAL HIERARCHY COMPARISON 158
4.3 EFFICIENCY / SCALEABILITY 160

5 RESULTS 161

5.1 BENEFIT TO GALEN 161
5.2 BENEFIT TO CCC 161
5.3 FUTURE WORK 161

APPENDIX 1 HIERARCHY COMPARISON REPORT 162

1 Introduction

1.1 Background

GALEN is mapping existing coding/classification schemes to Grail (a formal compositional system), so as to help to organise those schemes. This is being done as part of the GALEN-IN-USE project. The Grail classifier automatically organises the classification hierarchy according to the declared structure and properties of elements in the Grail model.

GALEN has introduced an intermediate representation so that clinicians can be protected from the full complexity of Grail in a major task: that of mapping elements of existing coding/classification schemes to the Grail model (and of specifying which parts of the model need to be extended).

Read has a manually asserted hierarchy, but is also introducing a structural description of its codes by means of 'templates'[37].

There was prima-facie evidence that we could leverage the Read templates so as to able to transform Read codes into dissections, given that they both seem to be attempting to represent the structure of a code or term. Given that, could we demonstrate any benefit to ourselves, or to the CCC, in being able to bring the Grail classifier to bear on the structure of the Read codes? This document describes a short un-funded experiment to see.

1.2 Summary

The experiment seems to be bearing fruit. The rest of this document describes the experiment and its results in detail; a summary follows here:

- we can automatically transform Read codes & Read templates into Grail Intermediate Representation dissections, with just a small amount of fixed knowledge of how particular templates should be translated;
- we can compare the results of applying the Grail classifier to the structure of the Read codes: this points out 'areas of interest' in both the Grail model and in the Read codes and templates themselves;
- in an initial experiment, of 425 parent-to-child relationships discovered, 57% were common to both the Read and Grail hierarchies; 37% were unique to the Grail hierarchy, 2% were unique to the Read hierarchy, and 4% were cases where both parent and child Read codes were transformed to the same Grail concept;

[37] Confusion limitation exercise: Grail Intermediate Representation dissections were initially called 'templates' before they were re-named to 'dissections' - there may be places in the software/documentation that still refer to the obsolete term. In this document, the phrase 'Read template' will be used to refer to the Read structure.

- initial tools have been developed to explore these discrepancies/commonalities: further work is now required to determine how useful this material really is;
- there are still unanswered questions, and further directions to explore: how might this experiment scale? Can we make use of other kinds of Read templates?

1.3 Overview

Some additional software has been developed in the overall environment of the KnoME (the Grail Knowledge Modelling Environment) to conduct the experiment. It reuses much of what is already present in the KnoME for exploring the Grail model itself, and for managing dissections and translating them into Grail.

The experiment has a six-stage process:

1. Loading Read templates from file: the CCC provided a subsection of their current working database in a standard ASCII relational database format. This is parsed by the KnoME into an object-oriented structure for further examination and processing. Read codes used within the templates are extracted and maintained.

   This approach was taken as it provides a convenient way of being able to interactively explore the structure and usage of the templates and codes. An alternative, and one we may wish to pursue for a larger-scale experiment, is to maintain the Read templates in an external database and access them remotely from within the KnoME.

2. Converting Read codes to dissections within the GALEN Intermediate Representation. This process is described further in Section 2.

3. Converting dissections to Grail, and compiling them into the GALEN Common Reference Model (CRM). This process is described further in Section 3.

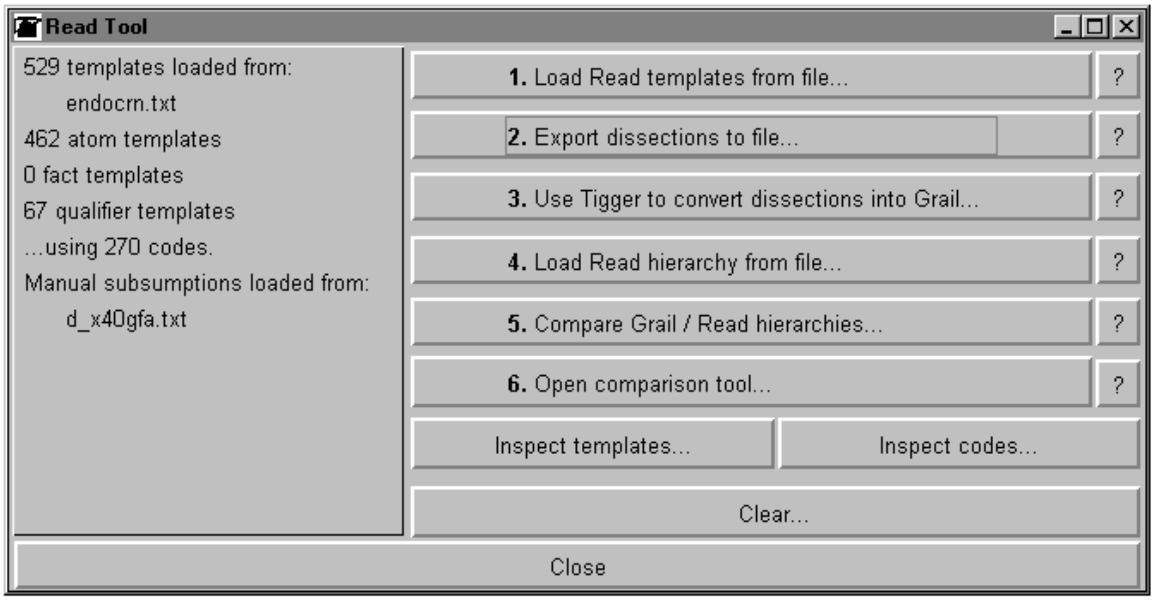
   This is as far as the process had got by the time of the meeting between Manchester and CCC in Loughborough on Monday 19th May 1997.

4. By now, Read codes have been represented as elements with the Grail CRM (typically, as large and complex composite Grail concepts), mapped back to Read codes. This is useful in itself (as described in Section 5.1). However, in order to verify these results, and explore additional benefits that might be available to the CCC, we wished to compare the classification of Read codes à la Grail, and their native (manually asserted) classification. To do this, we first need to import the native Read classification, which is again provided by CCC as an ASCII dump from their database.

5. Now all the data is in place, the system tries to find discrepancies and similarities between the classifications. These are stored in an internal form, ready to be further examined…

6. To examine and compare the two classifications, a tool was developed to help examine the comparison between the two classifications. This tool is in its early stage of development, but has already proved useful in filtering out some of the noise in the system. An important part of the next stage is to determine whether this sort of tool, or what others, prove useful. This is described further in Section 4.

To support the overall process, an interface was developed, illustrated below:


Read Tool
529 templates loaded from:
endocrn.txt
462 atom templates
0 fact templates
67 qualifier templates
...using 270 codes.
Manual subsumptions loaded from:
d_x40gfa.txt
1. Load Read templates from file...
?
2. Export dissections to file...
?
3. Use Tigger to convert dissections into Grail...
?
4. Load Read hierarchy from file...
?
5. Compare Grail / Read hierarchies...
?
6. Open comparison tool...
?
Inspect templates...
Inspect codes...
Clear...
Close

2 Read codes & templates ® dissections

This section describes how Read codes and templates are converted to Grail Intermediate Representation templates. This forms the crux of where CCC and GALEN meet: the hypothesis that this was possible drove the original experiment.

2.1 Read templates

There are three kinds of templates that are currently being used in CCC to represent the structure of the Read codes:

'atomic' templates, representing information about the definitional structure of codes. For example:

7100.' 'Hypophysectomy' 'X9019' 'Site' 'Xa06A' 'Pituitary structure'

This seems to match the Grail notion of 'defining criteria', for example that one of the defining criteria of the Grail concept Hypophysectomy (if it happened to be so named) is:

actsSpecificallyOn PituitaryGland.

There were 462 'atomic' templates in the initial sample.

1. 'fact' templates, which seem to correspond to the notion of Grail 'necessary' statements. There were no 'fact' templates in the initial sample.
2. 'qualifier templates', which represent was might be said about the code, for example that

   '71020' 'Excision of lesion of pituitary gland' 'X9041' 'Extent' 'X7940' 'Total'.

   This seems to correspond with the Grail sanctioning statement

   (SurgicalRemovingProcess which actsSpecififcallyOn BodyPart)
   sensibly hasSurgicalExtent
   (SurgicalExtent which hasAbsoluteState organExtentState)

   There were 67 'qualifier' templates in the initial sample.

2.2 Grail Intermediate Representation dissections

Dissections are intended to provide a syntax for clinicians to represent the definitional structure of terms or phrases from existing coding and classification schemes, such that these dissections can be automatically be transformed into Grail expressions. Dissections are a deliberate simplification of Grail, to support this particular task, so that:

- clinicians don't have to know the full complexities of Grail modelling;

- there is a level of indirection between a large corpus of structured information, and the final Grail expressions. This is useful so as allow us to be able to modify the form of the final Grail expressions as and when the modelling style or detail changes.

By convention, dissections have a ‘deed’ as their head, and it is this deed which is further described in the rest of the dissection.

2.3 Converting Read templates to dissections

Examination of the templates suggested that the dissection notion of having a main deed corresponded to the value of a template whose attribute was the Read code 'X900S Method'. So, we determined that Read codes which fell into this pattern could be converted into dissections, with any other ‘atomic’ templates being transformed into further details within the dissection. This process was followed recursively, following the structure represented by the ‘atomic’ templates.

In the initial sample, this resulted in 162 dissections, each of which represented a single Read code.

To correspond to the syntactic convention that dissection links are in upper-case, this transformation was applied to the text of Read codes acting as links. There were also 3 explicit mappings applied between Read codes and dissection links:

Read code	**Dissection link**
Extent	HAS_EXTENT
Revision status	HAS_TEMPORAL_MARKER
Approach	HAS_APPROACH

An example of a resulting dissection from the Read code ‘7100. Hypophysectomy’:

```
RUBRIC "Hypophysectomy"
ENGLISH_RUBRIC "Hypophysectomy"
CODE "7100."
SOURCE "READ"
MAIN excision action
        SITE pituitary structure
```

2.3.1 Efficiency/scaleability

This process seems to be a computationally trivial task (certainly in comparison with some of the later step); it takes less than one second to transform 462 atomic templates into 162 dissections

3 Dissections ® Grail

The process of transforming between dissections and Grail is exactly that developed in GALEN-IN-USE to support the work of the participating classification centres, and there has been a good deal of work already done to manage and support this process - indeed that was a major reason for targeting dissections, rather than Grail expressions, as the first conversion stage of the overall process.

There are 3 steps to the dissection-to-grail process: one manual, one semi-automatic, and one automatic:

1. link mapping: expressing how dissection links should map into Grail attributes or chains of attribute-concept-attribute. There may be multiple mappings, which are in force depending on context. This stage is entirely manual, but only has to be done once;
2. descriptor mappings: expressing how descriptors are mapped to Grail concepts. Dissection descriptors are the text of Read codes in this experiment. The task of mapping them is semi-automatic, in that lexical techniques can be used to help the mapping author. However, it may be that additional domain modelling is necessary, which is purely manual.

   Not all the descriptors have been mapped: there were 11 that have not been mapped.
3. dissection-to-grail transformation and classification: given link and descriptor mappings, this process is completely automatic.

The tool-set of creating and managing dissections and mappings, and for applying the conversion and classification process, is called ‘Tigger’.

3.1 Efficiency/scaleability

For the initial sample, there were 7 links used, of which 3 were not already mapped. Mapping these 3 took essentially very little time. It is not expected that this set will grow rapidly, but it depends on the number of attributes used in Read templates.

There were 71 descriptors (Read code terms) used in the 162 dissections. Additional domain modelling took approximately a half-day, and mapping the descriptors to that domain model took approximately another half-day. This task will scale better than linearly, however, as much of the work is coping with different spellings and notational variants, which, once in, will re-occur in other areas. 143 dissections could be compiled into Grail (those with unmapped descriptors couldn’t be).

The automatic process of transforming dissections to Grail takes approximately 1 second per dissection, and it takes a similar time to classify the resulting Grail expression. This seems to be a constant time, though we might expect classification to slow down as the hierarchy becomes more populated.

4 Comparison of Read & Grail hierarchies

Given the transformation of Read codes & templates into dissections, and thence into classified Grail expressions, we have the opportunity to compare the manually asserted Read hierarchy, with the automatically generated Grail hierarchy which is made on the basis of the structural description of the Read codes in the Read templates. This seems to offer the ability of an additional quality assurance check for both GALEN & CCC, and maybe to help manage the CCC's manual hierarchy. The question the arises: how to explore and examine these two hierarchies?

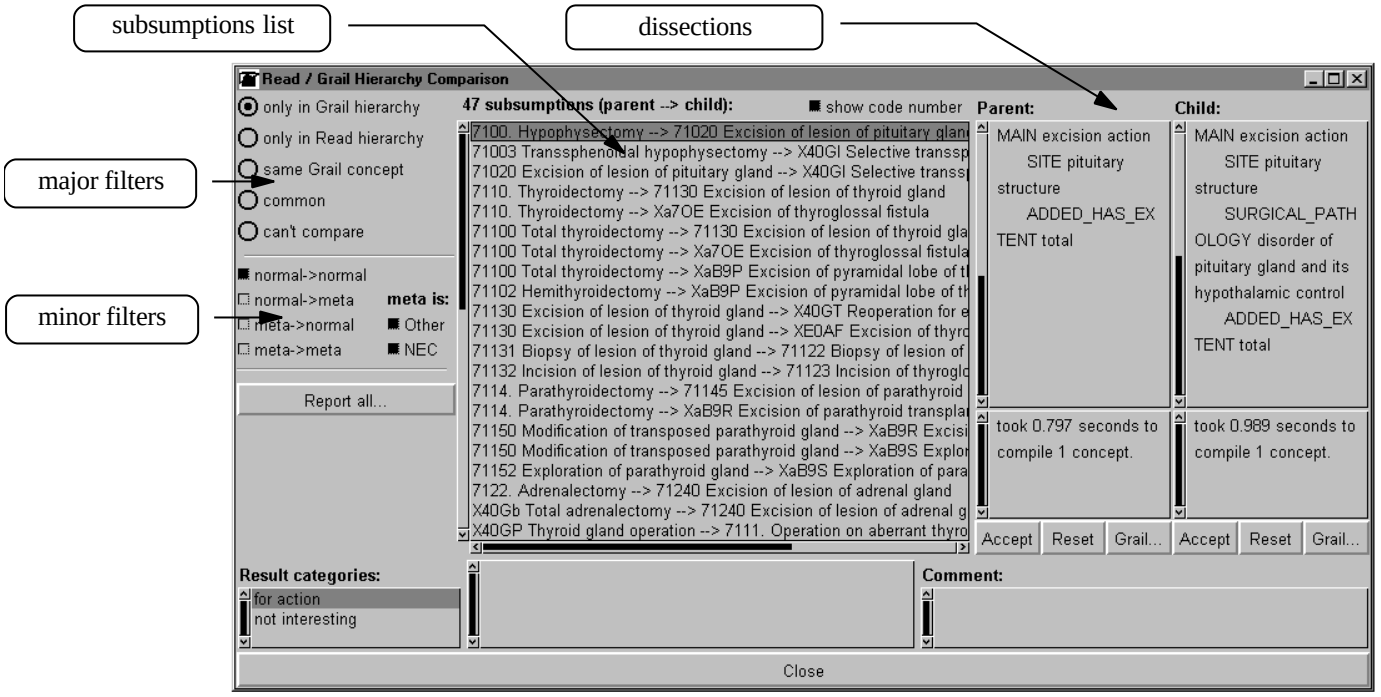
4.1 Graphical hierarchy comparison

A ubiquitous tool for exploring Grail (and Read) classifications is a graphical hierarchy browser which allows interactive examination of the parents and children of any concept or code, to an arbitrary depth. One obvious route toward solving the hierarchy comparison problem is to imagine a graphical tool that offered two views (one of the Grail model, one of the Read hierarchy), with differences and similarities being pointed out using graphical cues. However, this sounds to the author like a VCTI[38], so is left as an exercise for the interested reader.

4.2 Non-graphical hierarchy comparison

As an easier, more pragmatic, approach to a solution, we have individually examined each of the set of parent-child relationships in both the Grail and Read hierarchies, and categorised them according to, for example, whether they are unique or shared. We have developed a tool to support this, illustrated below:

[38] Very Clever Tool Indeed



The 'major filter' radio buttons allow you to choose between one of five categorisations:

1. Grail only: these subsumptions only exist in the Grail model, not in the Read hierarchy. These might indicate missing elements of the Read hierarchy, or some problem with the Grail model;
2. Read only: these subsumptions are only present in the Read hierarchy. These might indicate limitations in the structural description of Read codes (by Read templates), with some problem with the Grail model, or with some difference in modelling styles;
3. same Grail concept: these subsumptions are present in the Read hierarchy, but the parent and child have both been transformed into the same Grail concept. This probably illustrates limitations in the Read templates;
4. common: these subsumptions are present in both the Read & Grail hierarchies, illustrating some validation of both.
5. can't compare: one or other of the parent or child in subsumptions that are present in the Read hierarchies could not be successfully translated into dissections, or those dissections couldn't be translated into Grail.

The minor filters allow the further filtering of the list of subsumptions by whether either or both of the parent / child of each subsumption involve a 'meta' code, where 'meta' can be interpreted as either or both of 'Other' (including NOS) or 'NEC'. This is useful as it is an area which seems to be qualitatively different from the non-meta codes and their classification.

The 'Report all...' button dumps a textual description of these categorisations (and their contents) to file; the result is attached as an appendix to this document.

The dissection produced for the parent and child of each subsumption is shown to the right of the tool, as a further debugging / exploration aid.

The bottom area of the tool is intended to function as an area for further manual reorganisation, perhaps for users to pull out particularly interesting subsumptions. These can be drag-and-dropped around the tool. Individual sets can then be dumped to text file as either text with any comments that may have been added, or as '|' delimited database format in the form:

```
|<child-code-number>|<parent-code-number>|
```

intended to be able to be imported into other tools or databases.

4.3 Efficiency/scaleability

We have had very little experience with using this tool, so it is, as yet, hard to know how far it is useable in practice. It may well point out areas of interest to both CCC and GALEN; the next stage is to obtain feedback from CCC.

5 Results

5.1 Benefit to GALEN

The experiment has already offered benefits to GALEN, in three areas:

- code mapping: we have demonstrated a technique of helping to map Read codes to the GALEN model;
- quality assurance: early results have pointed out areas of interest in the Grail model (i.e. using the technique as a validation exercise);
- prototyping: the exercise has provided real examples with which to prototype functionality that may be required by classification centres within GALEN-IN-USE.

5.2 Benefit to CCC

The CCC have, as yet, only had close sight of the first half of the overall process. We need to obtain feedback from them as to the usefulness to CCC of the hierarchy comparison techniques described in Section 4. It is intended that this document makes the meeting in Manchester on August 7th more productive.

5.3 Future work

So far, we have spent relatively little time on this experiment (a few days each of developer & clinician time), and have obtained some promising results. There are several directions for future work, but they may require proper resourcing:

- scaling up: we have made a demonstration on a small sample: it needs to be scaled up, both to actually do the work, and to demonstrate that scaling up is, indeed, possible;
- other areas: there are other leads to follow: for example, we can do a similar exercise comparing the Read ‘qualifier’ templates with Grail sanctioning statements.

Appendix 1 Hierarchy comparison report

For completeness, this appendix is a report showing a view of the comparison between Grail and Read hierarchies organised only by major categories. The tool described in Section 4 is intended to help view and organise this information.

This hierarchy report generated from the Read / Grail comparison tool
on August 1, 1997 at 11:08:01 am.

157 only formal relationships (found from Grail model).
8 only explicit relationships (found from imported Read hierarchy file).
242 relationships in common.
18 relationships where different Read codes have been mapped to the same Grail concept.
31 relationships that couldn't be compared.

157 only formal relationships (found from Grail model):

Normal -> Normal (no involvement of OTHER or NEC):

- Adrenal gland operation --> Operation on aberrant adrenal tissue
- Adrenalectomy --> Excision of lesion of adrenal gland
- Biopsy of lesion of thyroid gland --> Biopsy of lesion of thyroglossal tract
- Endocrine: surg. biopsy --> Biopsy of adrenal gland
- Endocrine: surg. biopsy --> Biopsy of lesion of pituitary gland
- Endocrine: surg. biopsy --> Biopsy of parathyroid gland
- Endocrine: surg. biopsy --> Thyroid:surgical biopsy
- Excision of lesion of pituitary gland --> Selective transsphenoidal pituitary adenomectomy
- Excision of lesion of thyroid gland --> Excision of thyroglossal cyst
- Excision of lesion of thyroid gland --> Reoperation for excision of thyroglossal tissue
- Excision of thyroglossal fistula --> Excision of thyroglossal cyst
- Excision of thyroglossal fistula --> Excision of thyroglossal tract
- Excision of thyroglossal fistula --> Reoperation for excision of thyroglossal tissue
- Hemithyroidectomy --> Excision of pyramidal lobe of thyroid
- Hypophysectomy --> Excision of lesion of pituitary gland
- Incision of lesion of thyroid gland --> Incision of thyroglossal cyst
- Lobectomy of thyroid gland --> Excision of pyramidal lobe of thyroid
- Modification of transposed parathyroid gland --> Excision of parathyroid transplant
- Operation on lesion of thyroid gland --> Operation on thyroglossal tissue
- Parathyroidectomy --> Excision of lesion of parathyroid gland
- Parathyroidectomy --> Excision of parathyroid transplant
- Partial substernal thyroidectomy --> Bilateral subtotal thyroidectomy
- Partial substernal thyroidectomy --> Hemithyroidectomy
- Partial substernal thyroidectomy --> Isthmectomy of thyroid gland
- Partial substernal thyroidectomy --> Lobectomy of thyroid gland
- Partial substernal thyroidectomy --> Subtotal thyroidectomy
- Partial thyroidectomy --> Bilateral subtotal thyroidectomy
- Partial thyroidectomy --> Subtotal thyroidectomy
- Removal of thyroid nodule --> Excision of lesion of thyroid gland
- Removal of thyroid nodule --> Excision of thyroglossal fistula
- Removal of thyroid nodule --> Partial substernal thyroidectomy
- Removal of thyroid nodule --> Partial thyroidectomy
- Removal of thyroid nodule --> Total thyroidectomy
- Surgical biopsy of endocrine system NOS --> Biopsy of adrenal gland
- Surgical biopsy of endocrine system NOS --> Biopsy of lesion of pituitary gland
- Surgical biopsy of endocrine system NOS --> Biopsy of parathyroid gland
- Surgical biopsy of endocrine system NOS --> Thyroid:surgical biopsy
- Thyroid gland operation --> Operation on aberrant thyroid tissue
- Thyroid incision --> Incision of lesion of thyroid gland
- Thyroid:surgical biopsy --> Biopsy of lesion of thyroid gland
- Thyroidectomy --> Excision of lesion of thyroid gland
- Thyroidectomy --> Excision of thyroglossal fistula
- Total substernal thyroidectomy --> Excision of lesion of thyroid gland
- Total substernal thyroidectomy --> Excision of thyroglossal fistula
- Total substernal thyroidectomy --> Partial substernal thyroidectomy
- Total substernal thyroidectomy --> Partial thyroidectomy
- Total substernal thyroidectomy --> Total thyroidectomy
- Transsphenoidal hypophysectomy --> Selective transsphenoidal pituitary adenomectomy

Normal -> Meta (child is either OTHER or NEC):

- Hemithyroidectomy --> Lobectomy of thyroid gland NEC
- Injection of neurolytic substance into pituitary gland --> Injection of destructive substance into pituitary gland NEC
- Partial substernal thyroidectomy --> Partial thyroidectomy NEC
- Removal of thyroid nodule --> Other specified thyroidectomy
- Removal of thyroid nodule --> Thyroidectomy NEC
- Removal of thyroid nodule --> Thyroidectomy NOS
- Total substernal thyroidectomy --> Other specified thyroidectomy
- Total substernal thyroidectomy --> Thyroidectomy NEC

Total substernal thyroidectomy --> Thyroidectomy NOS
Meta -> Normal (parent is either OTHER or NEC):
~~ none ~~
Meta -> Meta (both parent and child are either OTHER or NEC):
Other endocrine gland operations --> Operation on pineal gland NOS
Other endocrine gland operations --> Other operation on adrenal gland NOS
Other endocrine gland operations --> Other operation on parathyroid gland NOS
Other endocrine gland operations --> Other operation on pituitary gland NOS
Other endocrine gland operations --> Other operation on thyroid gland NOS
Other endocrine gland operations --> Other operations on adrenal gland
Other endocrine gland operations --> Other operations on parathyroid gland
Other endocrine gland operations --> Other operations on pituitary gland
Other endocrine gland operations --> Other operations on thyroid gland
Other endocrine gland operations --> Other specified operation on pineal gland
Other endocrine gland operations --> Other specified other operation on adrenal gland
Other endocrine gland operations --> Other specified other operation on parathyroid gland
Other endocrine gland operations --> Other specified other operation on pituitary gland
Other endocrine gland operations --> Other specified other operation on thyroid gland
Other endocrine gland operations NOS --> Operation on pineal gland NOS
Other endocrine gland operations NOS --> Other operation on adrenal gland NOS
Other endocrine gland operations NOS --> Other operation on parathyroid gland NOS
Other endocrine gland operations NOS --> Other operation on pituitary gland NOS
Other endocrine gland operations NOS --> Other operation on thyroid gland NOS
Other endocrine gland operations NOS --> Other operations on adrenal gland
Other endocrine gland operations NOS --> Other operations on parathyroid gland
Other endocrine gland operations NOS --> Other operations on pituitary gland
Other endocrine gland operations NOS --> Other operations on thyroid gland
Other endocrine gland operations NOS --> Other specified operation on pineal gland
Other endocrine gland operations NOS --> Other specified other operation on adrenal gland
Other endocrine gland operations NOS --> Other specified other operation on parathyroid gland
Other endocrine gland operations NOS --> Other specified other operation on pituitary gland
Other endocrine gland operations NOS --> Other specified other operation on thyroid gland
Other operation on adrenal gland NOS --> Excision of adrenal gland NOS
Other operation on adrenal gland NOS --> Operation on aberrant adrenal tissue NOS
Other operation on adrenal gland NOS --> Other specified excision of adrenal gland
Other operation on adrenal gland NOS --> Other specified operation on aberrant adrenal tissue
Other operation on parathyroid gland NOS --> Excision of parathyroid gland NOS
Other operation on parathyroid gland NOS --> Other specified excision of parathyroid gland
Other operation on pituitary gland NOS --> Destruction of pituitary gland NOS
Other operation on pituitary gland NOS --> Excision of pituitary gland NOS
Other operation on pituitary gland NOS --> Other specified destruction of pituitary gland
Other operation on pituitary gland NOS --> Other specified excision of pituitary gland
Other operation on thymus gland NOS --> Excision of thymus gland NOS
Other operation on thymus gland NOS --> Other specified excision of thymus gland
Other operation on thyroid gland NOS --> Operation on aberrant thyroid tissue NOS
Other operation on thyroid gland NOS --> Operation on thyroglossal tissue NOS
Other operation on thyroid gland NOS --> Other specified operation on aberrant thyroid tissue
Other operation on thyroid gland NOS --> Other specified operation on thyroglossal tissue
Other operation on thyroid gland NOS --> Other specified thyroidectomy
Other operation on thyroid gland NOS --> Thyroidectomy NOS
Other operations on adrenal gland --> Excision of adrenal gland NOS
Other operations on adrenal gland --> Operation on aberrant adrenal tissue NOS
Other operations on adrenal gland --> Other specified excision of adrenal gland
Other operations on adrenal gland --> Other specified operation on aberrant adrenal tissue
Other operations on parathyroid gland --> Excision of parathyroid gland NOS
Other operations on parathyroid gland --> Other specified excision of parathyroid gland
Other operations on pituitary gland --> Destruction of pituitary gland NOS
Other operations on pituitary gland --> Excision of pituitary gland NOS
Other operations on pituitary gland --> Other specified destruction of pituitary gland
Other operations on pituitary gland --> Other specified excision of pituitary gland
Other operations on thymus gland --> Excision of thymus gland NOS
Other operations on thymus gland --> Other specified excision of thymus gland
Other operations on thyroid gland --> Operation on aberrant thyroid tissue NOS
Other operations on thyroid gland --> Operation on thyroglossal tissue NOS
Other operations on thyroid gland --> Other specified operation on aberrant thyroid tissue
Other operations on thyroid gland --> Other specified operation on thyroglossal tissue
Other operations on thyroid gland --> Other specified thyroidectomy
Other operations on thyroid gland --> Thyroidectomy NOS
Other specified operations on other endocrine gland --> Operation on pineal gland NOS
Other specified operations on other endocrine gland --> Other operation on adrenal gland NOS
Other specified operations on other endocrine gland --> Other operation on parathyroid gland NOS
Other specified operations on other endocrine gland --> Other operation on pituitary gland NOS
Other specified operations on other endocrine gland --> Other operation on thyroid gland NOS
Other specified operations on other endocrine gland --> Other operations on adrenal gland
Other specified operations on other endocrine gland --> Other operations on parathyroid gland
Other specified operations on other endocrine gland --> Other operations on pituitary gland
Other specified operations on other endocrine gland --> Other operations on thyroid gland

Other specified operations on other endocrine gland --> Other specified operation on pineal gland
Other specified operations on other endocrine gland --> Other specified other operation on adrenal gland
Other specified operations on other endocrine gland --> Other specified other operation on parathyroid gland
Other specified operations on other endocrine gland --> Other specified other operation on pituitary gland
Other specified operations on other endocrine gland --> Other specified other operation on thyroid gland
Other specified other operation on adrenal gland --> Excision of adrenal gland NOS
Other specified other operation on adrenal gland --> Operation on aberrant adrenal tissue NOS
Other specified other operation on adrenal gland --> Other specified excision of adrenal gland
Other specified other operation on adrenal gland --> Other specified operation on aberrant adrenal tissue
Other specified other operation on parathyroid gland --> Excision of parathyroid gland NOS
Other specified other operation on parathyroid gland --> Other specified excision of parathyroid gland
Other specified other operation on pituitary gland --> Destruction of pituitary gland NOS
Other specified other operation on pituitary gland --> Excision of pituitary gland NOS
Other specified other operation on pituitary gland --> Other specified destruction of pituitary gland
Other specified other operation on pituitary gland --> Other specified excision of pituitary gland
Other specified other operation on thymus gland --> Excision of thymus gland NOS
Other specified other operation on thymus gland --> Other specified excision of thymus gland
Other specified other operation on thyroid gland --> Operation on aberrant thyroid tissue NOS
Other specified other operation on thyroid gland --> Operation on thyroglossal tissue NOS
Other specified other operation on thyroid gland --> Other specified operation on aberrant thyroid tissue
Other specified other operation on thyroid gland --> Other specified operation on thyroglossal tissue
Other specified other operation on thyroid gland --> Other specified thyroidectomy
Other specified other operation on thyroid gland --> Thyroidectomy NOS
Parathyroidectomy NEC --> Global parathyroidectomy NEC
Parathyroidectomy NEC --> Partial parathyroidectomy NEC
Partial thyroidectomy NEC --> Lobectomy of thyroid gland NEC
Thyroidectomy NEC --> Partial thyroidectomy NEC

8 only explicit relationships (found from imported Read hierarchy file):

Normal -> Normal (no involvement of OTHER or NEC):

Destruction of pituitary gland --> Injection of destructive substance into pituitary gland
Endocrine system procedure --> Thymus gland operation
Excision of lesion of pituitary gland --> Pituitary tumour removal
Excision of lesion of thyroid gland --> Removal of thyroid nodule
Operation on thyroglossal tissue --> Excision of thyroglossal fistula
Operation on thyroglossal tissue --> Excision of thyroglossal tract
Total thyroidectomy --> Total substernal thyroidectomy
Transsphenoidal hypophysectomy --> Transethmoidal hypophysectomy

Normal -> Meta (child is either OTHER or NEC):

~~ none ~~

Meta -> Normal (parent is either OTHER or NEC):

~~ none ~~

Meta -> Meta (both parent and child are either OTHER or NEC):

~~ none ~~

242 relationships in common:

Normal -> Normal (no involvement of OTHER or NEC):

Adrenal gland operation --> Adrenalectomy
Adrenal gland operation --> Adrenalectomy
Adrenal gland operation --> Biopsy of adrenal gland
Adrenal gland operation --> Biopsy of adrenal gland
Adrenal gland operation --> Destruction of adrenal gland
Adrenal gland operation --> Destruction of adrenal gland
Adrenal gland operation --> Excision of lesion of adrenal gland
Adrenalectomy --> Partial adrenalectomy
Adrenalectomy --> Partial adrenalectomy
Adrenalectomy --> Total adrenalectomy
Adrenalectomy --> Total adrenalectomy
Biopsy of adrenal gland --> Biopsy of lesion of adrenal gland
Biopsy of adrenal gland --> Biopsy of lesion of adrenal gland
Biopsy of parathyroid gland --> Biopsy of lesion of parathyroid gland
Biopsy of parathyroid gland --> Biopsy of lesion of parathyroid gland
Destruction of pituitary gland --> Cryotherapy to pituitary gland
Destruction of pituitary gland --> Cryotherapy to pituitary gland
Destruction of pituitary gland --> Implantation of radioactive substance into pituitary gland
Destruction of pituitary gland --> Implantation of radioactive substance into pituitary gland
Destruction of pituitary gland --> Pituitary cryoablation
Destruction of pituitary gland --> Radiofrequency destruction of pituitary gland
Destruction of pituitary gland --> Radiofrequency destruction of pituitary gland
Endocrine system procedure --> Adrenal gland operation
Endocrine system procedure --> Adrenal gland operation
Endocrine system procedure --> Endocrine: surg. biopsy
Endocrine system procedure --> Endocrine: surg. biopsy
Endocrine system procedure --> Operation on aberrant adrenal tissue
Endocrine system procedure --> Operation on aberrant thyroid tissue
Endocrine system procedure --> Operation on thyroglossal tissue
Endocrine system procedure --> Parathyroid gland operation

Endocrine system procedure --> Parathyroid gland operation
Endocrine system procedure --> Pineal gland operation
Endocrine system procedure --> Pineal gland operation
Endocrine system procedure --> Pituitary gland operation
Endocrine system procedure --> Pituitary gland operation
Endocrine system procedure --> Surgical biopsy of endocrine system NOS
Endocrine system procedure --> Thyroid gland operation
Endocrine system procedure --> Thyroid gland operation
Excision of aberrant thyroid tissue --> Excision of sublingual thyroid tissue
Excision of aberrant thyroid tissue --> Excision of sublingual thyroid tissue
Excision of aberrant thyroid tissue --> Excision of substernal thyroid tissue
Excision of aberrant thyroid tissue --> Excision of substernal thyroid tissue
Excision of lesion of parathyroid gland --> Parathyroid adenoma excision
Excision of lesion of parathyroid gland --> Parathyroid adenoma excision
Excision of lesion of thyroid gland --> Excision of thyroid adenoma
Excision of lesion of thyroid gland --> Excision of thyroid adenoma
Hypophysectomy --> Transcranial hypophysectomy
Hypophysectomy --> Transcranial hypophysectomy
Hypophysectomy --> Transethmoidal hypophysectomy
Hypophysectomy --> Transsphenoidal hypophysectomy
Hypophysectomy --> Transsphenoidal hypophysectomy
Operation on aberrant adrenal tissue --> Excision of lesion of aberrant adrenal tissue
Operation on aberrant adrenal tissue --> Excision of lesion of aberrant adrenal tissue
Operation on aberrant thyroid tissue --> Excision of aberrant thyroid tissue
Operation on aberrant thyroid tissue --> Excision of aberrant thyroid tissue
Operation on lesion of pituitary gland --> Biopsy of lesion of pituitary gland
Operation on lesion of pituitary gland --> Biopsy of lesion of pituitary gland
Operation on lesion of pituitary gland --> Excision of lesion of pituitary gland
Operation on lesion of pituitary gland --> Excision of lesion of pituitary gland
Operation on lesion of pituitary gland --> Pituitary tumour removal
Operation on lesion of thyroid gland --> Biopsy of lesion of thyroid gland
Operation on lesion of thyroid gland --> Biopsy of lesion of thyroid gland
Operation on lesion of thyroid gland --> Excision of lesion of thyroid gland
Operation on lesion of thyroid gland --> Excision of lesion of thyroid gland
Operation on lesion of thyroid gland --> Incision of lesion of thyroid gland
Operation on lesion of thyroid gland --> Incision of lesion of thyroid gland
Operation on pituitary stalk --> Hypophyseal stalk section
Operation on pituitary stalk --> Hypophyseal stalk section
Operation on thyroglossal tissue --> Biopsy of lesion of thyroglossal tract
Operation on thyroglossal tissue --> Biopsy of lesion of thyroglossal tract
Operation on thyroglossal tissue --> Excision of thyroglossal cyst
Operation on thyroglossal tissue --> Excision of thyroglossal cyst
Operation on thyroglossal tissue --> Incision of thyroglossal cyst
Operation on thyroglossal tissue --> Incision of thyroglossal cyst
Operation on thyroglossal tissue --> Reoperation for excision of thyroglossal tissue
Operation on thyroglossal tissue --> Reoperation for excision of thyroglossal tissue
Operation on transposed parathyroid gland --> Excision of parathyroid transplant
Operation on transposed parathyroid gland --> Excision of parathyroid transplant
Parathyroid gland operation --> Biopsy of parathyroid gland
Parathyroid gland operation --> Biopsy of parathyroid gland
Parathyroid gland operation --> Destruction of parathyroid gland
Parathyroid gland operation --> Destruction of parathyroid gland
Parathyroid gland operation --> Excision of lesion of parathyroid gland
Parathyroid gland operation --> Modification of transposed parathyroid gland
Parathyroid gland operation --> Operation on transposed parathyroid gland
Parathyroid gland operation --> Operation on transposed parathyroid gland
Parathyroid gland operation --> Parathyroidectomy
Parathyroid gland operation --> Parathyroidectomy
Parathyroid gland operation --> Transplantation of parathyroid tissue
Parathyroid gland operation --> Transplantation of parathyroid tissue
Parathyroidectomy --> Excision of ectopic parathyroid gland
Parathyroidectomy --> Excision of ectopic parathyroid gland
Parathyroidectomy --> Partial parathyroidectomy
Parathyroidectomy --> Partial parathyroidectomy
Parathyroidectomy --> Total parathyroidectomy
Parathyroidectomy --> Total parathyroidectomy
Partial parathyroidectomy --> Partial parathyroidectomy and parathyroid tissue transposition
Partial parathyroidectomy --> Partial parathyroidectomy and parathyroid tissue transposition
Partial thyroidectomy --> Excision of pyramidal lobe of thyroid
Partial thyroidectomy --> Hemithyroidectomy
Partial thyroidectomy --> Hemithyroidectomy
Partial thyroidectomy --> Isthmectomy of thyroid gland
Partial thyroidectomy --> Isthmectomy of thyroid gland
Partial thyroidectomy --> Lobectomy of thyroid gland
Partial thyroidectomy --> Lobectomy of thyroid gland
Pineal gland operation --> Excision of pineal gland
Pineal gland operation --> Excision of pineal gland

Pituitary gland operation --> Chiasmopexy
Pituitary gland operation --> Chiasmopexy
Pituitary gland operation --> Decompression of pituitary gland
Pituitary gland operation --> Decompression of pituitary gland
Pituitary gland operation --> Destruction of pituitary gland
Pituitary gland operation --> Destruction of pituitary gland
Pituitary gland operation --> Hypophysectomy
Pituitary gland operation --> Hypophysectomy
Pituitary gland operation --> Injection of destructive substance into pituitary gland
Pituitary gland operation --> Injection of neurolytic substance into pituitary gland
Pituitary gland operation --> Operation on lesion of pituitary gland
Pituitary gland operation --> Operation on lesion of pituitary gland
Pituitary gland operation --> Operation on pituitary stalk
Pituitary gland operation --> Operation on pituitary stalk
Pituitary gland operation --> Repair of CSF leak from pituitary fossa
Pituitary gland operation --> Repair of CSF leak from pituitary fossa
Pituitary gland operation --> Selective transsphenoidal pituitary adenomectomy
Thymectomy --> Mediastinoscopic thymectomy
Thymectomy --> Mediastinoscopic thymectomy
Thymectomy --> Transcervical thymectomy
Thymectomy --> Transcervical thymectomy
Thymectomy --> Transsternal thymectomy
Thymectomy --> Transsternal thymectomy
Thymus gland operation --> Biopsy of lesion of thymus gland
Thymus gland operation --> Biopsy of lesion of thymus gland
Thymus gland operation --> Thymectomy
Thymus gland operation --> Thymectomy
Thymus gland operation --> Thymus transplantation
Thymus gland operation --> Thymus transplantation
Thyroid gland operation --> Drainage of thyroid
Thyroid gland operation --> Drainage of thyroid
Thyroid gland operation --> Operation on lesion of thyroid gland
Thyroid gland operation --> Operation on lesion of thyroid gland
Thyroid gland operation --> Removal of thyroid nodule
Thyroid gland operation --> Thyroid incision
Thyroid gland operation --> Thyroid incision
Thyroid gland operation --> Thyroid:surgical biopsy
Thyroid gland operation --> Thyroid:surgical biopsy
Thyroid gland operation --> Thyroidectomy
Thyroid gland operation --> Thyroidectomy
Thyroid gland operation --> Total substernal thyroidectomy
Thyroidectomy --> Partial substernal thyroidectomy
Thyroidectomy --> Partial thyroidectomy
Thyroidectomy --> Partial thyroidectomy
Thyroidectomy --> Subtotal thyroidectomy
Thyroidectomy --> Total thyroidectomy
Thyroidectomy --> Total thyroidectomy
Total parathyroidectomy --> Total parathyroidectomy and parathyroid tissue transplantation
Total parathyroidectomy --> Total parathyroidectomy and parathyroid tissue transplantation
Total thyroidectomy --> Total thyroidectomy with cervical lymph node dissection
Total thyroidectomy --> Total thyroidectomy with cervical lymph node dissection

Normal -> Meta (child is either OTHER or NEC):

Adrenal gland operation --> Other operation on adrenal gland NOS
Adrenal gland operation --> Other operations on adrenal gland
Adrenal gland operation --> Other operations on adrenal gland
Adrenal gland operation --> Other specified other operation on adrenal gland
Adrenalectomy --> Adrenalectomy NEC
Adrenalectomy --> Adrenalectomy NEC
Adrenalectomy --> Excision of adrenal gland NOS
Adrenalectomy --> Excision of adrenal gland NOS
Adrenalectomy --> Other specified excision of adrenal gland
Adrenalectomy --> Other specified excision of adrenal gland
Destruction of pituitary gland --> Destruction of pituitary gland NOS
Destruction of pituitary gland --> Destruction of pituitary gland NOS
Destruction of pituitary gland --> Other specified destruction of pituitary gland
Destruction of pituitary gland --> Other specified destruction of pituitary gland
Endocrine system procedure --> Other endocrine gland operations
Endocrine system procedure --> Other endocrine gland operations
Endocrine system procedure --> Other endocrine gland operations NOS
Endocrine system procedure --> Other specified operations on other endocrine gland
Hypophysectomy --> Excision of pituitary gland NOS
Hypophysectomy --> Excision of pituitary gland NOS
Hypophysectomy --> Hypophysectomy NEC
Hypophysectomy --> Hypophysectomy NEC
Hypophysectomy --> Other specified excision of pituitary gland
Hypophysectomy --> Other specified excision of pituitary gland
Injection of destructive substance into pituitary gland --> Injection of destructive substance into pituitary gland NEC

Injection of destructive substance into pituitary gland --> Injection of destructive substance into pituitary gland NEC
Lobectomy of thyroid gland --> Lobectomy of thyroid gland NEC
Lobectomy of thyroid gland --> Lobectomy of thyroid gland NEC
Operation on aberrant adrenal tissue --> Operation on aberrant adrenal tissue NOS
Operation on aberrant adrenal tissue --> Operation on aberrant adrenal tissue NOS
Operation on aberrant adrenal tissue --> Other specified operation on aberrant adrenal tissue
Operation on aberrant adrenal tissue --> Other specified operation on aberrant adrenal tissue
Operation on aberrant thyroid tissue --> Operation on aberrant thyroid tissue NOS
Operation on aberrant thyroid tissue --> Operation on aberrant thyroid tissue NOS
Operation on aberrant thyroid tissue --> Other specified operation on aberrant thyroid tissue
Operation on aberrant thyroid tissue --> Other specified operation on aberrant thyroid tissue
Operation on thyroglossal tissue --> Operation on thyroglossal tissue NOS
Operation on thyroglossal tissue --> Operation on thyroglossal tissue NOS
Operation on thyroglossal tissue --> Other specified operation on thyroglossal tissue
Operation on thyroglossal tissue --> Other specified operation on thyroglossal tissue
Parathyroid gland operation --> Other operation on parathyroid gland NOS
Parathyroid gland operation --> Other operations on parathyroid gland
Parathyroid gland operation --> Other operations on parathyroid gland
Parathyroid gland operation --> Other specified other operation on parathyroid gland
Parathyroidectomy --> Excision of parathyroid gland NOS
Parathyroidectomy --> Excision of parathyroid gland NOS
Parathyroidectomy --> Other specified excision of parathyroid gland
Parathyroidectomy --> Other specified excision of parathyroid gland
Parathyroidectomy --> Parathyroidectomy NEC
Parathyroidectomy --> Parathyroidectomy NEC
Partial parathyroidectomy --> Partial parathyroidectomy NEC
Partial parathyroidectomy --> Partial parathyroidectomy NEC
Partial thyroidectomy --> Partial thyroidectomy NEC
Partial thyroidectomy --> Partial thyroidectomy NEC
Pineal gland operation --> Operation on pineal gland NOS
Pineal gland operation --> Operation on pineal gland NOS
Pineal gland operation --> Other specified operation on pineal gland
Pineal gland operation --> Other specified operation on pineal gland
Pituitary gland operation --> Other operation on pituitary gland NOS
Pituitary gland operation --> Other operations on pituitary gland
Pituitary gland operation --> Other operations on pituitary gland
Pituitary gland operation --> Other specified other operation on pituitary gland
Thymectomy --> Excision of thymus gland NOS
Thymectomy --> Excision of thymus gland NOS
Thymectomy --> Other specified excision of thymus gland
Thymectomy --> Other specified excision of thymus gland
Thymectomy --> Thymectomy NEC
Thymectomy --> Thymectomy NEC
Thymus gland operation --> Other operation on thymus gland NOS
Thymus gland operation --> Other operations on thymus gland
Thymus gland operation --> Other operations on thymus gland
Thymus gland operation --> Other specified other operation on thymus gland
Thyroid gland operation --> Other operation on thyroid gland NOS
Thyroid gland operation --> Other operations on thyroid gland
Thyroid gland operation --> Other operations on thyroid gland
Thyroid gland operation --> Other specified other operation on thyroid gland
Thyroidectomy --> Other specified thyroidectomy
Thyroidectomy --> Other specified thyroidectomy
Thyroidectomy --> Thyroidectomy NEC
Thyroidectomy --> Thyroidectomy NEC
Thyroidectomy --> Thyroidectomy NOS
Thyroidectomy --> Thyroidectomy NOS
Total parathyroidectomy --> Global parathyroidectomy NEC
Total parathyroidectomy --> Global parathyroidectomy NEC

Meta -> Normal (parent is either OTHER or NEC):
~~ none ~~
Meta -> Meta (both parent and child are either OTHER or NEC):
~~ none ~~

18 relationships where different Read codes have been mapped to the same Grail concept:
Normal -> Normal (no involvement of OTHER or NEC):
Cryotherapy to pituitary gland --> Pituitary cryoablation
Endocrine: surg. biopsy --> Surgical biopsy of endocrine system NOS
Injection of destructive substance into pituitary gland --> Injection of neurolytic substance into pituitary gland
Operation on transposed parathyroid gland --> Modification of transposed parathyroid gland
Partial thyroidectomy --> Partial substernal thyroidectomy
Subtotal thyroidectomy --> Bilateral subtotal thyroidectomy
Normal -> Meta (child is either OTHER or NEC):
~~ none ~~
Meta -> Normal (parent is either OTHER or NEC):
~~ none ~~
Meta -> Meta (both parent and child are either OTHER or NEC):

Other endocrine gland operations --> Other endocrine gland operations NOS
Other endocrine gland operations --> Other specified operations on other endocrine gland
Other operations on adrenal gland --> Other operation on adrenal gland NOS
Other operations on adrenal gland --> Other specified other operation on adrenal gland
Other operations on parathyroid gland --> Other operation on parathyroid gland NOS
Other operations on parathyroid gland --> Other specified other operation on parathyroid gland
Other operations on pituitary gland --> Other operation on pituitary gland NOS
Other operations on pituitary gland --> Other specified other operation on pituitary gland
Other operations on thymus gland --> Other operation on thymus gland NOS
Other operations on thymus gland --> Other specified other operation on thymus gland
Other operations on thyroid gland --> Other operation on thyroid gland NOS
Other operations on thyroid gland --> Other specified other operation on thyroid gland

31 relationships that couldn't be compared:

Normal -> Normal (no involvement of OTHER or NEC):

Adrenal gland operation --> Embolisation of adrenal gland
Adrenal gland operation --> Exploration of adrenal gland
Adrenal gland operation --> Exploration of adrenal gland
Adrenal gland operation --> Transposition of adrenal tissue
Adrenalectomy --> Bilateral adrenalectomy
Adrenalectomy --> Transposition of adrenal tissue
Adrenalectomy --> Unilateral adrenalectomy
Bilateral adrenalectomy --> Bilateral adrenalectomy and transposition of adrenal tissue
Endocrine system procedure --> Pituitary gland and pineal gland operations
Endocrine system procedure --> Thyroid gland and parathyroid gland operations
Exploration of adrenal gland --> Exploration of aberrant adrenal tissue
Exploration of parathyroid gland --> Exploration of parathyroid transplant
Modification of transposed parathyroid gland --> Exploration of parathyroid transplant
Operation on aberrant adrenal tissue --> Exploration of aberrant adrenal tissue
Operation on transposed parathyroid gland --> Exploration of parathyroid transplant
Operation on transposed parathyroid gland --> Exploration of parathyroid transplant
Parathyroid gland operation --> Exploration of parathyroid gland
Parathyroid gland operation --> Exploration of parathyroid gland
Pituitary gland operation --> Exploration of pituitary gland
Pituitary gland operation --> Exploration of pituitary gland
Thymus gland operation --> Exploration of thymus gland
Thymus gland operation --> Exploration of thymus gland
Thyroid gland operation --> Exploration of thyroid gland
Thyroid gland operation --> Exploration of thyroid gland
Transsphenoidal hypophysectomy --> Sublabial hypophysectomy
Transsphenoidal hypophysectomy --> Transseptal hypophysectomy

Normal -> Meta (child is either OTHER or NEC):

Bilateral adrenalectomy --> Bilateral adrenalectomy NEC
Pituitary gland and pineal gland operations --> Other specified operations on pituitary gland or pineal gland
Pituitary gland and pineal gland operations --> Pituitary gland and pineal gland operations NOS
Thyroid gland and parathyroid gland operations --> Other specified operations on thyroid gland or parathyroid gland
Thyroid gland and parathyroid gland operations --> Thyroid gland and parathyroid gland operations NOS

Meta -> Normal (parent is either OTHER or NEC):

~~ none ~~

Meta -> Meta (both parent and child are either OTHER or NEC):

~~ none ~~

FILE ENDS

Appendix Three:
GRAIL Tutorial

GRAIL Tutorial

October 1997

This document presents a tutorial introduction to the GRAIL concept modelling language. It is designed to be used in an interactive manner, building up a small demonstrative model using the modelling tools which communicate with a GALEN Terminology Server.

This version of the tutorial is for use with the KnoME modelling environment (version KiT20c or above); different versions are available for different modelling environments.

For help or further information, please contact:

galen-help@cs.man.ac.uk

(+44) 161 275 6239/6133

Contents

1. INTRODUCTION 174

1.1 WHAT THIS TUTORIAL IS 174
1.2 WHAT THIS TUTORIAL ISN'T 174
1.3 STRUCTURE OF THIS TUTORIAL 175
1.4 WHERE TO FIND FURTHER INFORMATION 175

2. GALEN, AND MODELLING WITH GRAIL 176

2.1 A BRIEF HISTORY 176
2.2 WHAT IS A "TERMINOLOGICAL MODEL"? 177
2.3 PRE-TUTORIAL - AN INTERACTIVE INTRODUCTION TO THE GRAIL KNOME 177
2.3.1 Accessing the Tools 177
2.3.2 Emptying any existing model from the Terminology Server 178
2.3.3 Creating a new model 178
2.3.4 Viewing the model 179
2.4 SYNTAX 179
2.4.1 Use of Brackets 179
2.4.2 Use of full stops 180
2.4.3 Comments 180
2.5 A WORD ABOUT VOCABULARY 180
2.5.1 Changes in Vocabulary 180

3. AN INTRODUCTORY GRAIL TUTORIAL 182

3.1 AN EXAMPLE APPLICATION AREA 182
3.1.1 What we want to do 182
3.1.2 Doing it in GRAIL 183
3.2 FORMAL SUBSUMPTION: STATEMENTS, DEFINITIONS, AND COMPOSITE CATEGORIES 185
3.2.1 Creating attributes 185
3.2.2 Sanctioning statements 186
3.2.3 Defining Composite Categories 188
3.2.4 Cardinality 188
3.2.5 Criteria 189
3.2.6 Formal Subsumption 189
3.2.7 Value Types 189
3.2.8 Criteria 191
3.2.9 Naming 191
3.3 NECESSARY STATEMENTS AND CANONICAL FORM 192

3.3.1 Necessary statements 192
3.3.2 Canonical Forms 192
3.3.3 The Hierarchy 193
3.4 FURTHER FEATURES 195
3.4.1 Grammar level Queries (Grammar level Composite Categories) 195
3.4.2 Topic and Value necessary statements 196
3.5 SOME COMMON QUESTIONS 197
3.5.1 What is the difference between a criterion in a definition (a which expression) and a criterion in a necessary statement? 197
3.5.2 What is the model? 197

4. USING GRAIL WELL: A TUTORIAL IN BASIC GRAIL STYLE 199

4.1 RE-USE, APPLICATION INDEPENDENCE, AND MODELLING STYLE 199
4.2 THE EXAMPLE MODEL REVISITED 199
4.3 EXTENDING THE MODEL TO A SECOND CLASSIFICATION 200
4.4 REFORMULATING THE MODEL 201
4.4.1 Removing RoadVehicles and RailVehicles (flattening the asserted hierarchy) 202
4.4.2 Creating the attributes and values 202
4.4.3 Describing the elementary categories 202
4.4.4 Creating the abstractions 203
4.4.5 Sorting out the order of statements in the source file. 203
4.4.6 Checking the syntax 203
4.4.7 Checking the result 203
4.5 FINAL REMARKS ON THE EXAMPLE 204
4.5.1 Further reformulation of the model? The need for an 'ontology' 204
4.5.2 Natural kinds and elementary categories 204
4.5.3 When there must be multiple parents in the asserted hierarchy? 204
4.6 SUMMARY: PRINCIPLES OF BUILDING RE-USABLE MODELS IN GRAIL 205

5. REFINEMENT AND TRANSITIVITY: WHAT MAKES GRAIL DIFFERENT 206

5.1 TRANSITIVITY 206
5.2 THE SPECIALISEDBY OPERATOR ('REFINEMENT') 208
5.3 TRANSITIVE RELATIONS AND CARDINALITY 209
5.4 WHY ARE SPECIALISEDBY AND TRANSITIVEDOWN IMPORTANT 209
5.5 HOW TO REMEMBER HOW SPECIALISEDBY WORKS 210

6. OTHER FEATURES 211

6.1 EXTRINSIC INFORMATION, 'DEFAULTS', AND INDEXING 211
6.2 CODING SCHEMES 212
6.3 LANGUAGE 212

7. GLOSSARY.....213

1 Introduction

GRAIL is a concept modelling language that has been developed and used by the GALEN projects to represent medical terminology. It has also been used to model other domains. GRAIL stands for the GALEN Representation and Integration Language.

1.1 What this tutorial is

This tutorial is an attempt to soften the initial learning curve that has been seen when encountering GRAIL for the first time and to introduce you to what we consider to be good practice in GRAIL modelling. The document adopts a hands-on approach - through the use of examples - to gain experience in the modelling process, and of some of the supporting tools which have been developed.

To make best use of this tutorial, work through it step-by-step using the GRAIL modelling environment. If you do not have access to this environment, contact

`galen@cs.man.ac.uk`

for information on how to obtain and install the software. Note that there are now two different modelling environments. Be sure you get the version of the tutorial that goes with your modelling environment: this version is for use with the GRAIL KnoME (version KiT20c and above)[39].

This tutorial is designed to introduce the GRAIL language itself: it is *not* an introduction to the clinical models that have been developed using that language. A conscious decision was made not to use medicine in the examples, in an attempt to restrict the areas of discussion! Further information is available (see below) to introduce the way GRAIL has been used to represent clinical terminology in the GALEN projects.

This tutorial has been written from a limited number of people's experience and difficulties. We are keen that others - both existing members of the select group who model in GRAIL, and any newcomers - should contribute to the development of this tutorial.

1.2 What this tutorial isn't

This document is *not* an attempt to integrate all existing documentation into a handy pocket-sized edition and this document won't tell you in detail which mouse button to use for any but the basic functions - further documentation is available which describe the modelling environments in detail, and this will be needed if further modelling is to be undertaken.

This document is also not a tutorial in the GALEN ontology and how we use GRAIL, in detail, in GALEN. This is covered separately. It also does not cover the associated facilities for natural

[39] The two environments have roughly the same functionality from the point of view of this tutorial, but are arranged somewhat differently. Details of which button to click are different and the windows are arranged differently.

language handling, handling coding systems, and the use of GRAIL for indexing knowledge bases.

This tutorial is to get you started. There's lots more, but this needs to be mastered first.

1.3 Structure of this tutorial

After this short introduction, Section 2 provides some history of GALEN and GRAIL, and introduces the modelling process, and the tools used to perform and support that process. Section 3 is the tutorial proper, whilst Section 4 provides a discussion of the different ways of modelling in GRAIL: some preferred, some not. Section 5 concentrates on two features that distinguish GRAIL from its peers, and Section 6 provides an overview of some other features of GRAIL. A short glossary concludes the tutorial.

There are some sections in the tutorial specifically aimed at those with experience of other systems - description logics, conceptual graphs, or object oriented design environments. These 'side-bars' are marked optional and distinguished in the text. They are not needed to understand GRAIL, but may be useful to those coming to GRAIL from related environments to help them understand the similarities and differences between GRAIL and systems with which they are familiar.

An important thing to keep in mind is that modelling in and using GRAIL can only really be learnt through a 'hands on' approach. Follow the examples, but experiment on your own at any time.

1.4 Where to find further information

Much has been written on GRAIL, GALEN and the overall modelling process. The following documents may also be relevant:

- Overview of GALEN (Volume B of the GALEN Documentation Set) and a set of Frequently Asked Questions (Volume F);
- The GRAIL KnoME: Overview and User Guide, which offers full details of the modelling environment which is featured in this version of the tutorial;
- The GRAIL Concept Modelling Language for Medical Terminology, a paper which describes the GRAIL language in more detail, and in a different style, than this tutorial (a shortened version is published in the January-February 1997 issue of Artificial Intelligence in Medicine);
- for those wishing to use their knowledge of GRAIL to understand, or extend the clinical model being developed in the GALEN projects, see Deliverable 5.1 of the GALEN-IN-USE project.

Additional (possibly slightly historical) detailed information is available in the GALEN Project Deliverables (in particular 6, 14, and 15).

The GRAIL modelling environments are under continuous development - indeed they can only develop successfully with requirements from existing and new users. So, if there is any aspect of the environments that need changing or extending, or if you think that a totally new approach is needed - let us know!

2 GALEN, and modelling with GRAIL

2.1 A brief history

The GRAIL formalism was developed for use in the GALEN programme for formal representation of medical terminologies. It has since been found useful for a variety of other applications both within and outside medicine where there is a need to describe and classify information - be it the function of enzymes or DNA sequences, the content of pictures, or the nature of diseases and medical procedures.

The motivation for the special features of GRAIL is closely tied to its origins: the support for high quality user interfaces for doctors in the PEN&PAD system. PEN&PAD's aim is to allow the direct entry of detailed clinical data by the practitioner during the consultation, and to display that data in a useful way - to 'shed light' on the medical record.

It was found that building the interface which emerged from our User Centred Design Trials while using existing enumerative coding schemes (e.g. the enumerative Read classification of clinical terms) was simply impossible - a view borne out by the current state of commercial GP systems that attempt to use such schemes. GRAIL's immediate ancestor, SMK {xe "SMK "}(Structured Meta Knowledge), was developed to overcome some of the limitations of enumerative representations. SMK integrated medical terminological knowledge, medical records and the medical dialogue into a single formalism, with the knowledge base of medical terminology providing the semantic control over the patient record.

GRAIL is the development of the terminological, or conceptual, aspects of SMK.

Sidebar: Relation to other systems : skip this section if you are not familiar with Frame Systems, Conceptual Graphs or Description Logics)

In a broader context, GRAIL is a 'Description Logic', i.e. one of a class of formalisms which attempt to make rigorous the notions of informal 'Frame-based systems'. It is related to KL-ONE, Loom, Classic, or Back, although it uses a different notation. By comparison with other description logics, GRAIL has extra features to deal with part-whole relations and simple 'rules', but it lacks other features such as detailed handling of numbers which were found to be less important for medical terminology. (Some of these restrictions can be irksome, and some may be relaxed in future versions.)

Like other description logics, GRAIL is also closely related to Sowa's Conceptual Graphs, and much of GRAIL can be mapped onto the subset of conceptual graphs commonly used for describing terminology.

GRAIL's special features and its unusual syntax make it easy to follow a particular 'style' in developing 'terminological models' or 'ontologies'.. This style has been designed to make models more re-usable and less linked to particular applications. A detailed description of the GALEN ontology and modelling style can be found in Deliverable 5.1 of the GALEN-IN-USE project. This tutorial introduces only the mechanics of the language and the basic principles of modelling style.

GRAIL's definition is now (almost) stable. However, GRAIL is not the ultimate description logic or ultimate system for representing medical terminology. There will be successor systems. But GRAIL is an important step along the way and sufficient for many applications.

2.2 What is a "terminological model"?

Terminological {xe "terminological "}modelling represents the definitions and essential properties of concepts. The terminological definition of a concept provides sufficient information to define, or recognise, that concept. Contrast this with, for example, modelling facts or assertions about concepts, which we may know to be true in the world, either about particular concepts, or the interaction between them, but which do not add to the definition of those concepts. For example, 'wine' might be defined as a 'drink made of distilled grape juice'. Other information not part of the definition might include that wines are either red or white.

Terminology may be seen as the definitional 'backbone of the concept structure that we wish to describe. It may well be the case that one of the important uses of such a structure is to handle non-definitional assertions about the real world. The features GRAIL provides to handle assertions will be illustrated in the examples.

Terminological models are also called 'ontologies' by many authors, although other authors use the word 'ontologies' to indicate a rather broader kind of knowledge base than that which we consider 'terminological'.

Finally, it is not always clear whether what you wish to say falls into the category of terminological knowledge, or not. Don't worry, but come back to this point later, after you've been through the tutorial. If in doubt, raise the issue personally or by email.

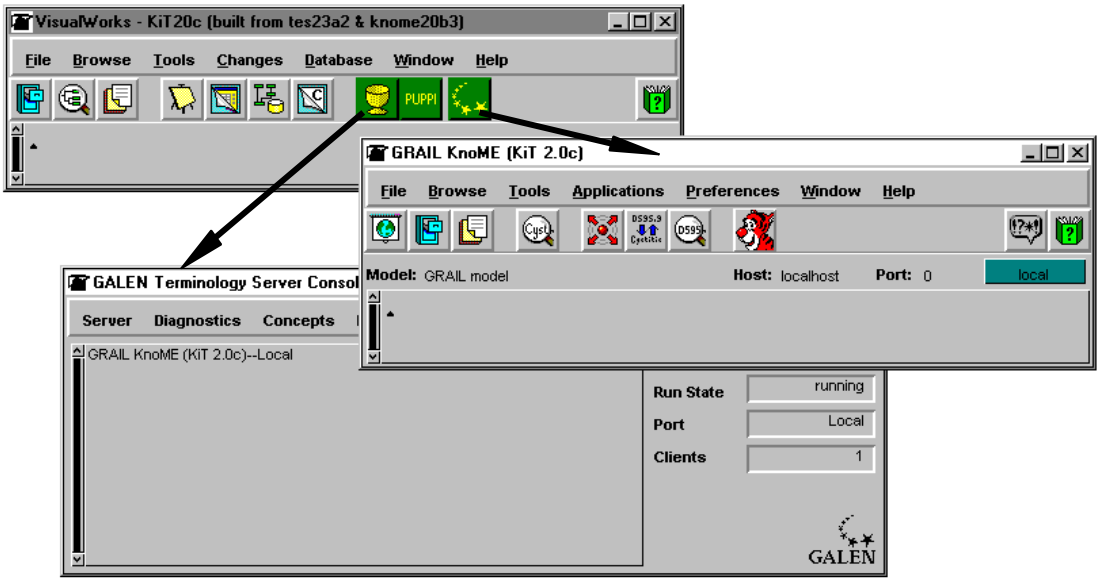
2.3 Pre-Tutorial - An Interactive Introduction to the GRAIL KnoME

One of the GRAIL development environments that is available is known as the 'KnoME' (for Knowledge Modelling Environment), which is a suite of modelling tools for building, browsing and exploring a GRAIL model. This pre-tutorial will provide you wish enough of a starting point to progress with the tutorial. See 'The GRAIL KnoME: Overview and User Guide' for more detailed coverage of the environment.

2.3.1 Accessing the Tools

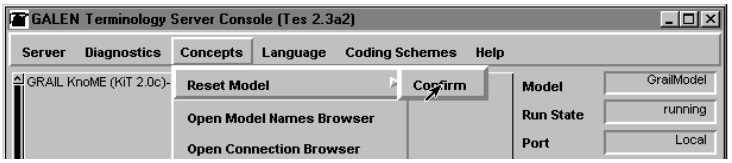
In the environment you will be using, the KnoME communicates with a Smalltalk implementation of a GALEN Terminology Server: most of the time you will only interact with the KnoME itself (via the KnoME Main Window), but occasionally you will need to access functions of the server itself (for example to empty a model) : a user interface for this is also provided (the Terminology Server Console).

When the environment starts (consult your local installation guide or technical support for details of this if you have difficulties), you will (at least) see the VisualWorks Main Window, from which both the KnoME Main Window and the Terminology Server Console may be started:



2.3.2 Emptying any existing model from the Terminology Server

Before starting to model, you need to make sure that there is no existing model being retained in the Terminology Server. Use the menu option from the Terminology Server Console **Concepts→Reset Model**:



Note that each time you reset the model, you will need to clear any cached results that the KnoME may have, use the menu option: **KnoME Main Window→File→Flush Cache**[40].

2.3.3 Creating a new model

The primary way of creating a model is via the GRAIL text compiler{xe "compiler"}, GRAIL expressions can be typed into a 'workspace' - a scratch area for compiling GRAIL expressions, or a file editor.

As a first start, create a new file by opening a GRAIL file editor (**Tools→GRAIL file editor**); this will either open a file that has been previously created, or create and open a new one. Type the text as shown in the screen-dump below. This GRAIL source is 'evaluated', or 'compiled', by using the evaluate option from the middle mouse-button menu. In other words, the GRAIL expressions you have entered are parsed into individual calls to the Terminology Server; they are then passed to the server and cause the GRAIL engine within the server to extend the model.

This is a super-simple model - all we have said is that there is a notion of an Ant, which is a kind of Insect, which is, in turn, a kind of Animal. This is just to introduce you to the tools.

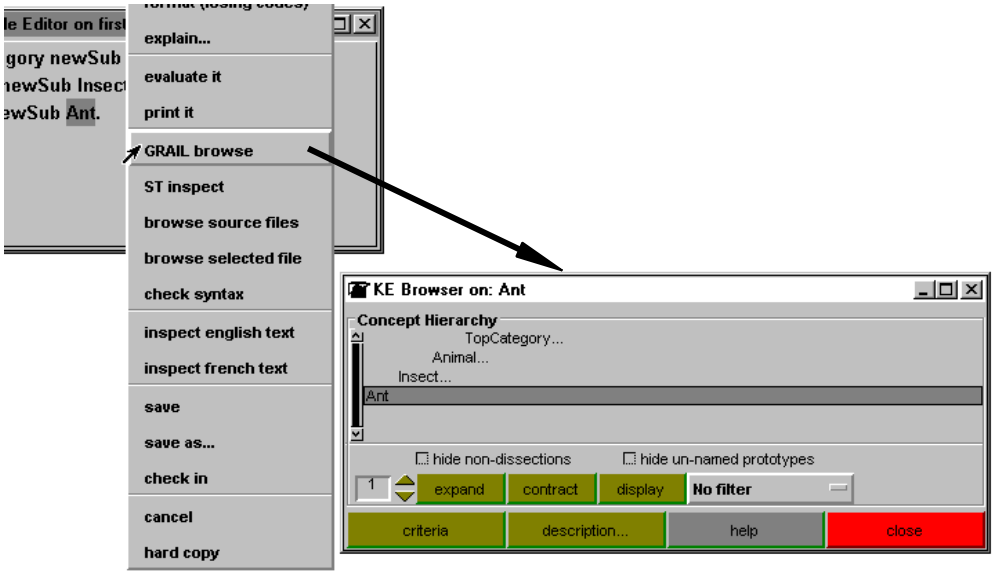
---

40 It may also be necessary to do this during interactive addition to the model: if some information doesn't appear in the browsers when you suspect it ought, try flushing the cache using this option.

2.3.4 Viewing the model

The primary tool for examining a model is called the Concept Browser. The browser always shows the hierarchical structure of the concept of interest (the browser's 'focus'), as well as offering the option of viewing other properties that may apply to the focus.

There are several ways of opening a browser, either using the menu option from the KnoME Main Window Browse→Browse Concept, or you can use the GRAIL browse option from the middle mouse-button menu from a GRAIL workspace or file editor:



Notice that the menu options apply to whatever is highlighted in the workspace. The **expand** and **contract** buttons are used to explore levels in the hierarchy (or try double-clicking concepts in the hierarchy).

The indentation of elements within the body of the browser indicates the hierarchical relationships that are in force. You might like to experiment with options on the middle-mouse button menu over the hierarchy, but important facilities will be introduced through the tutorial as you need them. See also 'The Grail KnoME: Overview and User Guide' for full descriptions of the browsers and other tools.

2.4 Syntax

2.4.1 Use of Brackets

One of the confusing things about the existing GRAIL syntax is that it uses three different kinds of brackets for three different purposes.

You will see different types of parentheses used in the compiler;

() Used to enclose a sub definition in long complicated definitions -' ordinary' brackets'. An example not using GRAIL syntax might be:

(people who are tall) who have (hair which is grey).

In this expression

(people who are tall)

defines a kind of *people* and

(hair which is grey)

defines a kind of *hair*. Together the two sub-definitions are used to define "tall people with grey hair", or if you prefer "tall grey-haired people".

[] Used for ‘expandable lists’ Expandable lists are a kind of short-hand ; these lists are expanded by the compiler before being passed to the underlying system. This will be explained more fully below, but the pattern is that

tell [mary john karen].

means exactly the same as

tell mary.
tell john.
tell karen.

<> Used to group criteria in definitions. Again, as another example not using proper GRAIL syntax,

People who <are tall have grey hair>

would be “people who are tall and have grey hair” or again “tall grey-haired people”. Note that the ‘and’ is understood inside <...>.

2.4.2 Use of full stops

All GRAIL expressions must end with a full stop. If you don't have a full stop at the end of a statement, you may get various odd syntax errors which are not always helpful. If you can’t figure out what is wrong with the syntax or why the GRAIL compiler does not recognise something, look for a missing full stop.

2.4.3 Comments

GRAIL treats anything between double quotes "..." as a comment. By convention long comments are put in the form:

“>>>>>>>>>>>>>>>>>>>>>
> This is a comment
> extending over two lines
<<<<<<<<<<<<<<<<<<<<<”

2.5 A word about Vocabulary

2.5.1 Changes in Vocabulary

The GRAIL vocabulary has evolved over the years. This tutorial attempts to be in line with current usage, but some older documentation may use slightly different words.

Two changes are perhaps worth noting. The earlier versions of this tutorial referred to ‘entities’ where this tutorial refers to ‘categories’. Earlier versions referred to ‘explicit subsumption’ where this tutorial refers to ‘asserted subsumption’.

Side-bar: GRAIL’s vocabulary compared with similar systems

GRAIL’s vocabulary was carefully selected to avoid clashes wherever possible with the vocabulary used in object oriented programming languages because GRAIL is itself usually implemented in an object oriented language. However, this sometimes causes confusion to new users. We hope the following will help.

In GRAIL, notions like “Dog”, “House” or “Swimming” are known as “categories” and usually written with an initial upper case letter. Categories correspond to ‘classes’ in object oriented languages, ‘types’ in Conceptual Graphs, or ‘concepts’ and ‘concept names’ in description logics.

GRAIL uses the word ‘attribute’ for the things which link categories together, corresponding to ‘relation’ in some object-oriented languages or ‘role’ in description logics. Conceptual Graphs also use the word ‘attribute’ in this way.

Two categories linked by an attribute make up a ‘statement’, or in the older usage, a ‘triple’. Some object oriented design formalisms refer to statements as ‘relationships’ Conceptual Graphs use the word ‘statement’ in the same way as does GRAIL. Description Logics are formulated in such a way as to avoid the notion.

If you find the vocabulary confusing because of previous experience, please refer to the table in the paper ‘The GRAIL Concept Modelling Language for Medical Terminology’. If you find it confusing because you are new to all such systems, please bear with us, and we hope it will become clearer as you work through the tutorial. If you still find it confusing, please get in touch with us.

3 An Introductory GRAIL tutorial

3.1 An example application area

3.1.1 What we want to do

GRAIL was originally designed for medical applications. However, the medicine often gets in the way on first acquaintance. This tutorial uses, as an example, a simple model of vehicles and transport.

Consider that we want to model a terminology about vehicles. It might contain:

Bus, Car, Train, Tram

The first task is to think how these might be divided up into categories. There are any number of possible ways to classify any collection of concepts, but we need to pick one to start with.

Let us say that we choose to classify them by whether they run on rail or road.

Then we might produce a hierarchy that looked like:

RoadVehicle
    Car
    Bus
RailVehicle
    Train
    Tram

We usually like to have a general abstract concept for any part of our terminological model or 'ontology'. We said at the beginning, that these were all kinds of 'vehicles' so we might add to this and rename it to get:

Vehicle
    RoadVehicle
        Car
        Bus
    RailVehicle
        Train
        Tram

By convention in GRAIL, everything which is about the topic or ‘domain’ we are considering is placed under DomainCategory, and DomainCategory comes under TopCategory which is built into the system. It is a good idea to separate out branches of an ‘ontology’ so we shall create an extra TransportCategory in between DomainCategory and Vehicle, So what we want to end up with at the end of the first stage is a hierarchy that looks like:

TopCategory
DomainCategory
TransportCategory
Vehicle
RoadVehicle
Car
Bus
RailVehicle
Train
Tram

3.1.2 Doing it in GRAIL

We are going to build an entirely new model, so the first thing to do is to clear any existing model, as described in Section 2.3.2.

Model resetting can also be useful if the model ever gets into a strange state while you're experimenting with the tools and model. Simply reset {xe "reset "}it to throw it all away and start again.

Now create a new file for the statements you will use to create the model by opening a file editor (as described in Section 2.3.3), and giving a new filename when asked. Note that the newly created file will contain the words *–new file* --. These should be deleted before you start adding any GRAIL statements.

3.1.2.1 Creating elementary categories

In an empty model, there are a few built-in, primitive categories. The first is TopThing{xe "TopThing"}, the root of the model, or network. Every concept must have at least one parent - except for the built-in starting pint, TopThing. Any other category is meaningless until it is connected to the model. TopCategory is a built-in child of TopThing to act as the starting point for all new categories

Type the following statements

TopCategory newSub DomainCategory.

DomainCategory newSub TransportCategory.

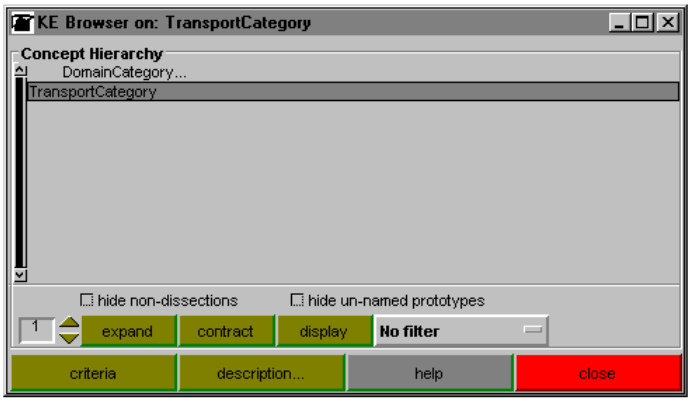
Do not forget the full stops at the end of each statement. Upper and lower case are significant, for example “Topcategory” is different from “TopCategory” and will not work.

Now select these two statements with the left mouse button and then use the middle mouse button to select **evaluate it** for the selection; this compiled these two expressions.

The operation newSub allows the creation of an “elementary concept”; it is given an identifier, and specifies at the same time where it appears in the concept structure hierarchy. After it has been defined in this way, it can be accessed by its identifier. We've created two new categories in this way - DomainCategory and TransportCategory. TransportCategory will be our starting point for the example. DomainCategory is a standard GRAIL convention which simply used for organisation[41]. By convention, the highest proper category in any GRAIL model is DomainCategory.

Now select TransportCategory in the editor and **GRAIL browse** from the middle mouse button menu. A browser should open which looks like:

[41] ‘Domain’ is used in the AI sense of the field which we are modelling, in this case ‘transport’ or ‘vehicles’. There is a second meaning as in ‘the domain of a relation’ which is entirely distinct and not relevant at this point.



3.1.2.2 Subsumption

Subsumption {xe "Subsumption "}is a posh word for the "is-a" relation. If B is a kind of A, then we say "A subsumes B", and "B is subsumed by A". As an informal example (which we'll see more formally later), Man is a kind of Person, so we can say that the concept of Man is subsumed by Person. It's a good idea to have fixed in your mind which way this relationship goes, as it sometimes seems that intuitively it should be in the other direction[42]. One way to remember it is that if B is a kind of A and sits "below" A in the hierarchy, then we will know more of the properties of B. Thus the properties of A are a *subset* of those of B and A *subsumes* B. There are two different forms of subsumption. One, *asserted* subsumption, we'll meet now; the other, *formal* subsumption, will explain up later.

Add an category that is subsumed by TransportCategory and embodies the concept of Vehicle.

TransportCategory newSub {xe "newSub "}Vehicle.

This is **asserted subsumption{xe "asserted subsumption"}**.. All we are saying is that there is some concept - Vehicle - that is "a kind of" TransportCategory. Note that the only interpretation of this relationship comes from *our own* linguistic interpretation of what Vehicle is except that it is a kind of TransportCategory. To illustrate this further, let's extend the model to a few specific types of vehicles.

Type in and evaluate the following:

```
Vehicle newSub RoadVehicle.
        RoadVehicle newSub Bus.
        RoadVehicle newSub Car.

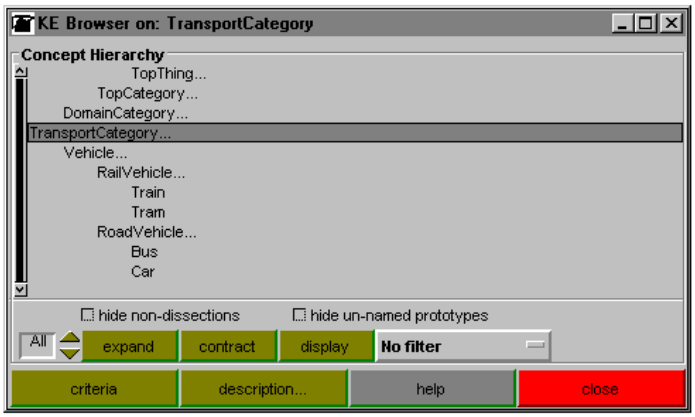
Vehicle newSub  RailVehicle.
        RailVehicle newSub Train.
        RailVehicle newSub Tram.
```

There are several points to note about the above expressions. Layout is ignored by the compiler. However, it is important to others trying to read your files. Good layout can help a great deal in clarifying awkward situations. In general, it is easier to read if you indent each level in the newSub statements is indented as shown.

Open a browser on TransportCategory by highlighting it and selecting GRAIL browse from the middle mouse-button menu. Click on Vehicle in the browser and then use the **expand** button on the browser to see what it subsumes. Click on and expand each of the subsumed categories in turn (as a short cut, use the menu option **show all levels**). You should see a browser as shown in below.

---

[42] Some object oriented tools use the phrase "is a generalisation of" or "generalises to" or just "generalisation" for "subsumes".



This is the 'asserted subsumption hierarchy' or 'asserted taxonomy' for your model so far. It is called the 'asserted subsumption hierarchy'[43] because all of the subsumptions have been created by asserting them with a newSub statement without any reason being given. In the next section we shall see how to create additional 'formal subsumptions'.

3.2 Formal Subsumption: Statements, Definitions, and Composite Categories

To explain the next steps requires adding descriptive information and definitions to our simple model. To do so, we will take a small detour and build a small model of kinds of people before we come back to relate people and vehicles.

3.2.1 Creating attributes

Attributes {xe "**Attributes** "**}**are used to describe relationships between categories, and define new categories. First add a new category to represent the concept of a Person:

```
DomainCategory newSub LivingThing.
LivingThing newSub Person.
```

Attributes will allow us to talk about, for instance, a Person that has sex male - a Man in other words. For that particular example, we *could* consider simply providing two new subsumptions of Person, Man and Woman. However, the use of attributes provides us with more power. As an illustration of this, consider the concept of people who drive buses. We could provide a new subsumption of Person, CarOwner, which would embody this concept. What if we then want to look at the relationship from the "other end"? We would have to also provide a new subsumption of Car, Car which isOwnedBy Person. One can easily see that this could lead to a proliferation of new categories which have to be added explicitly. In addition, this fails to capture the fact that we may also want to talk about Vehicle drivers in general, as they might all have something in common.

What we want to do is provide a mechanism for *defining* owners, men, women, etc. systematically. This also has the advantage that we know why a man is different from a women, at least in our model: because men and women have different sex. Likewise we will know what makes owners different from other persons: they own something.

First, some house-keeping. (The syntax in this section is not particularly nice, so you will just have to bear with us.) Type the following into a workspace, select it and evaluate it.

```
Attribute newAttribute DomainAttribute inverseDomainAttribute allAll manyMany.
```

---

[43] In earlier versions of the documentation, this was termed 'explicit subsumption', but this terminology has been found confusing.

Attribute is the top of the attribute hierarchy and comes built-in in every new model. (Perhaps we should have called it TopAttribute, but we didn't.) The expression which you typed introduces a new attribute DomainAttribute which will be used in a similar way as DomainCategory, as a top-most attribute to hang all our other attributes from.

Now let's introduce a more useful attribute:

DomainAttribute newAttribute hasOwner isOwnerOf allAll manyOne.

This introduces a new attribute hasOwner. It has an "inverse{xe "inverse"}" isOwnerOf which will allow us to talk about the relationship from the "other end". The keyword allAll is obsolete but persists in our compilers. The keyword manyOne says that a vehicle can have only one owner, but one owner can own many vehicles. This is referred to as the **cardinality {xe "cardinality "}**of the attribute. Other cardinalities are manyMany oneMany oneOne. Another way to describe attributes with cardinality manyOne or oneOne is to say that they are 'single-valued'[44]

3.2.2 Sanctioning statements

So far, we've only considered **elementary categories{xe "elementary categories"}** - those categories that we've explicitly defined and placed in the hierarchy. Using attributes, such as hasOwner declared above, we can construct **composite categories{xe "composite categories"}.** However, we can't just make composite categories at will: asserting that a Person is owned by a Bus is obviously nonsensical and should be prevented! Before it can be used in a definition, an attribute has to be **sanctioned{xe "sanctioned"}**. Sanctioning {xe "Sanctioning "}is the mechanism that describes how categories can be put together with attributes to make definitions, and thus specify what it is sensible and meaningful to say.

In GRAIL, there are two levels of sanctioning statements (plus a third level of 'necessary' statements which we will describe later):

- Grammatical{xe "Grammatical"}
- Sensible{xe "Sensible"}

To sanction the hasOwner relation at the grammatical level, we create a new **statement {**xe "**statement** "**}**by evaluating the following statement:

Vehicle grammatically hasOwner LivingThing.

This statement says that in a grammatical sense, a Vehicle can have a owner, which is a LivingThing, associated with it. Roughly speaking, this means that it would not be crazy to ask a question about any sort of living thing owning a vehicle, but it may not make much sense or be the sort of thing we would want to turn up in a list of choices in a user interface. We can ask questions about toads who own cars, but unless we are modelling some sort of cartoon world, it isn't very 'sensible'.

So what have we done? We've expressed the fact that in *general*, it's not crazy to talk about vehicles having owners. Not all vehicles will have owners, and not all living things will be owners of vehicles - the further levels of sanctioning will refine these constraints.

RoadVehicle sensibly hasOwner Person.

This statement tells us that for RoadVehicles, it's sensible to talk about them having people as owners.

---

[44] Note that in the description logic community, the word 'Role' is used for what GRAIL calls 'attributes' and the word 'Attribute' in the DL community is reserved for single-valued roles., i.e. what GRAIL terms 'single valued attributes'. Confused? Sorry about that, but the vocabularies have grown up independently and confusingly.

In order for a 'sensible' statement to be valid, there must already exist an appropriate grammatical statement. For instance, if we had attempted to evaluate the sensible statement before the grammatical statement, an error would have occurred. Note also that categories inherit the sanctions from their parents. As RoadVehicle inherits from Vehicle, it also inherits the grammatical sanction of the hasDriver relation. Likewise Person inherits the sanction from LivingThing. This is an important point to remember.

Note that there can be more than one grammatical statement for the same attribute. It is also possible for grammatical and sensible statements to occur at the same level. For example, if we say

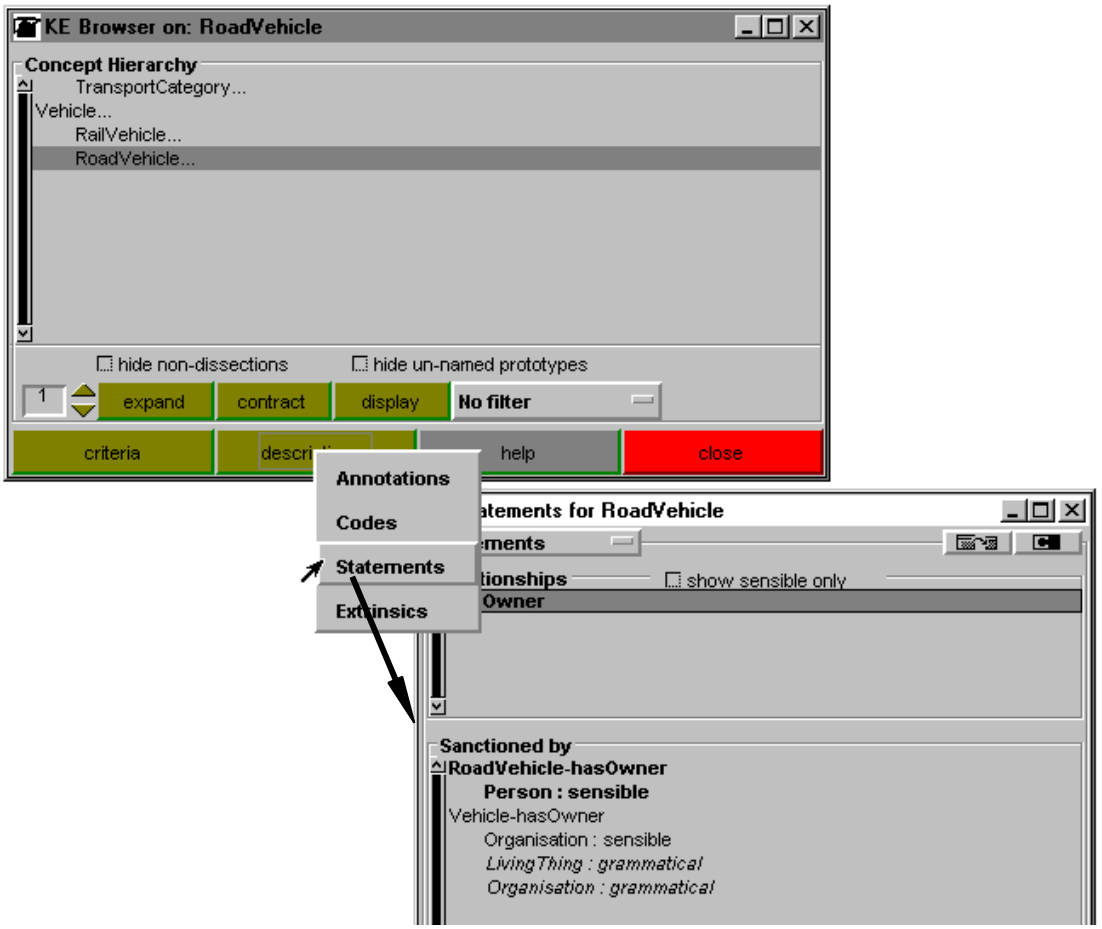
DomainCategory newSub Organisation.

Then we might want to add.

Vehicle grammatically hasOwner Organisation.

Vehicle sensibly hasOwner Organisation.

To view the statements that are in force for a particular concept, use the **Descriptions→ Statements** option from the Concept Browser, as shown below:



The display indicates that there is currently only one attribute sanctioned (hasOwner); bold-face indicates that the sanction has been applied at the browser focus (as opposed to being inherited). Italic font indicates grammatical rather than sensible sanction. Note that middle-button menu options are available to explore any of the categories displayed as parts of statements.

3.2.2.1 Form of GRAIL statements

All GRAIL statements follow the pattern above:

Topic *qualifierKeyWord* anAttribute Value.

The Topic and Value must be categories and anAttribute must be an attribute. The qualifier keywords grammatically and sensibly have been introduced so far. Other qualifier keywords will be introduced later.

In short, GRAIL statements are triples as in many other languages, plus a qualifier. GRAIL uses the word 'topic' rather than 'object' (as in 'object-attribute-value' triple) in order to avoid confusion with the many other uses of the word 'object'in current circulation.

3.2.3 Defining Composite Categories

Our model now has some elementary categories and an attribute which can be used to construct composite categories. To form composite categories, we use the which {xe "which "}operator[45]. For example, type the expression below, select it, and browse it by clicking **GRAIL browse** on the middle mouse-button menu.

Car which hasOwner Person.

Note the GRAIL browse option evaluates (compiles) the selection before opening a browser, so there is no need to evaluate it first (though it doesn't matter if you do). The first thing to note here is that this expression, unlike all the others we've met so far, does not *add* any knowledge to the model. Instead, it is a *query* which asks the terminology engine to return the category corresponding to the expression.

In order for the expression to be valid, and the evaluation to succeed, the relation must have been sanctioned at the sensible level. Recall that this sanction could come from a parent - in fact in this case it does, being inherited from the RoadVehicle category. To illustrate this you could try asking for, say, a Train which has an owner.

Notice that the object we asked for has been put in the appropriate place within the hierarchy. It's a child of Car, which is itself a child of RoadVehicle (the hierarchy can be seen by using the **Expand** feature of the browser).

Although the attribute hasDriver has an 'inverse' isDriverOf, it is important to remember that the categories they allow us to construct are different. If we ask for the category:

Person which isOwnerOf Car.

this is very different from a Car which hasOwner Person. The first expression provides us with a kind of Car, the second provides us with a kind of Person. People and cars are very different kinds of things: people are not kinds of cars nor are cars kinds of people.

3.2.4 Cardinality

In section 3.2.1, we briefly mentioned cardinality {xe "cardinality "}- the manyOne argument to the operation that created the hasDriver attribute. Now that we have introduced sanctioning and prototypes, we can describe cardinality properly.

The cardinality of an attribute specifies *how many* categories can be connected by a given attribute. For example, we have now sanctioned the hasOwner attribute to connect Vehicles and

[45] In the past, 'composite entities' have been called 'prototypes' and 'particularisations'. Both phrases caused confusion and are being dropped. However, some documentation still uses the old vocabulary.

Persons[46]. The cardinality we used was manyOne - meaning that any Vehicle has only one Owner, but that a Person could own many Vehicles. This mechanism allows us greater control on which composite categories are allowed; for example, evaluating the following expression will produce an error[47]:

Car which <hasOwner Man hasOwner Woman>

because we are asking for a kind of Car that has two owners. We have disallowed this with our choice of cardinality for the hasOwner attribute.

The other possible values for cardinality are (with their effect):

oneMany	this would mean that Vehicles could have many kinds of Persons as drivers, but a Person could only drive one sort of Vehicle;
oneOne	this would mean that a Vehicle could only have one kind of Person as a driver, and that a Person can only drive one kind of Vehicle;
manyMany	this would mean that Vehicles could have many kinds of Persons as drivers, and that Persons could be drivers of many kinds of Vehicles.

3.2.5 Criteria

A **criterion {**xe "**criterion** "**}**is the name given to a combination of an attribute and an category. For instance,

hasOwner Person

Criteria are the building blocks that are used to construct composite categories from elementary ones.

3.2.6 Formal Subsumption

We've already met asserted subsumption, which is where the subsumption is asserted using newSub. For instance, a Car is a kind of RoadVehicle. The other type of subsumption is what is known as **formal subsumption{xe "formal subsumption"}**.. This is where the relationship arises due to the structure of categories and is derived by the internal semantics of GRAIL.

In the earlier example, the category (Person which isDriverOf Car) is formally subsumed by (remember the direction!) Person.

One rule about formal subsumptions is that: any category X always subsumes any category of the form X which hasSomeAttribute Y.

There are other rules which define formal subsumption. We'll not go into detail here, but will return to this topic once we have introduced a few more attributes.

3.2.7 Value Types

So far, we've been dealing with fairly simple basic categories and combinations of those. Next, we'll go on to look at value types. These are again categories that sit in the hierarchy, but we separate them out for convenience in a special section of the model subsumed under a special built-in category, SymbolicValueType.

---

[46] At present, cardinality is specified when creating an attribute; in the future, we may extend GRAIL to allow different cardinalities for particular sanctions.

[47] We haven't yet defined Man and Woman - try the evaluation of this expression after evaluating the definitions which appear later.

Introduce a new Value Type{xe "Value Type"} to represent gender, along with a couple of subsumptions[48]:

```
SymbolicValueType newSub SexValueType.
  SexValueType newSub male.
  SexValueType newSub female.
```

The categories, male and female have been given lower case initial letters - this is simply a convention which helps to distinguish between value types and other categories.

Now, a new attribute:

```
DomainAttribute newAttribute hasSex isSexOf
        allAll manyOne.
```

The sanctioning expression which describes the way this attribute can be combined with categories:

```
Person grammaticallyAndSensibly hasSex SexValueType.
```

We've introduced the statement at both the grammatical and sensible levels with one operation. This is simply a time saving measure, and has the same effect as if the statements were introduced individually, i.e. this is exactly the same as:

```
Person grammatically hasSex SexValueType.
Person sensibly hasSex SexValueType.
```

We can now evaluate the expression

```
Person which hasSex male.
```

Try it, and browse the category to see where it fits in the hierarchy. In a similar way, we'll introduce the concept of age:

```
SymbolicValueType newSub AgeValueType.
  AgeValueType newSub young.
  AgeValueType newSub old.

DomainAttribute newAttribute hasAge isAgeOf    allAll manyOne.

Person grammaticallyAndSensibly hasAge AgeValueType.
```

We can combine expressions

```
(Person which hasSex male) which hasAge young.
```

We can write this expression more simply using the <...> brackets to put both criteria in a criteria set.

```
Person which < hasSex male  hasAge young >.
```

We might translate this roughly into English as “Person which is male *and* young” which means the same thing as “Person which is male which is young”.

Similarly, we can form the composite category:

```
(Person which hasSex female) which hasAge old.
```

---

[48] Note that we could have abbreviated this to:

```
(SymbolicValueType newSub SexValueType)
        newSub [male female].
```

The parentheses () are used in a standard manner to group expressions. Note that the result of the expression (SymbolicValueType newSub SexValueType) is the category SexValueType to which we can use as the topic of the next newSub operation.

However, the cardinality of the hasSex attribute will prevent us doing anything silly. The expression:

(Person which hasSex male) which hasSex female.

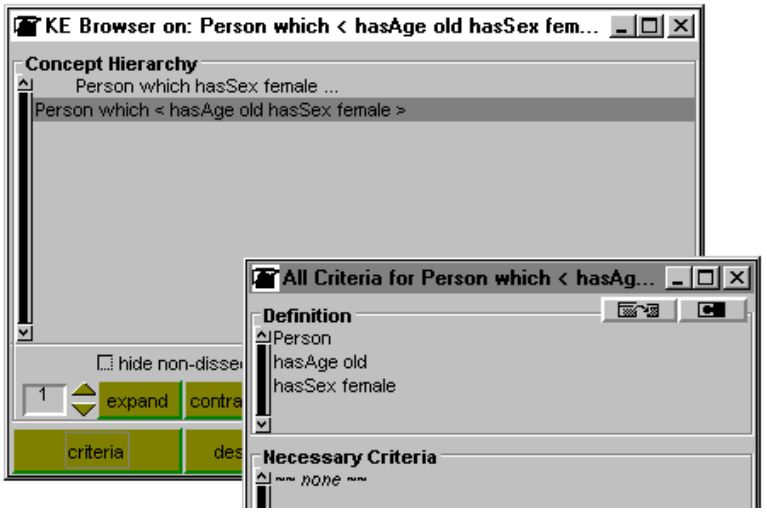
will produce an error if we try and evaluate it, due to the manyOne cardinality of hasSex.

3.2.8 Criteria

To examine the criteria of a category, browse again on

(Person which hasSex female) which hasAge old.

Now click on the **criteria** button on the browser, to open a display showing the structure of the category:



The criteria which form the definition of the concept are also known as its 'defining criteria'. The top line of the **Definition** pane shows the elementary category from which the category is formed (the 'base'), whilst the remaining entries show each of its defining criteria. (We will explain the lower two panes of this window later).

3.2.9 Naming

As we can already see in our little example, categories can become quite complex. For instance, we can now evaluate the expression:

```
Car which hasOwner
  ((Person which hasAge old)
        which hasSex male).
```

The concept of a car which is driven by an old man. In a more complex model, these expressions could quite easily become unwieldy and difficult to interpret. To help with this, GRAIL allows the naming or 'aliasing' of categories. For example:

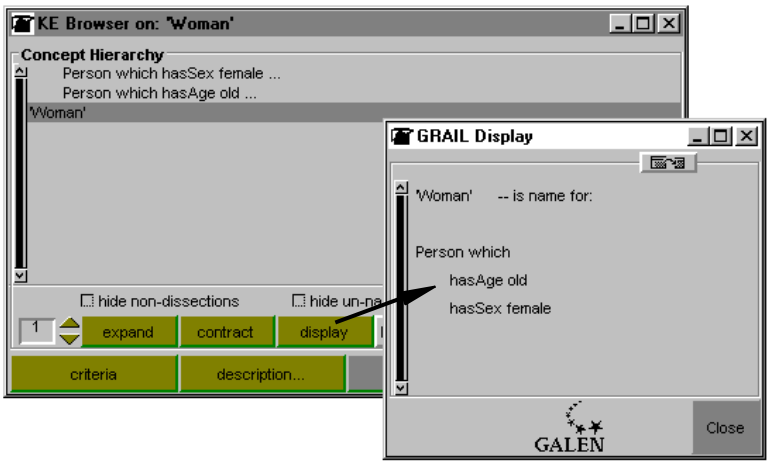
```
((Person which hasSex male) which hasAge young)
        name Boy.

((Person which hasSex female) which hasAge old)
        name Woman.
```

Now, we can simply use the names Boy, Woman, to represent, and to access the complex categories to which they refer.

To practice, add names for the concepts Man and Girl.

Now browse the category Woman and click again on the *criteria* button. Note that nothing has changed. Now click on the **Display** button to obtain another view of the structure of the browser's focus category:



Note that in the Concept Browser hierarchy display, Woman appears inside quotes: this is simply a display convention to indicate that it is a name for a complex category.

3.3 Necessary statements and Canonical form

3.3.1 Necessary statements

So far, we've seen the two kinds of sanctioning statement - grammatical and sensible. We now meet the third kind of statement, the **necessary** {xe "**necessary** "}statement. This is used for situations where a criterion is mandatory but not part of the definition. For instance, we may expect that all owners of road vehicles must be old (for the purposes of our example, 'old' is considered to be 'old enough to drive' - this does not necessarily reflect the actual views of the management!), We can incorporate this into the GRAIL model as follows:

(Person which isOwnerOf RoadVehicle) necessarily hasAge old.

Note that the category on the left is a complex category - previously all our statements involved basic categories. A statement can have any valid category expression on the left or right hand side. Now try browsing the expression:

(Person which isOwnerOf RoadVehicle).

Click on the **criteria** button. This time, in the **Definition** pane you'll notice the criterion isOwnerOf RoadVehicle. However, in the **necessary criteria** pane, the criterion hasAge old will also be present, telling us the extra information that we've just added to the model. The lowest pane (**inherited criteria**) shows any necessary criteria that the browser's focus may have inherited from its parents, illustrated in the next section.

If we now attempt to evaluate:

Girl which isOwnerOf Car.

The evaluation will fail as the criteria hasAge old and hasAge young would both be present in the result of the expression and are conflicting. Recall that hasAge has cardinality manyOne.

3.3.2 Canonical Forms

Consider the expression:

Man which hasSex male.

This is a well formed[49] and sensible expression: a Man is a (kind of) Person, and can thus be related to a SexValueType via hasSex. In addition, there is no cardinality problem as we're not asserting that a Person has two different sexes. However, we can see that the criterion hasSex male is redundant as we already know that about a Man because hasSex male is part of the definition of the category Man.

The terminology engine is able to reason in this way as well, and reduces categories to a **canonical form{xe "canonical form"}**. This consists of a basic category and a collection of criteria with any 'unnecessary' criteria like those above removed.

Try browsing the expression:

Man which hasSex male.

You will see that the browser opens simply on the category, Man.

3.3.2.1 Necessary statements and canonical forms.

Consider the naming:

(Person which isOwnerOf Car) name CarOwner.

If we now look at the categories:

CarOwner.

CarOwner which hasAge old.

They will be the same. CarOwner already has the criterion hasAge old due to the necessary statement introduced earlier, so the system has reduced them to the same category.

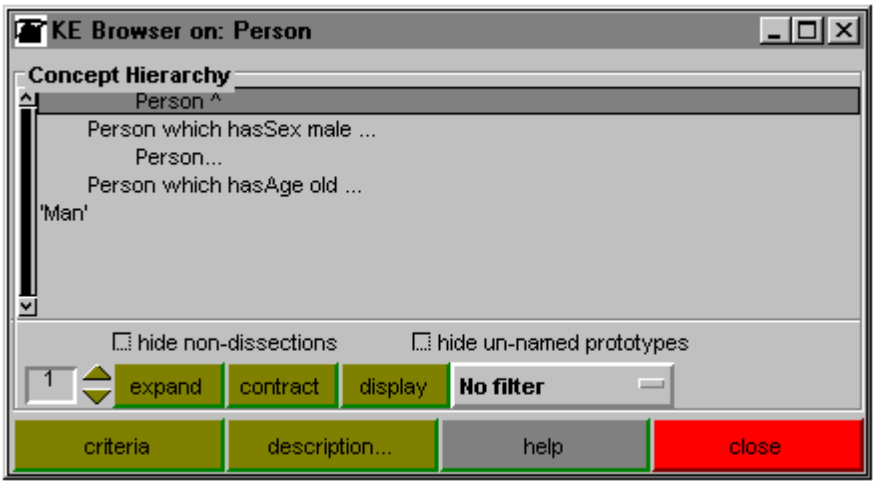
Click on the **criteria** button. The criterion hasAge old appears in the **inherited criteria** pane because it is inherited from one of its ancestors in the subsumption hierarchy: Person which isOwnerOf RoadVehicle.

To summarise, a canonical form is a starting elementary category known as the 'base' along with a collection of criteria in which all redundant information has been removed. If the GRAIL browser is invoked on a which expression, the compiler will reduce it to canonical form before opening the browser. To see the detailed criteria click on the **criteria** button. The criteria are divided into three panes: the defining criteria; the necessary criteria which result from necessary statements about this specific category; and the inherited criteria which result from necessary statements, or definitions of categories, further up the hierarchy.

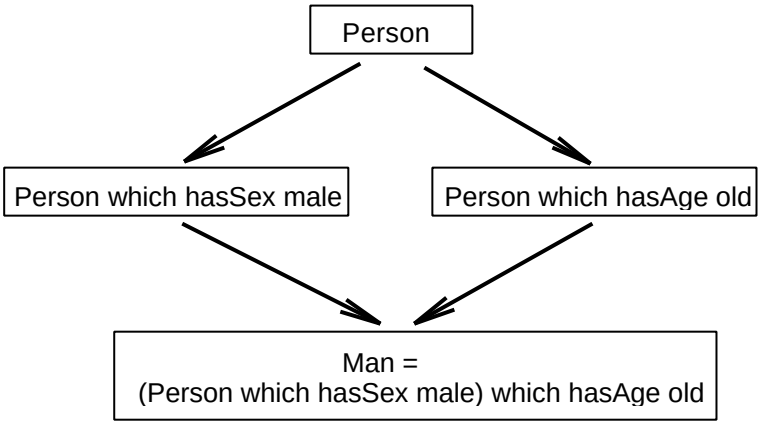
3.3.3 The Hierarchy

At this point, we'll take another look at the hierarchy **{**xe "hierarchy "**}**we've constructed. We'll only show parts of it, as the network is now quite large.

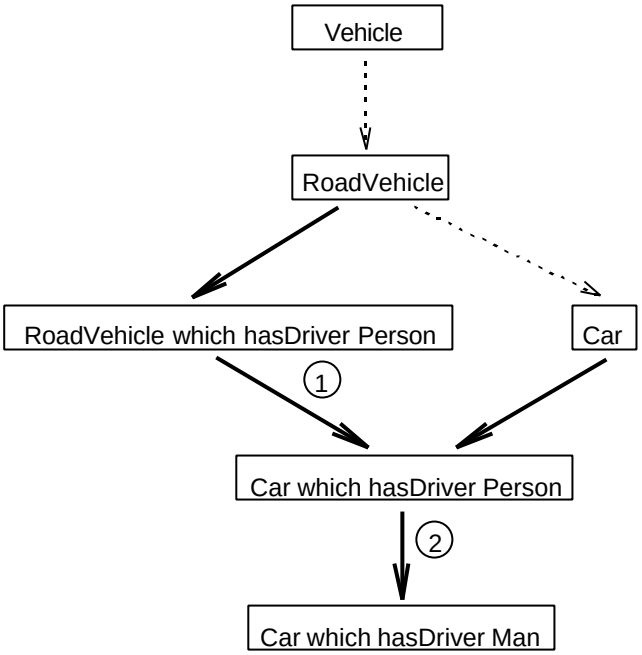
[49] By well-formed, we mean that the expression is syntactically valid, and will be parsed by the compiler.



The browser above shows us some of the position of Man in the hierarchy. Note that all the subsumptions shown above are formal: they are derived from the definition of the Man category. The browser is displaying the following subsumption relationships:



Formal subsumption for composite concepts is essentially a recursive notion using subsumptions of the constituent categories. In the following diagram, illustrating a more complex area of the hierarchy, Subsumption 1 illustrates a subsumption where the 'base' categories (RoadVehicle and Car) are related, while 2 illustrates the situation where the categories in the criteria (Person and Man) are related. The dotted arrows indicate asserted subsumptions, whereas the solid arrows indicate formal subsumption relationships that have been derived by the GRAIL engine.



3.4 Further features

3.4.1 Grammar level Queries (Grammar level Composite Categories)[50]

The which operator allows us to build up composite categories which have been sanctioned at the sensible level. Often we want to ask queries which are more general than are allowed by the sensible level sanctions. For example, we only have sanctions to say that a RoadVehicle can sensibly be owned by a Person. This says nothing about a Vehicle in general. (We might, for example, want to build a model in which RailVehicles could only sensibly be owned by companies.) However, it is perfectly reasonable to want to find all of the vehicles which are owned by men.

However, there is a grammatical sanction allowing us to talk about a Vehicle owned by any LivingThing, or conversely a LivingThing which owns a vehicle. To ask a general query allowed by this grammatical sanction, we use the whichG operator **{**xe "whichG "**}** (G for 'grammatical' categories). **{**xe "*grammatical prototypes*"**} {**xe "grammatical query"**}{**xe "grammatical composite category"**}**

We can browse:

Person whichG isOwnerOf Vehicle.

This represents the concept of a Person who owns a Vehicle (remember that this may not be sensible if the Vehicle is, say, a Train or a StealthBomber).

Note that Person which isOwnerOf RoadVehicle is classified automatically under Person whichG isOwnerOf Vehicle. Note the three dots following Person whichG isOwnerOf Vehicle (indicating that it has children). Select it and expand it, and underneath will be 'CarOwner'

[50] For short these are sometimes called 'Grammatical Categories' or for historical reasons 'Grammatical prototypes'.

which will have been classified automatically and put in the correct place in the hierarchy according to the rules for formal subsumption. (Remember, the quote marks indicate that this is a 'named' concept. Click on the **display** button to see the definition or the **criteria** button to see the full list of criteria).

Grammar level composite categories exist primarily to be used as a query mechanism as shown above. They can also be used for adding general knowledge, but should be used with caution. As an example of using grammar level composite categories for simple rules, consider the idea of insurance ratings for owners of Vehicles:

DomainAttribute newAttribute hasInsuranceRating isRatingOf
allAll manyOne.

(Person whichG isOwnerOf Vehicle) grammaticallyAndSensibly
hasInsuranceRating IntegerValueType.

We don't want just to have a rating for owners of vehicles on the roads, but for all vehicle owners - even owners of kinds of vehicles we have not yet thought of. This statement guarantees that when we do think of new kinds of vehicles and extend the model to include them, then the sanctioning for having an insurance rating will also apply to owners of these as-yet-un-thought-of kinds of vehicles.

3.4.2 Topic and Value necessary statements

We introduced the necessary qualifier earlier to assert necessary, or essential, facts about categories. For example, in the assertion:

(Person which isOwnerOf RoadVehicle) necessarily hasAge old.

we are saying that anyone who drives a road vehicle must be old. If we look at such an category with the GRAIL browser, the criteria hasAge old will appear in the necessary pane.

What is not obvious, is that the 'inverse' attribute will also have been qualified in this way. If we browse the category old, the criteria isAgeOf (Person which is isOwnerOf RoadVehicle) will be present in the necessary criteria pane. We may not always wish this to happen - some extra classification information has been added to the right hand category of the statement. GRAIL provided two special operations topicNecessary {xe "topicNecessary "}and valueNecessary {xe "valueNecessary "}which allow us to specify 'one-way' necessary qualification. Thus we can assert[51]:

(Person which isOwnerOf RoadVehicle) topicNecessarily hasAge old.

This would place the necessary qualifier on the left hand category but not the right hand. A similar effect could be achieved by asserting:

old valueNecessarily isAgeOf (Person which isOwnerOf RoadVehicle).

As with the necessary qualifier, the relation must be sanctioned to the sensible level in order for the statement to be valid

The keywords topicNecessary and valueNecessary come from the names for the different parts of a GRAIL statement:

Topic qualifierKeyword Attribute Value

[51] Note that as we have already asserted the necessary statement for this entity, the statement will have no effect. To see the effect of the one-way necessary try recompiling the model to this point then using topicNecessarily rather than necessary.

3.5 Some common questions

3.5.1 What is the difference between a criterion in a definition (a which expression) and a criterion in a necessary statement?

The difference is subtle but important. A which expression defines a new category. A necessary statement adds new information to an existing category. Put another way, the criteria in a which expression are sufficient to recognise a category, and the category cannot be recognised unless all of the criteria in the which expression are found. By contrast, a category can be recognised even though the information in a necessary statement is unknown, but it is added as soon as the category is recognised.

To take our example, we have said that CarOwners are necessarily old. We can recognise a CarOwner just because they own a car, but as soon as we recognise them, we know that they must be old (or so says this model). If we find something which claims to be both a CarOwner and young, the system will attempt to add that the thing is old because of the necessary statement. This will cause an inconsistency and the system will report an error.

In summary, we can recognise something even if we are ignorant of the information in the necessary statements, but if we have information which contradicts the information in the necessary statements we have an inconsistency.

> **Side-bar for the logically minded** The conjunction of all of the criteria in the definition is true if and only if the category defined is true, *i.e.* the definition is necessary and sufficient for the category defined. Each criterion in a necessary statement is true if the category is true, but there is no implication in the other direction, *i.e.*. each criterion in a necessary statement is individually necessary but not sufficient for the category.

Still confused? Again, bear with us until the end of the tutorial.

3.5.2 What is the model?

There are two things which are often referred to as 'the model{xe "the model"}'. These are:

a) The textual GRAIL statements containing the information presented by the modeller;

b) All the concepts and facts implied by that information and the GRAIL rules compiled into a form that they can be manipulated by the GRAIL engine.

In general it is more convenient to speak of a), the text files, as the 'source files' for the model and b) as the 'compiled model'. Exactly how the compiled model is represented, loaded, and saved depends on the implementation[52].

You look at the source files using a file editor, file list or workspace. (The source files are simply text files, and you can, if you wish, edit them with any text editor.) You look at the compiled model using the browser and other GRAIL tools. Information only gets from the source files to the compiled model when you evaluate an expression or expressions in the source files. (There are also tools to let you evaluate an entire source file or an entire set of source files with a single statement, but we will come to those later.)

The GRAIL engine only makes composite categories when it is asked to, since there are a nearly infinite number of possible composite categories which it could make. If you just look at the model using the browser, you will only find those composite categories which have previously been mentioned or requested by a which expression. In the example, we have never mentioned

[52] With the version of the software you are using, the compiled model is simply stored as an internal structure of Smalltalk objects, whereas in other implementations it may be stored externally in a database.

or requested the category (Car which hasOwner Woman). As a result, we will not find it if we look for it by starting with Car in the browser and then expanding to look at all of the kinds of car in the model. However, if we were to evaluate the expression (Car which hasOwner Woman), the GRAIL engine would immediately make the category and classify it in the correct place in the hierarchy.

4 Using GRAIL well: A tutorial in basic GRAIL style

4.1 Re-use, application independence, and modelling style

Chapter 3 should have given you understanding of the basics of building GRAIL models. It has introduced all of the basic ideas except for the way in which GRAIL handles transitive relationships (this is covered in section 5). This chapter explains in more detail how GRAIL is intended to be used. GRAIL has been designed to make a certain kind of modelling possible. To get the most out of GRAIL, you need to follow a few simple rules to produce the sort of models intended.

Our goal is to produce terminological models which are 'application independent' and 're-usable'. To do this, we want to make as much information explicit in the model as possible so that applications can use it, and we want to make as few decisions as possible on the basis of individual applications. We want to avoid 'building in' a bias towards one particular way of looking at the model.

Of course this can never be perfect, and no model will ever be re-usable over the entire range of human experience. However, our goal is to produce models which cover a range of applications in a given area rather than models which are tuned just to just one, very specific application.

The main strategy for producing application independent models is to break our hierarchies of elementary categories into independent sub-hierarchies which do not overlap and then to fit them together using definitions and necessary statements. The result of the classification and formal subsumption may be a very complex multiple hierarchy, but we know why each category is classified the way it is because the information is explicit in the definition and necessary statements.

We have to have some asserted subsumption: we have to start somewhere. But over-use of asserted subsumption is the enemy of re-use, because the reason for asserted subsumption is 'tacit' and invisible to the computer. Personally, we may or may not know or remember why we asserted that a given elementary category should be classified in a certain way, but it is certain that the information is *not* available to the system or to applications. We want to build the minimum of tacit assumptions into the asserted hierarchies. We want to make as much as possible explicit through the use of definitions and necessary statements.

4.2 The example model revisited

In the example we chose to classify vehicles into road vehicles and rail vehicles. But this is only one of many different ways of classifying vehicles. For example we might have been interested

in what fuel they use, whether they have wheels, or whether they are used for transporting people or freight.

The choice of what criteria to use for the asserted taxonomy is entirely arbitrary, but it should be consistent. Mixing different principles in the asserted hierarchy means that it is impossible to determine why something is classified in a given place.

4.3 Extending the model to a second classification

Let us consider that we also want to classify vehicles by the type of fuel they use.

Then the best way to do this is in three steps:

i) Create a hierarchy of kinds of fuel and an attribute linking that hierarchy to vehicles;

ii) Use necessary statements to describe which kinds of vehicle use which kinds of fuel;

iii) Create the general categories such as 'petrol driven vehicle' as composite concepts using which or whichG expressions and giving them convenient names.

To illustrate, create a hierarchy of kinds of fuel by typing the following into the file editor and evaluating it. (Anything between "..." is a comment and is ignored by the compiler.)

"Step i)"

```
DomainCategory newSub Fuel.
  Fuel newSub Petrol.
  Fuel newSub Diesel.
  Fuel newSub Electricity.
```

"Step 2)"

```
Attribute newAttribute hasFuel isFuelOf allAll manyOne.

Vehicle grammaticallyAndSensibly hasFuel Fuel.

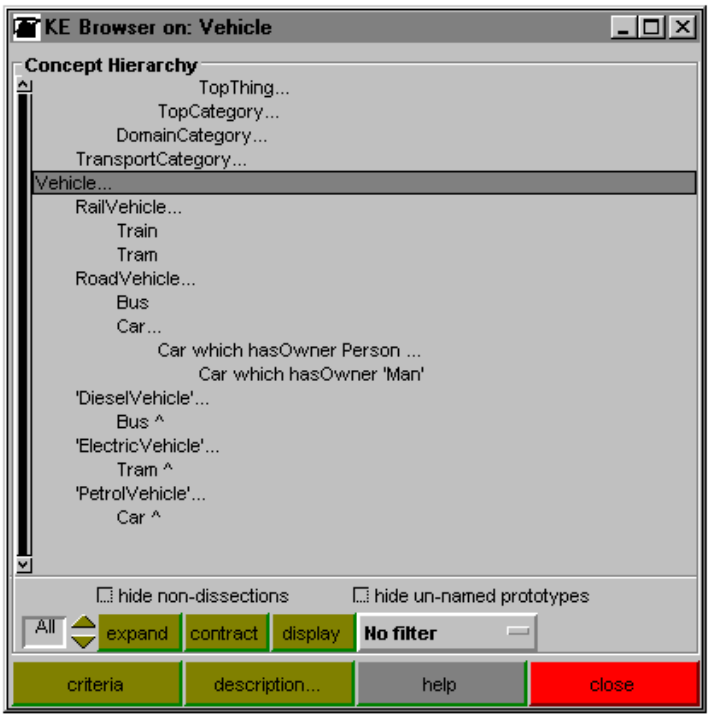
Car topicNecessarily hasFuel Petrol.
Bus topicNecessarily hasFuel Diesel.
Tram topicNecessarily hasFuel Electricity.
```

"Step 3)"

```
(Vehicle which hasFuel Petrol) name PetrolVehicle.
(Vehicle which hasFuel Diesel) name DieselVehicle.
(Vehicle which hasFuel Electricity) name ElectricVehicle.
```

Now browse Car. Note that Car is now classified under both PetrolVehicle and RoadVehicle. We often say that it has two 'parents'.

Also browse DieselVehicle. Finally, browse Vehicle and use the middle mouse-button to **show all levels,** as illustrated below:



(The carets '^' beside categories mean that they have already appeared further up the hierarchy and are visible on this screen.).

Notice that only Train has not been multiply classified because we have not said anything further about trains, because trains can be fuelled by either Diesel or Electricity or Petrol. (We have simplified the world by saying that all buses use diesel and all cars petrol, but that is convenient for the illustration.)

Note therefore that what we have found is not all of the vehicles which *might* use a particular fuel - that list could be very long indeed, but all of the vehicles which anybody has thought it worth modelling which use a particular fuel.

If we decide it is worth modelling diesel and electric trains, we can add the statements.

(Train which hasFuel Diesel) name DieselTrain.
(Train which hasFuel Electricity) name ElectricTrain.

Then DieselTrain and ElectricTrain will be multiply classified as required.

4.4 Reformulating the model

Browse Car again and look at the criteria by pressing the **criteria** button. Note that the criteria tell you that cars run on petrol but not that they are road vehicles or anything about what it means to be a road vehicle. That information is implicit in the hierarchy.

It would probably be better to rethink the model to make both pieces of information explicit. (Don't be afraid to rethink models and rebuild them from scratch. Most models have been rebuilt from scratch several, sometimes dozens of, times. You won't get it right the first time).

Let's reformulate the model so that both the fuel and whether it runs on rail or road are of equal status.

What we want to do:

i) Remove RoadVehicle and RailVehicle from the asserted hierarchy.

ii) Define attributes and values so that we can describe a vehicle running on road or rail.

iii) Describe each vehicle using the attribute as running either on road or rail using necessary statements.

iv) Define the idea of RoadVehicle and RailVehicle in terms of those attributes and values using which expressions.

First reset the model so you get rid of all the existing definitions as described in section 2.3.2.

Now make a new copy of the file you are using. (Sometimes it is easier to start from scratch, but in this case we can probably manage to keep most of what we have already done.) First, if you have not done so already, save the file in which you have been working by selecting the **save** option on the middle-mouse button. Then use the **save-as** option to save it to a new file with a different name.

4.4.1 Removing RoadVehicles and RailVehicles (flattening the asserted hierarchy)

First, since are going to use formal subsumption instead of asserted subsumption to classify the different kinds of vehicles, then we need to remove the elementary categories RoadVehicle and RailVehicle from the asserted hierarchy.

Edit the initial commands in the file until they look as shown below. (The cut and paste keys can make this a quick operation.)

```
TransportCategory newSub Vehicle.
        Vehicle newSub Bus.
        Vehicle newSub Car.
        Vehicle newSub Train.
        Vehicle newSub Tram.
```

4.4.2 Creating the attributes and values

Then let us create the attributes and values needed. We need something to call the attribute which describes whether a vehicle runs on rail or road. It is often not easy to come up with a name. We shall use travelsOn for the attribute and TravelMediumType for the rail, road etc. (We might extend this to air at the same time with little effort.)

Add to the file immediately after the asserted hierarchy a new attribute definition, a new value type, and the appropriate sanctions:

```
DomainAttribute newAttribute travelsOn isTraveledOnBy allAll manyOne.

SymbolicValueType newSub TravelMediumType
  TravelMediumType newSub air.
  TravelMediumType newSub rail.
  TravelMediumType newSub road.

Vehicle grammaticallyAndSensibly travelsOn TravelMediumType.
```

4.4.3 Describing the elementary categories.

Next we need to is add the descriptions of the different travel mediums. Immediately after the above add:

```
Car     topicNecessarily    travelsOn road.
Bus     topicNecessarily    travelsOn road.
Train   topicNecessarily    travelsOn rail.
Tram    topicNecessarily    travelsOn rail.
```

4.4.4 Creating the abstractions

Now all we have to do is define RoadVehicle and RailVehicle, but this time they are formal definitions so the model contains the explicit information on what they mean. Immediately after the above add:

```
(Vehicle which travelsOn Road)      name RoadVehicle.
(Vehicle which travelsOn Rail)      name RailVehicle.
```

4.4.5 Sorting out the order of statements in the source file.

Everything in GRAIL must be defined before it is used. This is one of its strengths and one of its weaknesses, but it often makes modifying files a bit painful.

Find the first point in the file where you have first used RoadVehicle or RailVehicle (after their introduction). It should be the statement

```
RoadVehicle sensibly hasOwner Person.
```

The new definitions must go before this point.

Since the new definitions are replacing some of the things which were in the original asserted hierarchy, it is good practice to place them immediately after the asserted hierarchy. This is what we have done in the example.

However, this still leaves a problem, because the definition of DomainAttribute is further down the file. (You will find this out if you do a **check syntax** on the file because you will get an error message).

Move the statement defining DomainAttribute up until it comes after the asserted hierarchy and before the new section you have just added.

(Good style organises source files in a sensible order so this happens as rarely as possible. More about this in later volumes. But for the tutorial, this is sufficient.)

4.4.6 Checking the syntax

Select the entire file (either by clicking three times in the upper left hand corner, or by extending a selection by clicking with the shift-key held down). Choose **check syntax** from the middle-mouse button menu. If you get errors it will be because either you have mis-typed something or that some statements are out of order so that some category or attribute is not defined before it is used.

It may take you several tries to get the syntax correct. Don't worry about it. GRAIL is an unforgiving language. That means that errors get detected early. It also means that it can take considerable effort before you have a clean file to compile.

(The syntax checker does not guarantee that a file compiles without errors. There can still be contradictions and other 'semantic' errors which can only be recognised when the expression is fully compiled. But it saves much time and often gives more helpful error messages.)

4.4.7 Checking the result

Once the syntax is checked, again select the entire file and **evaluate it**.

Browse Car and Vehicle again to be sure that you have the same overall result as previously. Check the criteria windows to see that you now have explicit criteria representing whether a vehicle travels on rails or road. Note that RoadVehicle and RailVehicle now appear in single quotes indicating that they are named composite categories rather than elementary categories.

4.5 Final Remarks on the Example

4.5.1 Further reformulation of the model? The need for an 'ontology'

Is this the ultimate way to formulate a terminological model of vehicles? Almost certainly not. Most obviously, the concepts labelled 'air', 'road', 'rail', etc. have been used in a very peculiar way. Almost anything else we might want to say about roads, rails, or air would not be sensibly said about a ValueType as used here.

To go further, we need a larger framework or 'ontology' into which to fit the various notions we come across. ('Ontology' is a posh word misappropriated from philosophy. In philosophy it means the study of meaning. In computing, it means the high level structure of a terminological model.)

There is a separate tutorial on the GALEN ontology for medical terminology which explains how it is organised. There is also a detailed manual and set of papers available as GALEN-IN-USE Deliverable 5.1. However, this is sufficient for now.

4.5.2 Natural kinds and elementary categories

Which things should be represented by elementary categories introduced using newSub and which things should be defined using a name statement? There is no hard and fast rule. Philosophers argue about whether or not there is any such thing as an elementary concept. However, any computer system must start somewhere from a hierarchy of elementary things.

The answer depends partly on the application and partly on the nature of the thing being represented. Philosophers and psychologists speak of many things which are 'natural kinds' - we know what they are when we see one but would be hard put to give a definition. Notions such as 'dog', 'child', 'tree', 'red' are usually thought of as natural kinds. It might be possible to define them in some sense for some purposes, but that definition might not correspond very well with common usage, and we certainly don't recognise them by checking their definition. On the other hand, ideas such as or 'parent' - a person who has children - or 'car-owner' - a person who owns a car - are relatively straightforward to define, at least for the purposes of a convenient model to cover 95% plus of all usage.

Roughly speaking, it is a waste of time and usually counter productive to try to define a representation for natural kind in GRAIL. Natural kinds can be described by necessary statements, but they will almost always be elementary categories. The reverse is not true. There may be many things in a model which could be defined, but whose definitions are irrelevant to any likely application. Alternatively we may just not yet have refined the model that far. A common strategy is to put things in first roughly as elementary categories and then come back and define them more carefully later.

4.5.3 When there must be multiple parents in the asserted hierarchy?

Ideally, no elementary category would have more than one elementary parent - i.e. no elementary category would be subsumed by more than one other elementary category. Put another way, the asserted hierarchy ought to be a pure tree.

Unfortunately, GRAIL is not sufficiently powerful to allow us to make this restriction absolute. In fact the GRAIL compiler will allow you to add many elementary parents to an elementary category, but it is in general a very bad idea to do so because it throws away most of the power of GRAIL.

There are normally only two places where there should be multiple parents in the asserted hierarchy: at the very top and at the very bottom.

At the very top, there is sometimes a need for very general categories which cut across other categories. An example from the medical experience is the notion of a 'disorder' which can be

anything from a tumour to a temperature. It cuts across so many other categories that it is easier to create a second supplementary structure than to try to reshape the rest to fit.

At the very bottom, there are sometimes values which are not mutually exclusive. For example in some versions of the medical hierarchy 'both-sided' is made a child of both 'left-sided' and 'right sided'.

The keywords addSuper and addSub can be used to add an additional asserted parent or child to an existing elementary category. Note that they may only be used between elementary categories. For example:

```
SymbolicValueType newSub SidedValueType.
  SidedValueType newSub leftSided.
  SidedValueType newSub rightSided.

  rightSided newSub bothSided.
  bothSided addSuper leftSided.
```

It makes no difference in this case which parent is added with newSub and which with addSuper.

Again we emphasise: you are unlikely to produce models that work well and are easy to maintain if they contain more than a very few statements using addSub or addSuper.

4.6 Summary: Principles of building Re-usable Models in GRAIL

Here then are two key principles for producing re-usable models in GRAIL:

- Keep the asserted hierarchy of elementary categories simple and clean. Always make sure there is just one principle involved in the asserted hierarchy - e.g. whether vehicles are road or rail vehicles. Don't mix different principles in the same hierarchy;
- Make as much of the information as possible explicit by using necessary statements and definitions. Check the categories when you are finished to make sure that all of the information which applications may need is explicitly represented in the criteria.

A common problem in many representations which violates these principles, is that modellers confuse the 'part-of' and 'kind-of' relations. Often this is inevitable, because many systems do not provide adequate support for 'part-of' relations. This is one of GRAIL's important features which will be explained in the next chapter.

5 Refinement and Transitivity: What makes GRAIL different

GRAIL differs from most other description logics and frame systems in providing extra facilities to cope with transitive relations and the interaction of transitive relations and subsumption. These are amongst the most powerful features of GRAIL. They are what allows GRAIL to deal cleanly with part-whole and causal relations. However, they can also be confusing, particularly the operator for co-ordinating transitive relations and subsumption which is known as refinedAlong or specialisedBy. (The fact that there are two different variants of the syntax reflects the difficulty we have had explaining it to others.)

5.1 Transitivity

Transitive relations are relatively straightforward to understand. The basic pattern is that *"The part of the part is a kind of part of the whole"*. The pattern carries on indefinitely.

So if we say that seat covers are parts of seats, and seats are part of vehicles, then seat covers are also part of vehicles. Likewise if tread is part of tyres, and tyres are part of vehicles, then the tread is part of the vehicle.

(Philosophers and linguists worry about whether indefinitely long chains of "part-of" links should really be considered transitive, but for purposes of classification, we shall maintain here unequivocally that they should.)

To illustrate transitivity, let us extend the model a bit further as follows:

```
">>>>>>>>>>>>>>>>>>>>>>
> New categories
<<<<<<<<<<<<<<<<<<<<<<"
DomainCategory newSub VehiclePart.
   VehiclePart newSub Seat.
   VehiclePart newSub Wheel.
   VehiclePart newSub Nut.
   VehiclePart newSub SeatCover.
   VehiclePart newSub Tyre.
   VehiclePart newSub Tread.

DomainCategory newSub Damage.
   Damage newSub Puncture.
   Damage newSub Wear.
   Damage newSub Tear.


">>>>>>>>>>>>>>>>>>>>>>>>>>
>New attribute and its transitive declaration.
<<<<<<<<<<<<<<<<<<<<<<<<<<"
```

```
DomainAttribute newAttribute hasPart isPartOf allAll oneMany.
hasPart transitiveDown.


">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>Sanctions
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
VehiclePart grammaticallyAndSensibly isPartOf Vehicle.
VehiclePart grammaticallyAndSensibly isPartOf VehiclePart.


">>>>>>>>>>>>>>>>>>>>>>
> Facts
<<<<<<<<<<<<<<<<<<<<<<<<"
Vehicle necessarily hasPart Seat.
   Seat necessarily hasPart SeatCover.
Vehicle necessarily hasPart Wheel.
   Wheel necessarily hasPart Tyre.
      Tyre necessarily hasPart Tread.

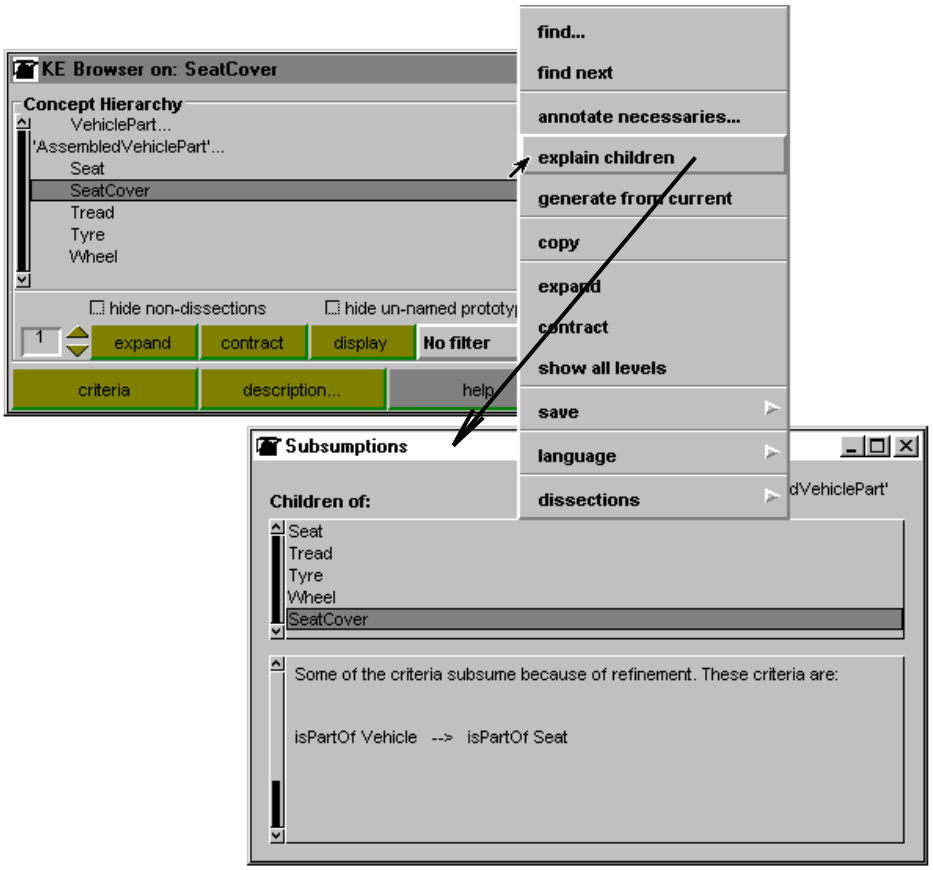
">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
> The definitions for demonstrating transitive relations
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
(VehiclePart which isPartOf Vehicle) name AssembledVehiclePart
```

Note the declaration:

```
hasPart transitiveDown
```

immediately following the newAttribute statement for hasPart. This declaration states that the attribute hasPart is transitive. (The downwards direction marker is now obsolete but can be useful to the compiler for optimisations.)

Evaluate everything and then browse AssembledVehiclePart. You should get an result as shown below. The list of subsumed categories includes Tread and SeatCover. Without the transitiveDown declaration neither Tread nor SeatCover would have been included. (Note that the order of the list of subsumed items is irrelevant - in fact it is alphabetical.)



The option **explain children**, as shown, offers a brief explanation of why each of these subsumptions exists.

5.2 The specialisedBy operator ('refinement')

The specialisedBy operator (and its converse form refinedAlong) are slightly more complicated.

The pattern is that *"Damage to a part is a kind of damage to the whole"*. For example, damage to a tyre's tread is a kind of damage to a vehicle. Conversely, a vehicle with a damaged part is itself damaged, *e.g.* a vehicle with a damaged tyre tread is a kind of damaged vehicle.

To illustrate the point, add the following to the model:

```
">>>>>>>>>>>>>>>>>>>>>>>>
>New attribute and specialisedBy declaration
<<<<<<<<<<<<<<<<<<<<<<<<"

DomainAttribute newAttribute hasLocation isLocationOf allAll manyOne.
hasLocation specialisedBy isPartOf.

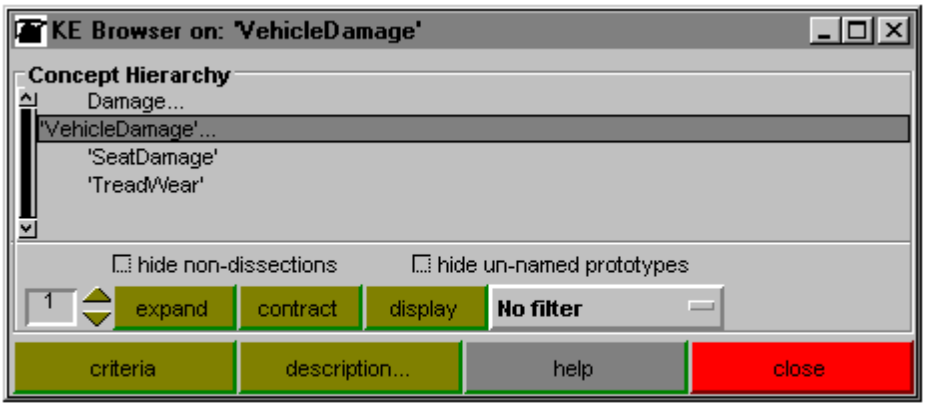
">>>>>>>>>>>>>>>>>>>>>>>>
Sanctions for hasLocation
<<<<<<<<<<<<<<<<<<<<<<<<"

Damage grammaticallyAndSensibly hasLocation VehiclePart.
Damage grammaticallyAndSensibly hasLocation Vehicle.

">>>>>>>>>>>>>>>>>>>>>>>>>>>>
> The definitions for demonstrating specialisedBy
<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
(Wear which hasLocation Tread) name TreadWear.
```

(Damage which hasLocation Seat) name SeatDamage.
(Damage which hasLocation Vehicle) name VehicleDamage.

Evaluate the above and then browse VehicleDamage. You should find a pattern as shown in the screen dump below. Both SeatDamage and TreadWear are subsumed by VehicleDamage because of the specialisedBy declaration.



To illustrate the effect of the converse pattern, enter the following further definitions, evaluate them and browse DamagedVehicle.

(Tread which isLocationOf Wear) name WornTread.
(Seat which isLocationOf Damage) name DamagedSeat.

(Car which hasPart (Tyre which hasPart WornTread)) name CarWithWornTread.
(Bus which hasPart DamagedSeat) name BusWithDamagedSeat.
(Vehicle which isLocationOf Damage) name DamagedVehicle.

DamagedVehicle should now subsume both CarWithWornTread and BusWithDamagedSeat.

(Note also in passing the use of two levels of parentheses in the definition of CarWithWornTread. GRAIL syntax allows definitions to be nested to any level required, but each category must be enclosed in brackets as shown in the example.)

5.3 Transitive relations and cardinality

Strictly speaking, it makes no sense to talk about a transitive relation with a cardinality of manyOne or oneOne *i.e.* for the attribute to be both transitive and 'single valued'. Obviously, if the relation is transitive, then there can be many values. However, GRAIL interprets single valued for transitive relations to mean that there can be only one value for each step which is equivalent to saying that the relation must form a strict tree. An analogous meaning is used for a single-valued relation which is specialisedBy another. Strict trees are very common and very useful, so having a value type for them is helpful.

Finally, GRAIL assumes that transitive relations do not form cycles, *i.e.* that there are never loops in the transitive relation, for example, it can never be the case that something is a part of itself if the hasPart attribute is declared as transitive. There is no check for this in the system, but cycles in transitive relations may cause the system to loop endlessly.

5.4 Why are specialisedBy and transitiveDown important

It is very common in knowledge representation systems to confuse part-whole relations with the kind-of (or subsumption) relation. For any given application, it is usually possible to work

around this limitation and to deliberately conflate part-whole and kind-of. However, as soon as you try to re-use the model, the work-arounds for one application get in the way of the next.

In later tutorials, many different applications of the transitiveDown and specialisedBy declarations will be described. The combination allows GRAIL models to be much simpler and more general than models in languages which lack these features.

5.5 How to remember how specialisedBy works

Everybody has trouble remembering the exactly how specialisedBy works when using it in a new situation. The best advice is to write down the basic pattern:

hasLocation specialisedBy isPartOf.

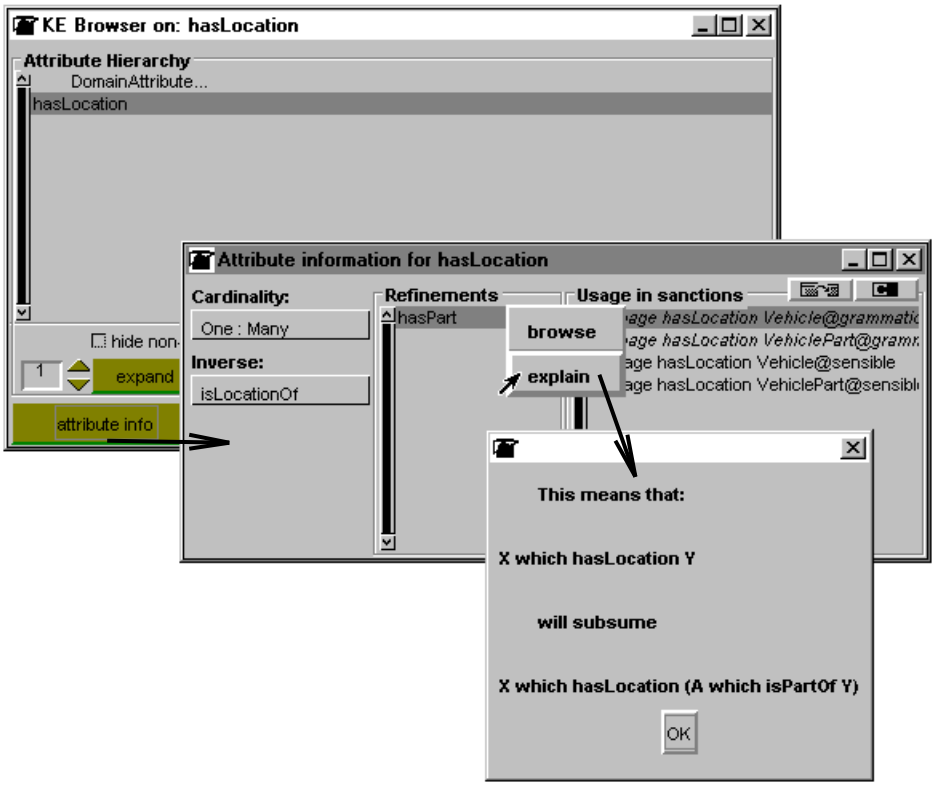
Therefore

Damage which hasLocation Thing

subsumes

Damage which hasLocation (Bit which isPartOf Thing)

Then match up the categories and attributes for your new problem precisely to this pattern. An additional prompt is available when browsing the refinements that have been applied toa particular attribute, as shown below:



6 Other Features

6.1 Extrinsic Information, 'Defaults', and Indexing

All information in definitions (which expressions) and necessary statements is inherited, without exception, by all concepts subsumed by the given concept. GRAIL does not support 'default reasoning' directly. If we make hasLocomotion flight part of the definition of birds, or if we add it to birds as a necessary statement, then hasLocomotion flight will be true of all birds in the model, including penguins, ostriches, and kiwis.

GRAIL does not support defaults directly. This is a deliberate choice. Default reasoning has been found to be extremely troublesome. One interpretation of the trouble is that there is just not enough information in a system like GRAIL to resolve the ambiguities which arise from defaults. GRAIL's approach is therefore not to try to resolve such ambiguities.

Instead, GRAIL makes it easy to hang 'extrinsic' information on the basic GRAIL model and use that information to identify the set of 'most specific' relevant information of a given sort. For example, we might define an attribute canFly with a value true or false and say:

Bird extrinsically canFly true.

Ostrich extrinsically canFly false

There is an operation which collects all the most specific extrinsic values, so in this case since Ostrich is more specific than Bird, it would collect just false. This gives many of the effects of default reasoning, but it cannot guarantee that there will be just one answer or that the set of answers will be consistent. It is up to the application or the user to decide how to chose the right answer if more than one is found, or indeed to accept all of the answers. The correct strategy depends on the application, and it often requires additional information beyond that held in the GRAIL hierarchy.

Extrinsic links do not affect classification. If we form the category Animal which canFly true we won't find Bird. This is a serious disadvantage, but it is the price to pay for a clean, simple system. We argue that ideas such as whether or not birds fly are not properly 'terminological', they are not a part of the naming of the thing, otherwise there would be, by definition, no exceptions.

However, we also argue, and have much experience to show, that having a clean explicit terminological structure minimises the clashes and ambiguities which otherwise occur with default reasoning in multiple hierarchies.

One way to think about this is that the GRAIL terminological structure provides a high quality index for looking up other information including information usually managed by default reasoning. Having a good index does not eliminate the need for reasoning, but it makes the reasoning easier and quicker.

6.2 Coding Schemes

GALEN includes services which allow annotation of concepts with codes from "external" coding schemes. The intention is to allow linking of the GALEN CORE with existing medical coding schemes. Volume J provides detailed descriptions of the services available to support this activity.

It is rather difficult to illustrate the code conversion capabilities of GALEN with the tutorial's transport example. However, the coding services can be seen simply as a more sophisticated version of the extrinsics mechanism described above.

6.3 Language

In the tutorial the only names given to expressions are their GRAIL names, otherwise known as ‘knowledge names’. Every knowledge name must identify exactly one category or attribute. No synonyms are allowed, and the names must conform to GRAIL syntax - they must contain no spaces and begin with a letter.

One of the main functions of GALEN is to build terminologies with convenient, user friendly names. In fact, a major function is to build multilingual terminologies in which any term can have names in each language.

The GALEN terminology servers contain special modules for generating natural looking text from GRAIL expressions and for ‘annotating’ GRAIL categories and attributes to provide multilingual lexicons. This is beyond the scope of this tutorial.

7 Glossary

This section provides a brief recap of the GRAIL features we've introduced in this tutorial. It is not exhaustive - please let us know what's missing.

attribute

or 'relationship' - if you consider the model to be made up of nodes (concepts) and arcs relating them, attributes are the arcs…

category

…and categories are the nodes. The objects in GRAIL representing concepts - things - in the world. Sometimes 'concept'. Categories may be **elementary** (ie introduced using the newSub operator) or **complex** - made up by combining together categories with relationships using the which operator.

canonical form

the unique form of a complex GRAIL concept - ie with any redundancies or tautologies removed. See GALEN Deliverable 6 for a much more detailed treatment of this.

concept

see **category**

criterion

an attribute-value pair. Eg hasOwner Person. There are several varieties, indicating where they are used. **Defining criteria** are criteria which make up the definition of a complex category; for example, the defining criteria of

Person which hasSex male

is the single (in this case) criterion.

hasSex male

necessary criteria (sometimes 'essential', or 'conventional' criteria) are those properties which have been applied to a category using necessary statements. Properties are inherited along the subsumption network maintained by the GRAIL engine; **inherited criteria** are those necessary criteria that hold true of a category by inheritance along with any defining criteria of 'explicit' parents. The union of the defining, necessary and inherited criteria are known as the **complete criteria**.

GALEN

is the project itself - Generalised Architecture for Languages, Encyclopaedias and Nomenclatures in Medicine.

GRAIL

(or, more properly, 'the GRAIL Kernel') is the concept modelling language (or formalism) being developed in GALEN. It is the GALEN Representation and Integration Language.

model

the set of information presented as source files to the system - *or* - the result of the system (implementing the GRAIL formalism) having interpreted those sources. Also sometimes the set of implied categories that the 'model' sanctions. The word 'model' is over-used, and often for different things. Usually context helps.

naming

the process of attaching a useful 'nick-name' (sometimes 'knowledge name') to a complex concept.

necessary statement

a necessary statement applies an indefeasible criterion (or property) to a category. For example

(Person which isOwnerOf RoadVehicle) necessarily hasAge old

The necessary operator is bi-directional - this statement has said something about people that drive road vehicles, *and* the concept of 'old'. Two unidirectional variants exist - topicNecessarily and valueNecessarily; to say things only about the first concept, (as this example should be written):

(Person which isDriverOf RoadVehicle) topicNecessarily hasAge old

prototype

a complex GRAIL category - sometimes also known as a 'particularisation'.

refinement

the notion of making one attribute transitive across another.

sanctioning

Before we can combine categories and relationships to form complex categories, we need to say that we can. This mechanism, of telling the system which kinds of things can be combined (using particular attributes) to other kinds of things, is called *sanctioning.* We take the position that nothing is allowed until it is sanctioned by means of a 'sanctioning statement' (often just 'statement'; also previously known as a 'triple'). There are multiple levels of sanctioning: *grammatical* and *sensible.* Before you can create a complex category, or attach a necessary statement, it must be sanctioned to the sensible level. To sanction to a sensible level, it must be already sanctioned at the grammatical level. Sanctions are inherited.

SMK

Structured Meta-Knowledge - a precursor to GRAIL.

subsumption

the notion of 'is kind of'. If Man is 'a kind of' Person, then we say that Person subsumes Man, or that Man is subsumed by Person. Person is the subsumer, Man the subsumee. **Asserted (or explicit) subsumption** is the subsumption that comes about because of explicit facts said by the modeller (using the newSub or addSub operators); **formal subsumption** is the subsumption that comes about due to the rules of the GRAIL formalism; for example that Person subsumes Person which hasSex male.

Value type

Value types are concepts which have different subsumption rules applied to them. They typically appear only as values in criteria. Some kinds of value types are built into the system

(eg IntegerValueType); additional types can be introduced as required (eg AgeValueType in this tutorial).

Appendix Four:
MIE paper: Rubrics to Dissections to GRAIL to Classifications

Rogers JE, Solomon WD et al. (1997) Rubrics to Dissections to GRAIL to Classifications. Fifteenth International Congress of the European Federation for Medical Informatics, MIE-97 Thessaloniki, Greece

Rubrics to Dissections to GRAIL to Classifications

JE Rogers[1] WD Solomon[1] AL Rector[1] P Pole[1] P Zanstra[2] E van der Haring[2]
1) Medical Informatics Group, Department of Computer Science, University of Manchester, UK
2) Medical Informatics Group, Catholic University of Nijmegen, Netherlands
http://www.cs.man.ac.uk/mig/galen/ or e-mail galen@cs.man.ac.uk

Abstract. This paper summarises the process in the GALEN-IN-USE project by which rubrics from traditional medical coding schemes are analysed into an intermediate, relatively informal conceptual representation which is then automatically translated into the GRAIL formalism and its Common Reference Model.

1 Introduction

GALEN-IN-USE is an EU funded project, a major goal of which is the development of tools and methods to assist in the collaborative construction and maintenance of surgical procedure classifications. Techniques and tools developed in the previous GALEN project [1,2,3,4,5,6] will support this task. Taking part in the initial phase are four national coding and classification centres: WCC (Netherlands), SPRI (Sweden), CNR (Italy) and University of Ste. Etienne (France). The goal is to author, using the GRAIL formalism [6], conceptual representations of individual surgical procedures, with each centre covering roughly one quarter of the total surgical domain. A combined total of 15-20,000 individual representations will be authored by the end of the project. The resulting GRAIL representations will be integrated into the existing GALEN Common Reference Model (CRM) [1,3,4,5]. This will allow:

- an initial classification of the represented procedures to be automatically derived, based on the knowledge explicitly authored in the analyses and the knowledge already in the reference model;
- machine generation of natural language expressions for all representations, in five European languages;
- refinement, extension and reorganisation of the classification using new classification management tools.

More than 20 individual clinicians have been recruited to analyse original code rubrics into conceptual representations, but most have little or no prior experience of the GRAIL formalism or of the particular ontology and modelling style of the Common Reference Model (CRM). This presented a significant challenge to the project: how to reduce the need for training to occur in the complexities of GRAIL and the CRM before any work could begin.

2 An Intermediate Representation

The solution proposed to the training problem was to begin work using a simpler, intermediate conceptual representation [7]. This was originally conceived as a migration step towards eventually authoring directly in GRAIL. The representation was designed in such a way that conceptual representations authored using it might then be automatically, or semi-automatically, expanded into GRAIL. The representation also allows the authors to capture some concepts which the GRAIL formalism in its present form is unable to handle. Finally, the representation serves as the preferred format in which the centres examine and validate their own, and each other's work. The intermediate representation is broadly similar to those used by the CANON group or MEDS [8,9,10,11]. It is characterised by:

- a relatively small set of semantic links (ACTS_ON, IS_PART_OF), compared to the CRM;
- a two-tier domain ontology (known as the 'descriptor list') specific to the surgical domain. Descriptors (leg, excising, tumour) are explicitly typed by one of a small number of classes (e.g. anatomy, deed, lesion);
- a set of constraints, declaring which links may be used with which descriptor classes;
- a grammar defining a layout, or 'template', for well-formed representations.

Domain experts in the centres work using existing local coding schemes (WCC, NCSP etc.) to scope their task. The rubrics (text) and associated codes, but not the original hierarchy, are extracted from the scheme. Working on sections of a few hundred related rubrics at one time, each rubric is manually analysed to author a conceptual representation of its meaning, using the intermediate representation. The immediate result of this analysis and authoring is a called a 'dissection' of the rubric.

The four centres produced more than 1200 'dissections' in the first four months. Figure 1 shows four completed dissections. Each has a header section, containing information about the original rubric and coding scheme, followed by the conceptual representation itself, introduced by the MAIN keyword. The semantic links are capitalised. Descriptors appear in lower case, preceded by their descriptor class. Initially, authoring involved directly editing an ASCII text file. Any convenience and familiarity which this afforded to the authors was, however, outweighed by the numerous spelling and formatting errors which resulted, preventing satisfactory parsing of the interchange file into the GRAIL expansion environment. Subsequent analyses will be authored using a purpose-built tool, the Surgical Procedure Editing Tool (SPET).

Figure 1.

```
RUBRIC "operations on papillary muscle"
CODE "35.31"
MAIN deed:surgical deed
  ACTS_ON anatomy:papillary muscle
  HAS_OTHER_FEATURE method VALUE induced arrest of heart
```

```
RUBRIC "reattachment of papillary muscle"
CODE "35.31.i2"
MAIN deed:repairing
  ACTS_ON anatomy:papillary muscle
  BY_TECHNIQUE deed:reattaching
        ACTS_ON anatomy:papillary muscle
  HAS_OTHER_FEATURE method VALUE induced arrest of heart
```

```
RUBRIC "dividing of papillary muscle"
CODE "35.31.i1"
MAIN deed:dividing
  ACTS_ON anatomy:papillary muscle
  HAS_OTHER_FEATURE method VALUE induced arrest of heart
```

```
RUBRIC "repair of papillary muscle"
CODE "35.31.i3"
MAIN deed:repairing
  ACTS_ON anatomy:papillary muscle
  HAS_OTHER_FEATURE method VALUE induced arrest of heart
```

3 Expanding dissections into GRAIL

Dissections authored in the intermediate representation are subsequently imported into an environment (TIGGER) built to manage the process of converting them into GRAIL, simultaneously translating them into both the ontology and the style of the CRM. Imported dissections are first parsed for syntax and for whether they comply with certain agreed modelling conventions (such as that all deeds must ACTS_ON something, a convention taken from CEN [12]). The final GRAIL produced is generally more complex than the dissection from which it comes - sometimes very much more so. The translation process is known as 'expanding' and the GRAIL produced from a single dissection as its GRAIL 'expansion'.

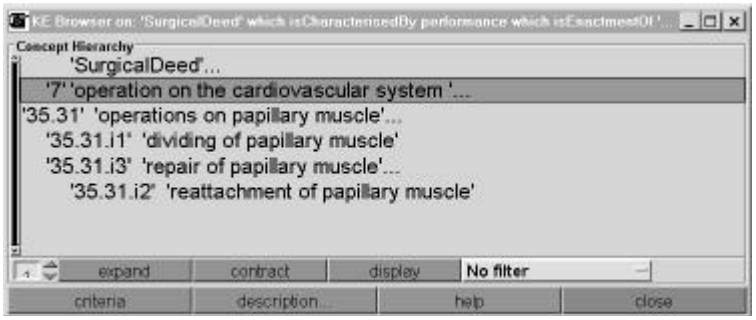
Figure 2 shows a dissection (left) and an automatically generated expansion (right). An expansion comprises a GRAIL representation of the concept at hand, and a series of statements attaching incidental, non-classificatory information to that GRAIL concept, such as the text of the original rubric or the name of the original source file. TIGGER automatically generates, in batches or individually, a GRAIL expansion for each original dissection. However, automatic expansions can only be considered candidate GRAIL conceptual representations of the original rubrics: some may be rejected as invalid when presented to a terminology server. This may occur, for example, if there is a cardinality conflict between CRM semantic links used in the expansion. Rejection may indicate a problem which requires alterations to the original dissection, or to the CRM. A few dissections whose expansions are rejected may require their GRAIL representation to be done manually, bypassing the intermediate representation completely.

Figure 2.

Original Dissection	Generated GRAIL expansion
RUBRIC "dividing of papillary muscle" CODE "35.31.i1" MAIN deed:dividing ACTS_ON anatomy:papillary muscle HAS_OTHER_FEATURE method VALUE induced arrest of heart	(SurgicalDeed which isCharacterisedBy (performance whichG isEnactmentOf (Dividing whichG < playsClinicalRole SurgicalRole actsSpecificallyOn PapillaryMuscle hasSubprocess InducedCardiacArrest>))) extrinsically hasRubric 'dividing of papillary muscle'; extrinsically hasCode '35.31.i1'; extrinsically hasPhysicalSource 'cnr.txt'.

Sets of automatic expansions produced in this way are presented to the terminology server for classification. The resulting hierarchy of valid expansions may be browsed in a number of ways. The screenshot (figure 3) shows the automatic classification which is derived for the four dissections given in figure 1. The classification of 'operations on the papillary muscle' as a kind of 'operation on the cardiovascular system' occurs because the CRM 'knows' that the papillary muscle is part of the heart which is, in turn, a component of the cardiovascular system. The GRAIL refinement operation [4] is used to declare that actions on part of something are subsumed by actions on the

Figure 3.



whole. By contrast, the classification of 'reattachment of papillary muscle' as a child of 'repair of papillary muscle' instead of a sibling (as in the original coding scheme) comes from the knowledge explicitly authored in the intermediate representation.

To do the automatic expansion of dissections, however, the TIGGER first requires explicit CRM mappings to be declared for both the descriptors and the links used in a given dissection.

3.1 Mapping the descriptors

This task of declaring mappings for descriptors is performed by modellers already familiar with the GRAIL formalism and the CRM ontology and style. Once declared, a mapping is presumed valid for all subsequent occurrences of the descriptor in any batch of dissections. As more dissections are processed, the list of already mapped descriptors grows. The accumulated, mapped descriptor list is made available via the SPET to all collaborating authors as a suggested core ontology for use in the next round of authoring dissections.

This approach has the advantage that the ontology with which the dissection authors must become familiar is initially quite small. Further, its growth is owned by the authors themselves but can be guided by those familiar with the GALEN ontology. This contrasts with the already large and complex ontology in the CRM, with which the authors would need to be familiar to author directly in GRAIL. Because the process of getting from dissections to GRAIL is planned to be unidirectional, this methodology also permits some redundancy or duplication of descriptors, as many descriptors can be mapped to the same CRM concept.

Mapping a given descriptor to the CRM is informed by inspection, using TIGGER, of all the dissections which employ it - either in the current batch or in all batches processed so far. Such inspection may provide clarification of what is meant by a descriptor, but may also reveal that one descriptor has been used with very different intentions by different authors. A mechanism exists for rejecting such ambiguous descriptors and their associated dissections at this stage, to invite re-authoring.

3.2 Mapping the Links

The links available to dissection authors were chosen such that each is equivalent in intention to the common parent of a range of more expressive links already present in the CRM. Figure 4, for example, shows the CRM links which are to be considered by TIGGER as possible default mappings for the dissection link IS_PART_OF. The mappings from dissection links to the CRM are, therefore, necessarily one-to-many and are declared by the same team of modellers undertaking the descriptor mapping.

To expand a dissection link, TIGGER must determine which, if any, of its candidate CRM link mappings is most appropriate. To achieve this, TIGGER 'translates' the descriptors either side of a dissection link into their declared CRM entity mappings. The candidate CRM link mappings are then tested in list order: the first one permitted to be used in the CRM between the two entities is chosen. Thus, the dissection fragment:

segment IS_PART_OF intestine

is expanded into the GRAIL:

Segment which isSpecificLinearDivisionOf Intestine.

because the CRM includes the constraint:

Segment sensibly isSpecificLinearDivisionOf TubularBodyStructure.

This mechanism can also be used to detect dissections which, whilst considered 'well formed' within the limited dissection grammar, are still semantically incorrect. For example:

MAIN excising
ACTS_ON tumour

Figure 4.

Dissection Link	Default possible CRM mappings
IS_PART_OF	isSolidRegionOf isSpecificStructuralComponentOf isStructuralComponentOf isSpecificSolidDivisionOf isSolidDivisionOf isSpecificLinearDivisionOf isLinearDivisionOf isSpecificSurfaceDivisionOf isSurfaceDivisionOf isSpecificLayerOf isLayerOf

IS_PART_OF liver

…can not be expanded: none of the candidate mappings of IS_PART_OF is permitted to be used in the CRM between [Excising], a process, and [Liver], a structure. In future, some semantically incorrect dissections may be rejected at the authoring stage - the SPET will use both the dissection grammar and a limited constraint mechanism to prevent certain link-descriptor pairings being entered at all.

4 Limitations and Future extensions

Coding scheme rubrics frequently contain disjunctions (Excision of tumour or cyst) or conjunctions (Drainage and marsupialisation of cyst). To make the intermediate representation simple, bracketing was omitted from its syntax. As a result, it can only support relatively simple conjunctions or disjunctions. Rubrics with more complex relationships (e.g. partial or complete excision of tibia and fibula, with prosthesis) must be manually enumerated in all their logically and semantically correct conceptual forms. True negation also remains unsupported at present by the GRAIL formalism. However, most of the rubrics which might at first appear to require negation are exclusion or exception criteria. The intermediate representation includes mechanisms for identifying such criteria, which may then be handled within GRAIL by various modelling workarounds or future extensions to the formalism itself.

The relatively relaxed approach to building the descriptor list, with its rudimentary and *ad hoc* class hierarchy, risks it growing to unmanageable and un-navigable proportions. Imposing a more formal organisation would be to some extent to re-invent GRAIL. This may be obviated by offering navigation of the set of mapped descriptors via the hierarchy of their corresponding CRM conceptual mappings.

Both the descriptor list, and the CRM itself, are presently authored in English. This is not the first language of many dissection authors. The generation of natural language expressions for the final GRAIL concepts requires annotations in the destination languages for each CRM primitive. A mechanism to address both of the these problems is being studied: dissections will be authored in the local languages, using a local language descriptor list. Each authoring centre will separately maintain a many-to-one translation table from the local terms to a common, English descriptor list shared between all centres. The various translation tables, and the explicit English descriptor to CRM mappings, may be read backwards to derive a list of possible linguistic annotations.

5 Discussion and Conclusion

Although originally conceived as a migration step towards full GRAIL authoring, the intermediate representation and automated expansion process have proved effective in their own right. A high proportion of rubrics can be represented and reliably expanded automatically, and it may be more efficient to author the small remainder directly in GRAIL than to make further enhancements. The success of the intermediate representation is such that, whilst it remains the intention of the project to export CRM modelling expertise from its current localised base, this is no longer on the critical path for the immediate task of building a surgical procedure classification. Further, adding an intermediate layer between knowledge authors and the final representation serves to insulate them from changes in the CRM, and allows those changes to take place more easily and with less disruption.

We are confident that much of the surgical procedure domain can continue to be captured using the intermediate representation, facilitating the involvement of many domain experts by deferring indefinitely any need for them to become familiar with GRAIL or the CRM. A useful by-product of the process is that the act of declaring link and descriptor mappings is building a partial meta-model description of the CRM ontology and style. This will form an important resource for the exporting of CORE modelling expertise, when that occurs.

With thanks to the centres involved in dissection authoring, and others in the GALEN-IN-USE consortium

GALEN-IN-USE is funded as part Framework IV of the EC Healthcare Telematics research program.

6 References:

1. Rector, A. (1994). Compositional models of medical concepts: towards re-usable application-independent medical terminologies. in Knowledge and Decisions in Health Telematics P. Barahona and J. Christensen (ed). IOS Press. 133-142.

2. Rector, A., W. Solomon, W. Nowlan and T. Rush (1995). A Terminology Server for Medical Language and Medical Information Systems. Methods of Information in Medicine, Vol. 34, 147-157

3. Rector, A., A. Gangemi, E. Galeazzi, A. Glowinski and A. Rossi-Mori (1994). The GALEN CORE Model Schemata for Anatomy: Towards a re-usable application-independent model of medical concepts. Twelfth International Congress of the European Federation for Medical Informatics, MIE-94, Lisbon, Portugal, 229-233.

4. Rector, A. (1995). Coordinating taxonomies: Key to re-usable concept representations. Fifth conference on Artificial Intelligence in Medicine Europe (AIME '95), Pavia, Italy, Springer. 17-28.

5. Rector, A., JE. Rogers, P. Pole (1996) The GALEN High Level Ontology. Fourteenth International Congress of the European Federation for Medical Informatics, MIE-96, Copenhagen, Denmark

6. Rector, A. and W.A. Nowlan (1993). The GALEN Representation and Integration Language (GRAIL) Kernel, Version 1. The GALEN Consortium for the EC AIM Programme. (Available from Medical Informatics Group, University of Manchester).

7. Gaines BR, Shaw ML and Woodward JB (1993). Modelling as Framework for Knowledge Acquisition Methodologies and Tools. Internatonal Journal of Intelligent Systems 8(2): 155-168.

8. Campbell KE, Das AK and Musen MA (1994). A logical foundation for representation of clinical data. JAMIA 1(3): 218-232.

9. Cimino J (1994). Controlled Medical Vocabulary Construction: Methods from the Canon Group. Journal of the American Medical Informatics Association 1(3): 296-197.

10. Evans D (1988). Pragmatically-structured, lexical-semantic knowledge bases for unified medical language systems. Proceedings of the Twelfth Annual Symposium on Computer Applications in Medical Care, Washington DC, IEEE Computer Society Press: 169-173.

11. Huff S and Warner H (1990). A comparison of Meta-1 and HELP terms: implications for clinical data. Fourteenth Annual Symposium on Computer Applciations in Medical Care (SCAMC-90), Washington DC, iEEE Computer Society Press: 166-169.

12. CEN PT 251:ENV 1828 Surgical Procedure Modelling