CHANGE MANAGEMENT AND SYNCHRONIZATION OF LOCAL AND SHARED VERSIONS OF A CONTROLLED VOCABULARY

A DISSERTATION

SUBMITTED TO THE PROGRAM IN

BIOMEDICAL INFORMATICS

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Diane Elizabeth Oliver

August 2000

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

_____

Mark A. Musen, M.D., Ph.D., Principal Adviser
(Stanford Medical Informatics)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

_____

Edward H. Shortliffe, M.D., Ph.D.
(Stanford Medical Informatics)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

_____

Yuval Shahar, M.D., Ph.D.
(Stanford Medical Informatics)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

_____

Gio Wiederhold, Ph.D.
(Department of Computer Science)

Approved for the University Committee on Graduate Studies:

_____

Abstract

To share clinical data and to build interoperating computer systems that permit data entry, data retrieval, and data analysis, users and systems at multiple sites must share a common controlled clinical vocabulary (or ontology). However, local sites that adopt a shared vocabulary have local needs, and local-vocabulary maintainers make changes to the local version of that vocabulary. If the local site is motivated to conform to the shared vocabulary, then the burden lies with the local site to manage its own changes and to incorporate changes from the shared version at periodic intervals. I call this process synchronization. In this dissertation, I present an approach to change management and synchronization of local and shared versions of a controlled vocabulary. I describe the CONCORDIA model, which comprises a structural model, a change model, and a log model to which the shared and local vocabularies conform. I demonstrate use of this model in the implementation of a synchronization-support tool that supports carefully controlled divergence. To evaluate my model and methods, I performed synchronization on a small test set of medical concepts in the subdomain of rickettsial diseases. The CONCORDIA model served as an effective approach for representation and communication of vocabulary change.

Acknowledgements

I thank Mark Musen, Yuval Shahar, Ted Shortliffe, and Gio Wiederhold for their contributions as members of my dissertation committee. I thank Octo Barnett for introducing me to the challenges of controlled medical vocabularies. I thank Keith Campbell, Alan Rector, Samson Tu, Ray Fergerson, Larry Fagan, John Gennari, Jeremy Wyatt, and Darlene Vian for valuable discussions and inspiration. I thank Lyn Dupre for editing this dissertation. Above all, I am grateful for the support from my family and friends.

This work was funded by the Agency for Health Care Policy and Research, the Veterans Administration, and the National Library of Medicine.

Table of Contents

List of Figures

List of Tables

List of Boxes

1 Management of Change in Controlled Medical Vocabularies

"Quot homines tot sententiae; suos quoque mos."

(So many men, so many minds, every one has his point of view.)

(Terence, c. 170 B.C.) [Sargeaunt 1995]

Controlled medical vocabularies in computer-based patient-record systems facilitate data entry [Musen 1995, Nowlan 1991], data retrieval [Elhanan 1997], data aggregation [Baorto 1997], and data analysis [Paty 1994]. Work on patient-record systems [Cimino 1994, Nowlan 1991], diagnostic decision-support applications [Elhanan 1996, Giuse 1995], alerts and reminders [Thurin 1995], and computer-based guidelines [Musen 1996, Ohno-Machado 1998, Shahar 1998] has revealed that management of medical vocabulary and its evolution is crucial. These vocabularies are frequently large: on the order of thousands [Forrey 1996], tens of thousands [Cimino 2000], or hundreds of thousands [Humphreys 1996a, Humphreys 1997] of elements. Maintainers of large medical vocabularies find that updates are essential and that management of change is a nontrivial problem [Cimino 1996a, Olson 1996, Robinson 1997, Tuttle 1996]. Changes occur due to advances in medical knowledge and due to changes in user requirements. Changes typically involve modifications to concepts, terms, and relationships in structured vocabularies.

Independent development of a multitude of coding systems, thesauri, and data dictionaries in computer-based systems for health care that were not initially intended to interoperate has led to the creation of numerous different controlled medical vocabularies. Nonetheless, if health-care providers, payors, patients, administrators, researchers, and government regulatory agencies want to communicate health-care data via computer-based systems, the software they use must recognize the names and meanings of the same data elements. Hence, there is a need for a common shared controlled medical vocabulary.

It is not sufficient to share only the names and dictionary definitions of data elements. Management of a vast number of terms requires organization of those terms so that both people and computer-based systems can use them. Because the existing diverse

controlled vocabularies were developed by different groups for different purposes, their developers used a wide variety of organizational structures and concept-representation techniques. Yet, if the goal of interoperation is to be achieved, then we must have a common understanding of how terms and concepts are represented and organized.

Assume that health-care organizations recognize that shared medical vocabularies are essential. These organizations recognize that it is too expensive for every group to develop its own comprehensive medical vocabulary, and to create translations to every other group's vocabulary. They agree that pooling resources to create a vocabulary that everyone can use is a good idea. However, local sites that adopt a shared vocabulary have local needs that prompt the local-vocabulary maintainers to make changes to the local version of that vocabulary. For a local site, there is a tradeoff between having autonomy over a local vocabulary and conforming to a shared vocabulary to obtain the benefits of interoperation. If the local site is motivated to conform, then the burden lies with the local site to manage its own changes and to incorporate the changes of the shared version at periodic intervals.

Ultimately, to support interoperation of systems that manage vocabularies and of applications that use vocabularies that change over time, the community needs standards for the way that concepts are represented, for the types of changes that can be made, for the semantics of those changes, and for the way that changes are documented. Given such standards, automated or partially automated tools could support the task of updating mapped, merged, or divergent vocabularies. It is important to note, however, that a standard for representation of changes to concepts invariably depends on the standard chosen for representation of those concepts.

In this chapter, I begin by stating the hypothesis of my research. I elaborate the need for a shared vocabulary and the problem of local divergence. Next, I summarize my approach for communicating about change at the shared and local level. I describe the implementation of the system, **Concept Manager**, that I have built to evaluate my methods. Concept Manager is a system of five software tools that supports vocabulary management in shared and local environments. Finally, I comment on related vocabulary problems that are not a part of my research, and I offer a guide to the reader that explains the organization of this dissertation.

1.1 Hypothesis

Communication of changes between an organization that develops a shared vocabulary and a local site that uses and adapts that vocabulary requires a shared understanding of an explicit formal vocabulary structural model, change model, and log model; the utility of such formal models can be demonstrated by their implementation in a synchronization-support tool that enables a vocabulary developer to synchronize a local version of a shared vocabulary with the shared version from which it was derived.

1.2 The Need for a Shared Controlled Medical Vocabulary

There are multiple reasons why health-care organizations need to use a controlled medical vocabulary in computer-based patient records, and there are many controlled medical vocabularies that exist for different purposes. Reasons to use such vocabularies in patient-record systems include displaying terms for structured data entry, indexing text and images by content, naming data elements in databases, linking patient data to literature-retrieval systems, reporting data to payors and government agencies, and connecting decision-support systems with patient data. Well-known examples of controlled medical vocabularies are the International Classification of Diseases, Ninth edition, Clinical Modification (ICD-9-CM) [ICD-9-CM 1993], which is used for health-care reimbursement in the United States; Medical Subject Headings (MeSH) [NLM 1999], which indexes the medical literature; and the Unified Medical Language System (UMLS) [McCray 1995], which provides a mapping among multiple source vocabularies [McCray 1995]. Other controlled medical vocabularies have evolved at sites that collect and store patient data. For example, the Computer-Stored Ambulatory Record (COSTAR) developed at the Massachusetts General Hospital in Boston contained a standardized vocabulary for clinical terms [Barnett 1982], and the Medical Entities Dictionary (MED) [Cimino 1995] is the vocabulary that integrates the names of data elements used in computer-based patient-care systems at Columbia-Presbyterian Medical Center in New York.

As institutions accumulate patient data, multiple parties develop a demand for those data. Because groups need to share data, there is a need to share the vocabulary that people use to represent those data.

When the developer of a computer-based patient-record system confronts the problem of choosing a controlled medical vocabulary, he typically finds that, although there are many controlled medical vocabularies from which to choose, there is no single existing vocabulary that completely suits his needs. The developer can create a new

vocabulary or use whatever vocabulary is available. On the one hand, unfortunately, creation of an entire controlled vocabulary that covers a large portion of medical care requires significant resources of time and expertise. On the other hand, if the developer chooses an existing vocabulary, he will face disadvantages that arise because the vocabulary was originally designed for a different purpose.

Patient-care providers, managers of health-care organizations, and clinical researchers need to share patient data. A single patient often has many providers, all of whom need to have access to the same data. Analysis of health-care practices within an organization requires aggregation of data from groups of patients cared for in a number of different settings. Similarly, epidemiological studies of disease patterns and clinical trials that evaluate the efficacy of medical treatments frequently require that patient data be combined from multiple sources. However, when data exist in disparate systems that use different concept representations, it is often prohibitively costly to translate data from one system to another, and research and clinical questions remain unanswered.

Thus, there are two primary reasons why people choose to share vocabularies: (1) they do not want to do all the development and maintenance of a vocabulary themselves; and (2) they want to share data among diverse applications without a costly or ineffective translation process.

1.3 The Problem of Local Divergence

If there is a shared vocabulary that many people want to use, people frequently adapt it for their own local purposes. Unfortunately, if both the shared vocabulary and the local vocabulary evolve independently, the problem of **local divergence** from the shared clinical vocabulary emerges.

People build vocabularies using other existing vocabularies, either because the existing vocabularies provide needed content or because the authors of a new vocabulary think that their system would be more valuable if it included a mapping to another commonly used vocabulary. In the first case, the vocabulary developers can adapt the existing vocabulary to their own needs and not worry about divergence of the two vocabularies—that is, until data managers decide that they want a system based on the original vocabulary to interoperate with a system based on the derived vocabulary. In the second case, where the goal is to have a mapping to an existing vocabulary to enhance the value of the new vocabulary, it is necessary to update the mapping to avoid disparity as both vocabularies evolve.

1.3.1 An Example of Divergence: The VA Lexicon

An example of a new vocabulary that was built from a preexisting vocabulary is the VA Lexicon, which is used for patient problem lists in the Veterans Administration Health Care System. The VA Lexicon was adapted from the Unified Medical Language System (UMLS) from the National Library of Medicine (NLM). The VA Lexicon was derived initially from the 1993 version of the UMLS. The developers of the VA Lexicon added their own terms, and the UMLS maintainers proceeded to make changes and to release new versions every year. By adopting the UMLS as the starting point for the VA Lexicon, the VA benefited from the work done by other vocabulary developers. However, as the UMLS changed and the VA Lexicon changed, the two vocabularies became dissimilar. Mapping them back to each other is a monumental task; so far, neither the VA nor the NLM has devoted the resources to make the two vocabularies compatible with one another. We can infer that, at this point, the cost of creating the mapping outweighs the perceived benefits of doing the job.

1.3.2 Update of a Local Terminology: A Report from the Trenches

If a local vocabulary depends on one or more external vocabularies maintained by other groups, then the local-vocabulary developers have the burden of accepting and implementing update information. For example, the MED at Columbia–Presbyterian Medical Center includes the names and codes found in ICD-9-CM because reimbursement for health-care services in the United States depends on those codes. Updates for ICD-9-CM are released annually.

The experience reported by James Cimino [Cimino 1996a, Cimino 1996b], who directs maintenance of the MED, no doubt is similar to what other clinical vocabulary maintainers will encounter. Cimino incorporated ICD-9-CM into the MED during initial development of the MED and regularly incorporates changes to ICD-9-CM. To share his strategy with other researchers and to raise awareness of the problems, he reported the details of the process he undertook and the difficulties he faced when he updated the MED with the 1994 version of ICD-9-CM.

Two crucial features of the MED are that a code for a concept keeps the same meaning always and that a code is never deleted, because historical patient data depend on those codes. There is an old rule that patient records should never be erased or altered. Surely, erasing or altering the meaning of codes would violate this principle. However, in

performing the update, Cimino found that the meanings of codes in ICD-9-CM occasionally did change, and codes sometimes were deleted entirely.

ICD-9-CM changes were limited to additions, deletions, name changes, and code changes. Cimino found these change operations to be too nonspecific for his purposes. For additions, he needed to distinguish among *simple addition* (addition of an entirely new concept), *refinement* (addition of a child concept to an existing concept), *precoordination* (combination of two existing concepts—for example, the attachment of a modifier to a base concept), *disambiguation* (creation of new terms to replace a previously ambiguous term), and *redundancy* (addition of a term that has the same meaning as a term that already exists). For deletions, he needed to distinguish between *obsolescence* (identification of a term that is no longer in use) and *discovered redundancy* (removal of a term that duplicates the meaning of another term). For name changes, he needed to determine whether the change was a *minor name change* (a change of name, but essentially no change in meaning) or a *major name change* (a change in name, and a change in meaning). In his taxonomy of changes, he also included *code reuse* and *code change*, which were operations allowed in ICD-9-CM, but not in the MED. He needed to distinguish among these 11 types of changes, because actions taken depended on change type.

Another problem with the update process was that the documentation distributed by the maintainers of ICD-9-CM provided only an overview of changes made since the previous year, and offered few details. Therefore, Cimino found that his best option was to do a line-by-line comparison of ASCII text of the 1993 and 1994 ICD-9-CM files for the full sets of codes and terms for each year. For this task, he used the "diff" program available on Unix systems. He then had to process manually each line that differed. Identification of the change type according to Cimino's taxonomy of changes facilitated the next step, which was to update the MED.

Thus, classification of change operations identified by ICD-9-CM maintainers was not what was needed, the change documentation gave inadequate explanations, and change files were difficult to process. Tuttle and Nelson, who are also familiar with the challenges of updating vocabularies [Tuttle 1995, Tuttle 1991], offered comments on the approach [Tuttle 1996]. They bemoaned the need to resort to a line-by-line comparison of ASCII text files, using a program such as "diff." They voiced a hope that, in the future, vocabulary maintainers will be more explicit about changes and will provide computer-processable change files. They said that they might be able to make use of Cimino's experience in updating external vocabularies to the UMLS, but expressed hope that such

an approach would be a short-term interim solution, and that vocabulary maintainers ultimately will report their changes in a more disciplined manner.

1.3.3 Tradeoffs Between Autonomy and Conformance

Administrators and clinicians at a local site typically want to have control over their patient data. They must make decisions about the database-management systems, hardware, and software applications used at their site. The nomenclature used for the data must be acceptable to local users. If the local site is a specialty clinic, then users may need more specialized concepts and terms than the shared vocabulary supports. Computer systems at the local site must support the kinds of queries that people at the site need to ask about the data. A local site's ability to adopt a shared vocabulary may be constrained by these local factors. There are additional costs in building consensus about vocabulary with other sites, and it is costly to make changes to local data, databases, and applications for the sake of conforming if those changes are not useful at the local site. Thus, the pressures to optimize systems and vocabularies for local use are significant.

Although people at a local site want to have autonomy over their data and vocabularies, managing and maintaining a large vocabulary is a time-consuming and expensive task that may require expertise in a variety of specialized subdomains. If a single individual is in charge of maintaining the vocabulary, that individual may not have expertise in all the necessary areas. If many people with differing expertise contribute to the vocabulary, they can provide broader and deeper content coverage, but then they face difficulties in ensuring that patterns are followed consistently and in avoiding duplication of concepts. If there is already a vocabulary available that has been built and is kept up to date by another organization, then people at the local site may prefer to use that product because doing so could save money and improve quality. Thus, there are tradeoffs between having complete control over a vocabulary locally, and having the work done by other people who may have more experience, resources, and expertise for the task.

1.3.4 Reasons for Shared and Local Change

Natural languages change over time, and the language of medicine is a classic example. Certain aspects of the language of medicine change more rapidly than others. For example, anatomic terms such as *hand* or *stomach* and physical-examination terms such as *hepatomegaly* or *systolic ejection murmur* have changed little compared to terms for procedures such as *cholestectomy*, which now includes both *laparoscopic*

cholecystectomy and *open cholecystectomy*, or tests that did not exist previously, such as *dobutamine stress echocardiogram* or *plasma HIV-1 RNA load*. We now know of diseases that were unheard of in the past, and as we learn more about known diseases, we give them new names that reflect our increased understanding. For example, 20 years ago, the term *acquired immune deficiency syndrome* had not yet been coined, and what was called *epidemic catarrhal jaundice* in a previous era is now recognized as *hepatitis A*. Other concepts are no longer useful; for example, *purpura variolosa* and *variola hoemorrhagica pustulosa*, terms denoting subtypes of smallpox, probably should not clutter the terminology we use daily. A terminology that is useful today will not be useful tomorrow if it is not kept up to date.

Factors that contribute to differences among health-care sites include differences in patient population, health-care providers, health-care resources, services rendered, management structures of health-care organizations, and local expertise. Because of these differences, local sites differ in the kinds of data that can and should be collected, the level of detail desired for data, and the terms that people prefer to use. Given these differences, people at a local site may decide that, although they want to benefit from the shared work of other vocabulary developers, they also want to be able to make their own modifications.

The shared-vocabulary maintainers may try to take into account requests by local sites, and may seek to modify the shared vocabulary in ways that will benefit the greatest number of local sites. However, if there are many local sites and numerous requests, no single site can expect the shared-vocabulary maintainers to respond to all its needs at any given time. Yet, if the changing needs of a local site are shared by many other local sites, changes that address those needs probably will be reflected eventually in the shared vocabulary. For example, when a new form of juvenile arthritis became epidemic in Lyme, Connecticut, clinicians needed a name for the illness locally before that illness was recognized in other parts of the country. With time, however, the term *Lyme disease* and terms for various Lyme-disease diagnostic tests became important for clinical care nationwide. A local site, however, may not be able to wait until a term needed locally is sanctioned by the shared-vocabulary maintainers.

1.4 Research Assumptions

Given the need for a shared clinical vocabulary and ongoing work to develop widely used vocabularies [Cimino 1994, Lindberg 1993, O'Neil 1995, Rocha 1994, Rothwell 1995], I assume in my research that, in the future, there will be one, or at least

one, clinical vocabulary that can be shared for patient care at multiple sites. Given that there are tradeoffs between local autonomy and conformance, my work is also predicated on the assumption that local divergence will continue to occur.

The local site needs the controlled vocabulary to support local needs and, in particular, to store and manage patient data. Commercial software developers who produce decision-support applications and programs that facilitate reporting of patient data would like to use a controlled vocabulary that is compatible with patient data at many local sites so that widespread distribution of their software is possible. If such applications conform to the shared vocabulary, but run on local patient data, mechanisms must be in place to ensure that the vocabulary of the local data does not diverge too far from the shared vocabulary.

When a shared vocabulary is available that serves the needs of many sites, a local site downloads a copy and makes changes as needed. When a new version of the shared vocabulary becomes available, the local site takes the shared-vocabulary log file and incorporates those changes into the local version.

1.5 Research Approach

Incompatibilities can arise that make it difficult to update the local vocabulary to ensure compatibility with the shared vocabulary. To make the task more manageable and to make it possible to automate partially the task, I propose that the shared and local vocabularies conform to the same vocabulary model and follow certain constraints. I have designed an *extended vocabulary model* to which the shared and local versions conform, and I have developed methods for partially automated support of *synchronization* of the local version with the shared version of the vocabulary.

1.5.1 Extended Vocabulary Model

The **extended vocabulary model** comprises

1. A *structural model* to which the shared vocabulary conforms
2. A *local extension of the structural model* to which the local vocabulary conforms
3. A *change model* to which the shared vocabulary conforms
4. A *local extension of the change model* to which the local vocabulary conforms

5. A *log model* that the shared vocabulary and local vocabulary use for storing change data in log files

Thus, I define an *extended vocabulary model* as a set of five submodels. The **structural model** specifies the data required to represent a concept. The **local extension to the structural model** adds flags that indicate whether a concept originated in the shared vocabulary, in the local vocabulary, or in the shared vocabulary that has been modified locally. The **change model** specifies allowable changes, with their inputs, effects, and constraints. The **local extension to the change model** adds change operations and constraints on certain changes according to a concept's site of origin. The **log model** specifies data that are tracked regarding changes made to the vocabulary. Although there is a local extension to the log model that adds a few data items needed to document local changes, the local extension does not affect the synchronization process, and I do not discuss it further.

I call the extended vocabulary model that I have developed CONCORDIA (CONcept and Change Operations for DIAlects). I have fully specified the semantics of the CONCORDIA model, and have implemented in software the operations that a developer uses to create and maintain the vocabulary.

1.5.2 Synchronization

Synchronization is the periodic process by which developers update the local vocabulary to obtain the benefits of shared-vocabulary updates, while maintaining local changes that serve local needs.

Synchronization is depicted in Figure 1.1. I propose that, to manage the problem of local divergence, local sites periodically update their local versions to include changes made to the shared vocabulary. In the figure, there are two phases shown in a series of changes. Each phase consists of modification of the shared vocabulary and modification of the local vocabulary during a given time interval, followed by synchronization.

At the beginning of the first phase, the shared and local vocabularies are equivalent; at the end of the first phase, the vocabularies are synchronized. The resulting vocabularies of the first phase become the inputs to the second phase when the individual vocabularies are again modified, and the local vocabulary is synchronized with the shared vocabulary.

The target state after synchronization is not a state of exact equivalence between the shared and local vocabularies; instead, the target state for the local vocabulary

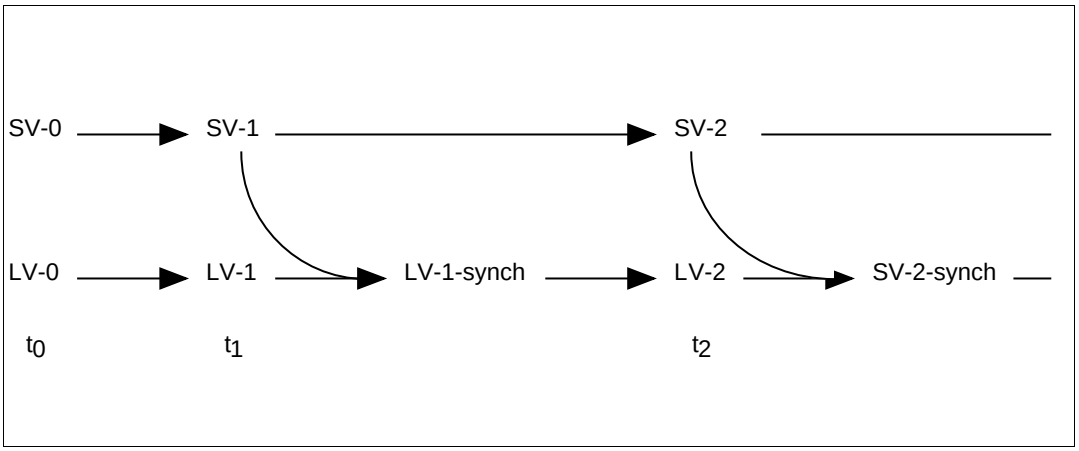

Figure 1.1. Synchronization process. The shared and local vocabularies (*SV* and *LV*) are identical at time 0 (t_0). They diverge at time 1 (t_1). The *LV* is synchronized with *SV*, and the process is repeated.

requires that the following conditions are fulfilled: (1) every concept in the shared vocabulary is represented in the local vocabulary, (2) every concept in the shared vocabulary has the same unique identifier that it has in the local vocabulary, (3) all subsumption relationships in the shared vocabulary are preserved in the local vocabulary, and (4) every attribute−value pair that is present in or inherited by a concept in the shared vocabulary must be present in or inherited by that same concept in the local vocabulary.

A **subsumption relationship** exists between concept *A* and concept *B* if *B* is a kind of *A*, or alternatively, if all instances of *B* are also instances of *A*. Preservation of subsumption relationships is required because an application that retrieves patient data may use a more general form of a term, although the particular data element uses a more specific term. For example, if the application searches for patients who have a history of gastrointestinal bleeding, then the program should be able to retrieve patients who have diagnoses such as *upper gastrointestinal bleeding secondary to esophageal varices* as well as *rectal bleeding due to hemorrhoids*. If the shared-vocabulary concepts and subsumption relationships are preserved in the local vocabulary, then the more specific patient-data elements will be retrieved.

1.5.3 System Architecture

Figure 1.2 displays the Concept-Manager system architecture. At the level of the shared vocabulary, there is (1) the shared-vocabulary concept repository; (2) the shared-vocabulary change-data repository; (3) the shared-vocabulary browser, which permits

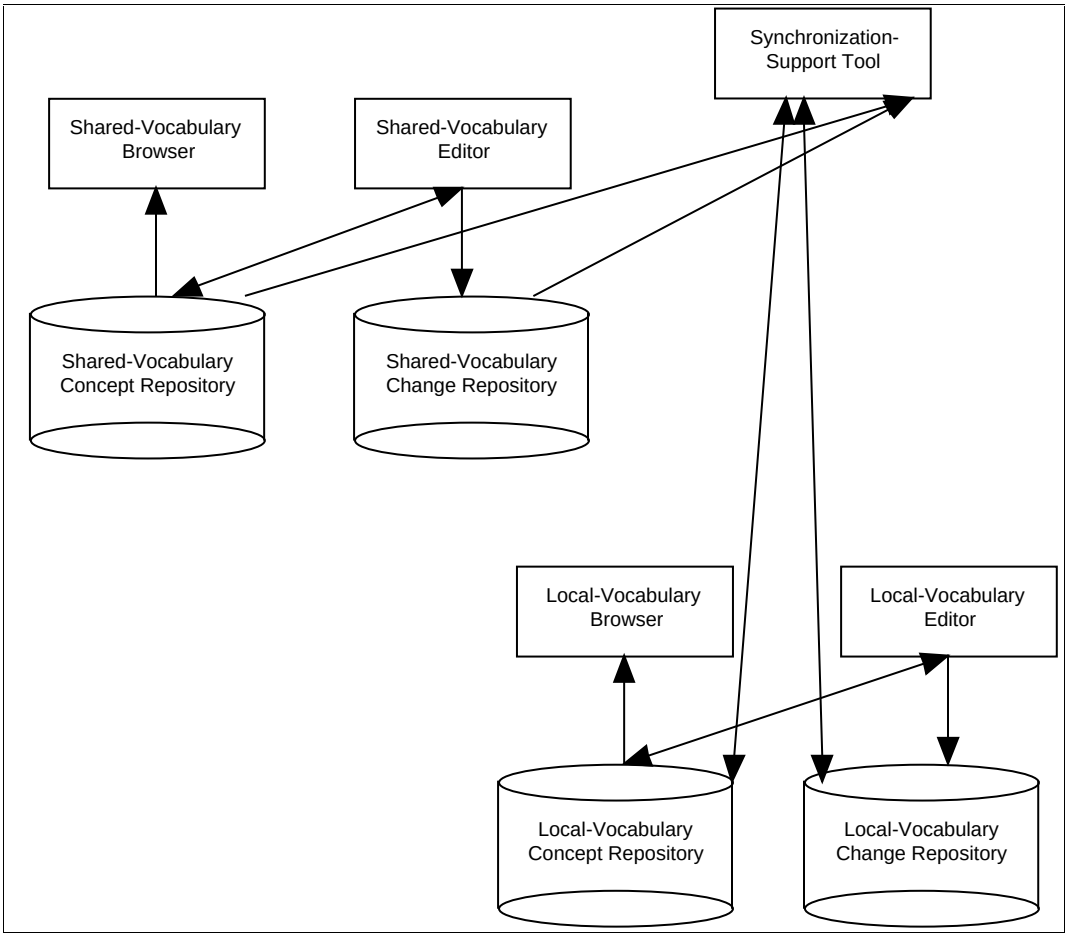

Figure 1.2. System architecture of Concept Manager. Concept Manager includes a suite of tools and concept and change-data repositories. Arrows indicate flow of data.

read-only views of the vocabulary; and (4) the shared-vocabulary editor, which allows read–write functions for updating the vocabulary. At the level of the local vocabulary, there is, analogously, (1) the local-vocabulary concept repository; (2) the local-vocabulary change-data repository; (3) the local-vocabulary browser; and (4) the local-vocabulary editor. In addition, at the local level, there is the **synchronization-support tool**, which enables the local maintainer to synchronize the local vocabulary with the shared vocabulary.

1.5.3.1 Concept and Change-Data Repositories

The vocabulary is stored in a concept repository, which could be implemented either as a database or as a file. Similarly, change data require persistent storage and could be implemented as a database or as a file. I chose to use text files as my persistent

storage medium for both concept data and change data in the shared vocabulary and in the local vocabulary.

1.5.3.2 Browsers and Editors

The browser and editor for the shared vocabulary run as separate applications because a vocabulary end-user would not have rights to vocabulary editing operations, whereas a vocabulary maintainer would have such rights. The browser application offers only search and display capabilities. The editor application permits the vocabulary maintainer to make changes to the vocabulary; it also contains all the browsing features that are present in the browser application.

The local-vocabulary browser and editor are similar in appearance and functionality to the shared-vocabulary browser and editor. The primary difference in the local-vocabulary browser is the ability to distinguish concepts that originated in the shared vocabulary from those that originated in the local vocabulary. The local-vocabulary editor enforces constraints depending on the concept or attribute site of origin; it also offers change operations that are unique to the local vocabulary.

1.5.3.3 Synchronization-Support Tool

The synchronization-support tool processes the shared-vocabulary log file from the most recent modification interval. The local maintainer uses the tool to monitor the synchronization process and to provide input when feedback is required. The tool can handle certain changes automatically, but for those changes about which the local maintainer should have a choice or for which there is a conflict that needs resolution by the local maintainer, the tool presents the essential information and requests the necessary data from the maintainer.

Figure 1.3 shows the user interface of the synchronization-support tool. The screen shown presents information about the removal, or retirement, of *Wassermann test* from the shared vocabulary. The *Wassermann test* was an old test used to diagnose syphilis; the test has been replaced by more modern tests. During synchronization of the local vocabulary with the shared vocabulary, the tool processes changes sequentially from the shared-vocabulary log. The system processes certain changes automatically, and provides recommendations for other changes about which the maintainer may want to have choices. In the example, the maintainer has the choice of retiring *Wassermann test*, because this term was retired in the shared vocabulary, or of preserving the term, because

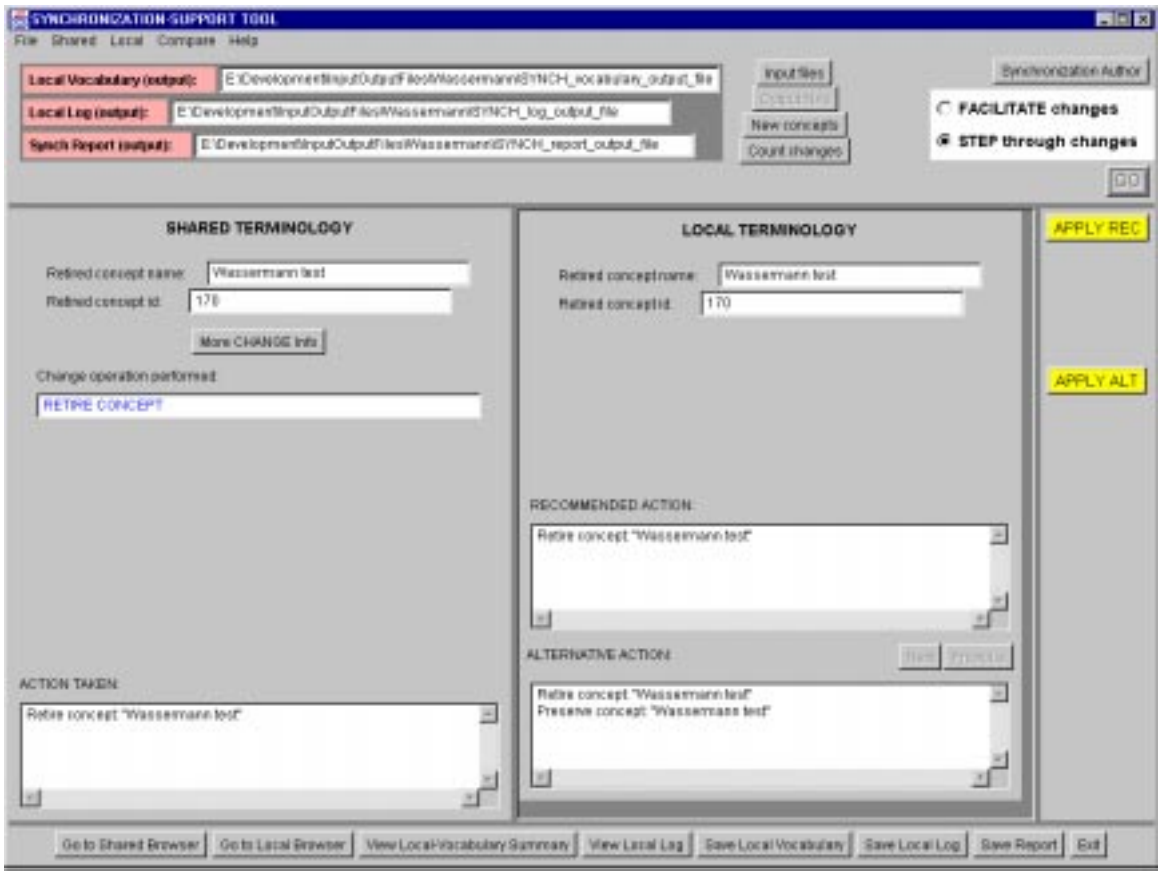

Figure 1.3. User interface of the synchronization-support tool. The tool recommends that the user retire *Wassermann test*, but also makes it possible for the user to retire and then to preserve the concept.

the local site still needs it. For either choice, the system makes the changes that are necessary to ensure structural consistency of the local vocabulary with the shared vocabulary. I discuss use of the tool in greater detail in Chapters 5 and 6.

1.6 Evaluation

To demonstrate use of the CONCORDIA model, I applied the model to an example of the synchronization process. I selected the subdomain of rickettsial diseases as the content area and used medical textbooks as the sources of content for different versions of a small medical vocabulary. I used a 1917 textbook entitled *The Diagnostics and Treatment of Tropical Diseases* [Stitt 1917] as the source for the initial vocabulary. I then used two modern textbooks of medicine as sources for divergent versions to which the initial vocabulary evolved. *Harrison's Principles of Internal Medicine* [Fauci 1998] and *Cecil Textbook of Medicine* [Bennett 1996] are two well-known textbooks of internal medicine. I randomly chose *Harrison's* and *Cecil* to be the source of the modified shared

and local vocabularies. First, I created the initial vocabulary according to content in the 1917 textbook. Then, I modified that vocabulary according to content in *Harrison's* to create the modified shared vocabulary, and I modified that same vocabulary according to content in *Cecil* to create the modified local vocabulary. Finally, I synchronized the local version with the shared version.

Thus, for this evaluation, I created a sample test set based on existing sources of medical knowledge in a single subdomain and demonstrated evolution of medical vocabulary in two divergent directions. I synchronized the local version with the shared version using the synchronization-support tool. I discuss the results of this process in Chapter 6.

1.7 Vocabulary Problems Not Addressed Herein

The topic of local variation of shared vocabularies is broad. Three areas that are related to my work—but that deal with problems I do not address—are (1) mapping of legacy systems to a shared vocabulary, (2) distributed development of a shared vocabulary by multiple domain experts, and (3) feedback to shared-vocabulary developers from local sites in the form of change requests.

A common situation is that many sites have *legacy systems*—systems already in place. Legacy systems pose a problem because they have their own vocabularies that are incompatible with other vocabularies, yet the cost and work to replace those systems is prohibitive. People at these sites may want to conform to a shared clinical vocabulary, but they may not want to incur the expense of creating and maintaining mappings. They would be delighted to have an application that could translate any legacy-system vocabulary to the shared vocabulary. Creating such an application, of course, would be no small task, since legacy systems vary markedly. Because a legacy system and a shared vocabulary are unlikely to follow my extended vocabulary model or to start out in a structurally consistent state, my specific approach does not apply. If, however, a local site were able to create and maintain a mapping to a shared vocabulary and to fulfill the assumptions of my approach, the model and methods provided in this dissertation would be relevant.

It is likely that development of a useful, large clinical vocabulary for the computer-based patient record will require contributions from many experts. Mechanisms that allow for distributed development without threatening to create incoherent organizational patterns within the vocabulary are essential, but are not the topic of my

thesis. Campbell proposed a strategy for resolving conflicts that result from distributed development of a shared controlled vocabulary by multiple contributing authors at different sites [Campbell 1996, Campbell 1998a].

The interactive relationship between the shared vocabulary site and the local sites is bi-directional. I consider the perspective of the local site and the flow of changes that goes from shared site to local site. I do not deal with what feedback the local sites give to the shared site, or with how the shared site manages that feedback. In the United Kingdom, for example, local sites that use the Read codes send requests for changes to the National Health Service for review and for possible incorporation into the national set of Read codes [Bailey 1999, Robinson 1997].

1.8 Summary

The lack of a standard for concept representation makes it difficult to map or to merge independently developed vocabulary systems, and the lack of a standard for change representation makes updates to mapped, merged, or divergent vocabularies difficult.

In this work, it is my goal not to set standards, but rather to analyze the commonalities and differences in existing vocabulary systems, and then to present a possible approach for concept modeling and maintenance that is based on the analysis of existing systems. I have created CONCORDIA, an extended vocabulary model that supports concept and change representation for local variants of a shared controlled vocabulary.

To evaluate my model, I produced a synchronization-support tool that depends crucially on a formal change model. I generated a test set of medical concepts based on knowledge contained in two contemporary textbooks of medicine and one out-of-date textbook of medicine. In demonstration of the value of a formal change model, I ran this test set through the synchronization-support tool and analyzed the results.

1.9 Guide for the Reader

In this chapter, I presented the problem of divergence of a local version of a shared vocabulary from the version from which it was derived. I stated my hypothesis and gave an overview of my proposed solution. I introduced CONCORDIA, and I defined terms—such as *extended vocabulary model, structural model, change model, log model,* and *synchronization*—that form the basis of my work.

In Chapter 2, I review the literature on the representation of concepts and change in existing controlled medical vocabularies and frame-based knowledge representation systems. Subsequently, in Chapter 3, I describe the CONCORDIA model, which is an extended vocabulary model that includes three component models—a structural model, a change model, and a log model—for both a shared vocabulary and a local vocabulary. To evaluate the usefulness of the change model, I have produced a methodology for synchronization that depends on the change model. In Chapter 4, I discuss the synchronization process and its underlying methodology. In Chapter 5, I describe the implementation of the software that I have developed to demonstrate the entire process of editing and synchronizing; in Chapter 6, I describe my test vocabulary and my experience with this test set in the synchronization-support tool. In the final chapter, I discuss the implications of my work and the relationship of my work to other work in the field. I outline the contributions of my research in the fields of medical informatics, computer science, and medicine. I conclude with a discussion of work that lies ahead.

2 Structural Models, Change Models, and Log Models in Existing Systems

To reach consensus on controlled-vocabulary standards, we need agreement on formal *vocabulary structural models* (representations of terms, concepts, and relationships in organized structures), formal *vocabulary change models* (representations of allowable changes), and formal *vocabulary log models* (representations of completed changes). Pleas for a standard vocabulary have been voiced many times [AMIA 1994, Hammond 1997]. Without consensus on such models, however, standards will remain elusive. Users and developers of health-care information systems would like a standard vocabulary that permits the exchange of data between systems without translation of data at either end, that allows decision-support systems built in one place to run on patient data stored in another, and that facilitates the reporting of data in the form in which the data were collected. The names and codes used by one system must be understood by another. However, just as standards for vocabulary content that provide an agreed-on set of comprehensible names and stable unique codes are important, standards for vocabulary structure, vocabulary change operations, and vocabulary log files are also essential if content is to be shared.

There are several reasons why a specification of change operations, which is provided by the change model, and a data model for log files, which is provided by the log model, are important. First, any software application that makes use of an evolving shared vocabulary must be able to recognize types of changes and to incorporate those changes easily. Second, if local-vocabulary maintainers want to keep up with changes made to the shared vocabulary, their local-vocabulary maintenance software must be able to incorporate changes from the shared site. Third, if local-vocabulary maintainers also make their own changes to the shared vocabulary, then there must be a mechanism in the local-vocabulary maintenance software for handling divergence of the shared and local versions. Thus, a common understanding of the change model and log model must be shared between software that generates and reports changes and applications that incorporate or adapt to those changes.

In this chapter, I review existing controlled medical vocabularies and computer-based systems that store vocabularies and knowledge bases. Controlled vocabularies contain knowledge and have similarities to knowledge bases. Knowledge bases contain

named concepts and have similarities to controlled vocabularies. There is significant overlap between the work that has been done in the medical-informatics community on controlled medical vocabularies, and that done in the computer-science community on knowledge-representation systems. The purpose of this review is to explore existing vocabularies and knowledge-based systems according to a framework of structural models, change models, and log models. In addition, I present an overview of articles from the medical-informatics literature and from the computer-science literature that address problems of sharing and local variation of vocabularies and knowledge bases. Based on this analysis of the current state of the art, I have developed CONCORDIA, a model that is based on prevalent trends in design, but that adds features to assist with local divergence. The CONCORDIA model is the topic of Chapter 3.

2.1 Controlled Medical Vocabularies and Frame-Based Knowledge-Representation Systems

Controlled medical vocabularies are prevalent in computer-based systems that support health care. Many of the current well-known vocabularies that have medical content useful for their intended purposes do not use formal approaches in the representation of concepts. On the other hand, frame-based knowledge-representation languages, including description logics, provide formal methods for representing concepts, but many are still only prototypes in research environments. Few systems based on framed-based knowledge-representation languages have been populated with medical concepts to the degree that is necessary for widespread use in health care. Developers of controlled medical vocabularies have expressed interest in adopting more formal approaches to concept representation such as those that have been popularized in the knowledge-representation community [Cimino 1994, O'Neil 1995, Spackman 1997, Tuttle 1994]. However, controlled medical vocabularies may not need all the features available in typical frame-based knowledge-representation systems.

Frame-based knowledge-representation languages are characterized by classes, slots, and slot values. Classes have slots, where slots are binary relationships between classes, and slot values. *Kind-of* relationships between classes are used to organize classification of hierarchies, and classes inherit slots from superclasses.

Description logics are also frame-based languages, but they typically call classes *concepts*, slots *roles*, and slot values *role values*. Description logics represent logical

Table 2.1. Examples of controlled medical vocabularies.

Popular Name or Acronym	Complete Name
ICD-9	International Classification of Diseases, 9th Revision
ICD-9-CM	International Classification of Diseases, 9th Revision, Clinical Modification
ICD-10	International Classification of Diseases, 10th Revision
CPT	Current Procedural Terminology
MeSH	Medical Subject Headings
DSM	Diagnostic and Statistical Manual of Mental Disorders
MED	Medical Entities Dictionary
LOINC	Logical Observation Identifiers, Names, and Codes
NANDA	North American Nursing Diagnosis Association
GALEN	Generalised Architecture, Languages, Encyclopaedias and Nomenclature in Medicine
UMLS	Unified Medical Language System
SNOMED	Systematized Nomenclature of Human and Veterinary Medicine
SNOMED RT	SNOMED Reference Terminology
Read	Read Clinical Classification
SNOMED CT	SNOMED Clinical Terms

connectors such as *and*, *or*, *for all*, and *there exists*. Implementations of description-logic systems generally offer automatic classification of concepts.

Although change is inevitable in controlled medical vocabularies that are in active use, formal models for change are lacking just as formal models for concept representation are lacking for these vocabularies. In contrast, frame-based knowledge-representation systems have formal underlying models for concept representation and have well-specified operations for queries and updates, but the developers have had minimal experience in managing the evolution of large populated systems. Therefore, the requirements for change and for the representation of change have received only

Table 2.2. Examples of frame-based knowledge-representation languages (including description logics), protocols, and specifications. Languages identified as frame-based knowledge-representation languages in this table are non–description-logic languages.

Name or Acronym	Type
KL-ONE	description logic
BACK	description logic
NIKL (New Implementation of KL-ONE)	description logic
Krypton	description logic
CLASSIC (CLASSification of Individuals and Concepts)	description logic
Loom	description logic
K-Rep	description logic
GRAIL (GALEN Representation And Integration Language)	description logic
Theo	frame-based knowledge-representation language
Protégé	frame-based knowledge-representation language
Ontolingua	frame-based knowledge-representation language
OKBC (Open Knowledge-Base Connectivity)	protocol for frame-based languages, including description logics
KRSS (Knowledge Representation System Specification)	specification for description logics

minimal attention by researchers in the knowledge-representation field, particularly with respect to health care.

Tables 2.1 and 2.2 list controlled medical vocabularies and frame-based knowledge representation languages that are familiar to the medical community and to the knowledge-representation community, respectively. In these tables, I make a distinction between GRAIL and GALEN, although the latter is based on the former. GRAIL is a knowledge-representation language, whereas GALEN is an implemented GRAIL system

that is populated with medical concepts, called the GALEN CORE model. Thus, I classify GALEN as a controlled medical vocabulary, that, unlike the other medical vocabularies listed, does follow a formal approach to concept representation (GRAIL).

A complete review of vocabularies and frame-based knowledge-representation systems relevant to or potentially relevant to medical care is beyond the scope of this review; therefore, I have chosen for discussion only a few existing systems. First, I compare structural models of selected controlled medical vocabularies and frame-based knowledge-representation systems, whose underlying structural models can be inferred from the literature. For controlled medical vocabularies, I have chosen ICD-9-CM (coding system used for health-services reimbursement in the United States), MeSH (thesaurus used for indexing the medical literature), the UMLS (controlled vocabulary that links multiple source vocabularies, developed by the National Library of Medicine), SNOMED (comprehensive medical nomenclature developed by the College of American Pathologists), and Read (coding system used for health-data reporting in the United Kingdom); for frame-based approaches, I have chosen CLASSIC, GRAIL, OKBC, and KRSS. CLASSIC and GRAIL are knowledge-representation languages. OKBC is a protocol that permits communication among different frame-based knowledge-representation systems; KRSS is a specification for description logics. SNOMED Clinical Terms is a new vocabulary that will combine SNOMED and Read. I do not discuss it further because there is little published literature on SNOMED Clinical Terms at the time of this writing. Next, I compare the change operations that are implied or explicit in controlled medical vocabularies and in frame-based knowledge-representation systems. For the discussion of change, I again consider ICD-9-CM, MeSH, the UMLS, SNOMED, Read, CLASSIC, GRAIL, OKBC, and KRSS. Finally, I present examples of change data from several controlled medical vocabularies to demonstrate the variability in data models and change-file formats. I have selected change files from ICD-9-CM, MeSH, DSM (system used for classification of psychiatric disorders), SNOMED, and the UMLS because they are readily available.

2.2 Structural Models

The structure of a vocabulary is determined by the elements that are available to content developers for the representation of concepts, and for the organization of concepts relative to one another. Therefore, choices made for the structural model depend on choices made for concept representation. In this section, I look at features pertinent to

Table 2.3. Comparison of features that relate to naming and identification of concepts.

Feature	I	M	U	S	R	CL	GR	OK	KR
Code	+	+	+	+	+				
Constant unique code			+	[a]	+				
Unique name	+	+	+	+	+	+	+	+	+
Synonyms		+	+	+	+	+			
Abbreviations									
Lexical variants			+	+					
Text definition		+	+					+	
Translation to other coding schemes			+	+	+		+		
Translation to other natural languages		+	+	+			+		

[a] Later versions of SNOMED have constant unique codes.
Legend:
I: ICD-9-CM; M: MeSH; U: UMLS; S: SNOMED III; R: READ Version 3;
CL: CLASSIC; GR: GRAIL; OK: OKBC; KR: KRSS.

concept representation, and discuss the variability of those features in different systems. First, I look at features that relate to naming and identification of concepts. Next, I consider features that provide ways of organizing concepts, such as hierarchical relationships and other types of relationships between concepts.

2.2.1 Naming and Identification of Concepts

Various features contribute to naming and identification of concepts in the different systems. Codes, names, and unique identifiers are common, but are not necessarily handled the same way in each system. Systems may or may not have synonyms, abbreviations, lexical variants, and/or text definitions. In addition, translations to other vocabularies or to natural languages may be a central task or may not be addressed. Table 2.3 compares features that relate to naming and identification of concepts.

2.2.1.1 Codes, Names, and Unique Identifiers

Many controlled medical vocabularies have both a code and a name to identify each concept uniquely. Codes are strings that typically contain numeric digits and that sometimes contain characters, hyphens, or periods. For example, ICD-9-CM, the UMLS,

SNOMED, and the Read codes have codes that serve as concept unique identifiers. In the UMLS, such codes are called concept unique identifiers (**CUI**s); in SNOMED, they are called **term codes**. Frame-based knowledge-representation languages, in general, do not use both codes and names that are visible to the user as unique identifiers. Instead, a concept name that could be a meaningful name or a code-like string may serve as the only unique identifier.

Users of vocabularies often assume that coded unique identifiers will remain constant in meaning over time. Developers of the UMLS and of the MED have emphasized and promoted the importance of this goal [Cimino 1998, Cimino 1996a, McCray 1995]. As described in Section 1.2.2, however, ICD-9-CM developers have not always kept the meaning of a code constant, and have reused old codes for new meanings [Cimino 1996a]. Reuse of codes for different meanings can cause problems if health-care providers record patient data with the codes, because the meaning of a patient data item should never change.

Nearly all vocabularies and frame-based knowledge-representation languages require concepts to have unique names. ICD-9-CM is an exception. Names such as *Other* and *Unspecified site* occur repeatedly. The UMLS guarantees unique names, but in certain cases an appended integer is required to create distinct terms. Because the term *cold* can mean either cold temperature or the common cold, the UMLS contains the concept name *Cold<1>*, which links to one concept unique identifier that means cold temperature, and *Cold<2>*, which links to a different concept unique identifier that means the common cold [Humphreys 1996a].

In ICD-9-CM, SNOMED III, and Read Version 2, the code not only serves as a unique identifier for a term, but also indicates where the term lies in a hierarchy. A disadvantage in using the identifier to indicate location is that doing so limits placement of the term to only one place in the hierarchy. For example, in ICD-9-CM, *acute pharyngitis* is located under *diseases of the respiratory system*, but *streptococcal sore throat* is located under *infectious and parasitic diseases*. Hence, there is no apparent relationship between *acute pharyngitis* and *streptococcal sore throat*. Another disadvantage is that this practice limits the number of levels in the hierarchy if the code has a fixed number of digits and each digit indicates a level. Using a code both to provide a unique identifier and to specify location in the hierarchy is now considered poor practice by vocabulary specialists in the medical-informatics community [Chute 1998, Cimino 1998, O'Neil 1995, Spackman 1998].

OKBC has an identifier called a **frame handle** that identifies a frame uniquely. Both classes and slots are frames and are identified by handles. Frame handles are required. OKBC makes it possible to have a **frame name** that identifies the frame as well, however there is an operator *frame-names-required* that may be set to *true* or *false*. When the value is *false*, frame names are not required, and may not be unique if they are given [Chaudhri 1998a]. CLASSIC and KRSS both have concept names and role names that are unique and that serve as the only unique identifiers. Such names can be arbitrary strings. A GRAIL **canonical form** is a unique name and serves as the unique identifier; however, this type of unique identifier is not an arbitrary string, or code. A user of a GRAIL system forms a GRAIL canonical form by naming the parent concept that subsumes the concept and by specifying the concept's attributes and attribute values in a structured way. An example of a GRAIL canonical form is the following:

Fracture *which*<

hasLocation Femur

hasCause (Osteoporosis *which* hasCause PostmenopausalChange)>.

This concept refers to a femur fracture caused by osteoporosis due to postmenopausal change [Rector 1997].

2.2.1.2 Synonyms and Abbreviations

Synonyms are common in controlled medical vocabularies and uncommon in knowledge-representation systems. Often, systems offer synonyms to facilitate search by users. Users of large medical vocabularies need to be able to find terms of interest, and alphabetized lists of terms are often not adequate. If high recall is more important than high precision for retrieval of terms, both exact synonyms and near-synonyms are useful. On the other hand, if a user expects two terms to have exactly the same meaning, because interchangeability in any context is a goal, synonyms should be more restrictive and only exact synonyms should be included in a synonym list.

MESH has **print entry terms**, which facilitate Medline searches. Print entry terms are alternate names for Mesh headings that make it possible for a user to find an article in Medline on a particular topic even if he enters a term that is not a MESH heading, but that is close in meaning to a MESH heading. In such cases, near-synonyms as well as exact synonyms are useful as print entry terms.

The UMLS stores multiple terms that map to the same CUI because the goal of the UMLS is "to facilitate the development of conceptual connections between users and relevant machine-readable information" [Humphreys 1993]. This goal led UMLS developers to provide multiple terms for the same concept in case the user enters a term that differs from the term used in a machine-readable source. The UMLS developers organized terms that mean the same thing by mapping them to the same CUI. However, terms that map to the same CUI may not be interchangeable in all settings. Campbell and colleagues pointed out that the concept named *aspirin* has the same CUI as *Ecotrin* [Campbell 1998b]. The former term is a generic name and the latter term is a trade name for an enteric-coated preparation that is less irritative to the stomach. In certain contexts, these two terms would be interchangeable because they both contain aspirin; however, the distinction may be important if a patient has trouble tolerating one and not the other.

In SNOMED, two terms are recognized as synonyms if they have the same term code. An extra 2-digit field, which is distinct from the term code, designates a term as a **primary term** (01) or as a **secondary term** (02). The primary term is also called a **preferred term**, and the secondary term is also called a **synonym**.

Most knowledge bases built in frame-based knowledge-representation languages are small, in comparison to controlled medical vocabularies, and synonyms have not been a high priority. For example, KRSS does not have a representation for synonyms. CLASSIC is unusual because it does have a specific representation for synonyms. OKBC permits the designation of a **pretty name** for a concept, but does not have a construct for a set of synonyms. A pretty name in OKBC does not have to be unique, but is intended for display on a user interface, and therefore, should be visually appealing and terse [Chaudhri 1998a].

Use of abbreviations is common in medicine, and controlled medical vocabularies typically store abbreviations as synonyms. Generally, there is no distinction between abbreviations and synonyms. CLASSIC, GRAIL, OKBC, and KRSS make no mention of abbreviations as separate recognizable entities.

2.2.1.3 Translation Between Coding Systems

Translation between coding systems is essential in health care if there are requirements or services that are crucial for patient care that depend on data coded in different standard coding systems. When using an electronic medical record, a provider may collect patient data using a clinical terminology such as the Read codes or SNOMED,

and then report data in ICD-10 or ICD-9-CM for epidemiologic purposes or for financial reimbursement. In addition, the provider may need to jump quickly to Medline, which is indexed with MeSH, to find the latest medical information relevant to the original patient data. The mapping between the coding schemes may not be one to one, and translation can result in loss of information.

One of the original goals of the UMLS was to link many coding systems together, and the UMLS continues to excel in this area. No other controlled medical vocabulary in existence today contains such an extensive set of links among multiple health-care coding systems.

SNOMED includes codes for diagnoses from ICD-9-CM to provide compatibility with billing codes.

Read codes Version 2 included cross-mappings to ICD-9. When Read Version 3 was created, ICD-10 had been released, and cross-mappings to ICD-10 were built in [O'Neil 1995].

2.2.1.4 Translation Between Natural Languages

As communication of information becomes increasingly important on a global scale, the need for translation among different natural languages increases. SNOMED III (also called SNOMED International) supports translation to multiple languages, such as French, Spanish, and German. A different type of translation is used in GALEN. Since a GRAIL concept name (a GRAIL description) may not be a name with which users are comfortable, because of its formal structured nature (consider, for example, the GRAIL description "Disorder *which* actsOn (Valve *which* isComponentOf Heart)" [Rector 1997]), more familiar names are necessary (e.g., "valvular heart disease"). GALEN researchers have developed tools that can translate formal GRAIL descriptions to natural language, a process that they call automatic generation of natural language [Rassinoux 1998, Rogers 1997]. Most other frame-based knowledge-representation systems have not dealt with translation between coding systems or natural languages.

2.2.2 Organization of Concepts

Features relevant to concept organization include hierarchies, *is-a* relations, non–*is-a* binary relations, and individuals. Table 2.4 compares systems by these characteristics. How concepts are organized affects the ease of maintenance of a

Table 2.4. Comparison of features that relate to organization of concepts.

Feature	I	M	U	S	R	CL	GR	OK	KR
Hierarchy	+	+	+	+	+	+	+	+	+
Subsumption hierarchy					+	+	+	+	+
Strict hierarchy	+			+					
Multiple parents		+	+		+	+	+	+	+
Binary relations		+	+	+	+	+	+	+	+
Named binary relations			+		+	+	+	+	+
Binary-relation hierarchy						+	+		
Primitive and nonprimitive concepts						+	+	+	+
Transitivity of roles							+		
User-defined facets								+	
Maximum cardinality						+		+	+
Minimum cardinality						+		+	+
Exact cardinality								+	+
Individuals						+	+	+	+
Inheritance					+	+	+	+	+
Conjunction of concepts						+	+		+
Disjunction of concepts									+
Negation of concepts									+
Conjunction of binary relations									+
Disjunction of binary relations									+
Negation of binary relations									+

Legend:
I: ICD-9-CM; M: MeSH; U: UMLS; S: SNOMED III; R: READ Version 3;
CL: CLASSIC; GR: GRAIL; OK: OKBC; KR: KRSS.

vocabulary. Hierarchical approaches are popular, and are useful both to end users and to maintainers of a vocabulary system. In this section, I use *parent* and *child* to refer to two concepts that are linked by an unspecified type of hierarchical relationship or by a clearly defined subsumption relationship in the main hierarchy, and *binary relation* to refer to a binary relationship that is not a hierarchical relationship in the main hierarchy.

2.2.2.1 Hierarchy

All the systems mentioned in this discussion have hierarchies of concepts, where concepts higher up in the hierarchy are more general than their descendants. However, the relationship between parent and child is not always named in controlled medical vocabularies. In MeSH, for example, the relationship may be one whose implied meaning is *is-a*, *part-of*, *has-location*, or *contains* [McCray 1995]. For example, *finger* is a child of *hand*, and *Chicago* is a child of *Illinois*.

In contrast, frame-based knowledge-representation systems, in which precise classification is of central importance, are rigorous about the relationship between parent and child: Every concept, except the top-level concept, must have at least one parent that subsumes it.

A number of controlled medical vocabularies have strict hierarchies, in which each concept can have only one parent. ICD-9-CM, SNOMED, and Read Version 2 use such hierarchies. Frame-based knowledge-representation systems typically are directed acyclic graphs that form polyhierarchies, in which each concept can be classified under more than one more general concept. The trend appears to be toward multiple classification, because the newer controlled medical vocabularies—including GALEN, the MED, Read Version 3, and SNOMED RT—permit multiple classification [Cimino 1994, O'Neil 1995, Rector 1997, Spackman 1997].

2.2.2.2 Binary Relations

Binary relations are more common in frame-based knowledge-representation systems than in controlled medical vocabularies. In knowledge-representation systems, binary relations are often called **roles** (e.g., KL-ONE [Brachman 1985], KRSS [Patel-Schneider 1993], and LOOM [MacGregor 1991]) or **slots** (e.g., Ontolingua [Gruber 1993] and OKBC [Chaudhri 1998a]). In certain cases (e.g., KRSS and CLASSIC) multiple-valued binary relations are called roles, whereas single-valued binary relations are called **attributes**. GRAIL uses the term attribute for all its binary relations [Rector 1997]. In each of these systems, binary relations are named.

ICD-9-CM is a controlled medical vocabulary that does not have binary relations to link concepts. Other vocabularies, such as MeSH and SNOMED, use binary relations called **cross-references**. For example, in MeSH, there is a cross-reference between “Neoplasms”

and "Precancerous Conditions." These terms are not synonyms, but they are related.[1] There is no name assigned to describe their relationship. Similarly, in SNOMED, there is an unnamed cross-reference between "botulism" and "Clostridium botulinum toxin." The UMLS has named binary relations, called **semantic relations**, which link **semantic types** in the UMLS semantic network. Every concept in the UMLS is assigned to one or more semantic types. However, if two semantic types are linked by a semantic relation, concepts assigned to those two semantic types are only potentially—but not necessarily—linked by the given semantic relation [Humphreys 1996a].

MeSH and SNOMED cross-references and UMLS semantic relations do not serve the same purpose that binary relations often serve in frame-based knowledge-representation languages. In frame-based knowledge-representation languages, when a set of binary relations and their values constitute a set of necessary and sufficient conditions for a concept, those conditions may be used for concept classification. Concepts that have necessary and sufficient conditions are called **nonprimitive concepts** in OKBC, **defined concepts** in CLASSIC and KRSS, and **composite concepts** in GRAIL. Concepts for which necessary and sufficient conditions cannot be defined are called **primitive concepts** in OKBC, CLASSIC, and KRSS, and **elementary concepts** in GRAIL. An elementary concept in GRAIL may have necessary conditions specified that are also used to classify a concept.

2.2.2.3 User-Defined Facets and Cardinality

A feature associated with binary relations in knowledge-representation systems is the **facet**, which provides information specific to a particular slot or role. In OKBC and in various languages for which OKBC provides a common protocol, the user can define facets. KRSS and GRAIL, on the other hand, do not permit user-defined facets. They both have a representation of cardinality, however, which is one of the predefined facets in OKBC. The general notion of a facet does not appear to have a counterpart in existing controlled medical vocabularies.

The **cardinality** of a binary relation is the number of values that the slot, role, or attribute can have. Cardinality may be specified for a particular relation for a particular concept, as it is in KRSS, or it may be specified for a relation regardless of which concepts

[1] MeSH also refers to links between headings and print entry terms as *cross-references*, but for purposes of this discussion, such cross-references are synonyms. The other type of cross-reference in MeSH links nonsynonymous, but related, headings and is indicated in MeSH manuals by the phrase *see related*. It is this nonsynonymous type of cross-reference that is analogous to slots, roles, and SNOMED's cross-references.

use that binary relation, as it is in GRAIL. In CLASSIC, KRSS, and OKBC, a role can have a maximum cardinality, a minimum cardinality, or an exact cardinality specified. Work in GALEN has revealed that such detail about cardinality frequently is not necessary for large medical vocabularies [Rector 1997]. Consequently, cardinality choices in GALEN are limited to *one* or *many*.

2.2.2.4 Transitivity of Roles

An interesting feature that is unique to GRAIL is the *specialisedBy* construct (or its inverse, *refinedAlong*).

For example, if GALEN contains the statement

hasLocation specialisedBy isDivisionOf,

then the system can infer

Fracture which hasLocation Femur

subsumes

Fracture which hasLocation (AnatomicalNeck which isDivisionOf Femur)

In other words, the system can infer that a fracture of the anatomical neck of the femur is a fracture of the femur [Rector 1997].

Rector and colleagues call this feature **transitivity of roles**. They describe the general case as follows [Rector 1997]:

A_1, A_2: attributes

E_1, E_2: entities (i.e., concepts)

V: attribute value

A_1 *specialisedBy* A_2 implies:

E_1 *which* A_1 V

subsumes

E_1 *which* A_1 (E_2 *which* A_2V)

The connection between *location* and *anatomic part-of* relations is important in medicine when vocabulary users need to determine subsumption [Bernauer 1994].

2.2.2.5 Individuals or No Individuals

A key difference between controlled medical vocabularies and frame-based knowledge-representation systems is that the former have only concepts (or classes), and no individuals (or instances), whereas the latter have both. An **individual** is a member of the set that the concept represents. The GRAIL language permits the creation of individuals, but the GALEN vocabulary developers chose to use only concepts [Rector 1997]. Brachman and colleagues pointed out that the decision where to draw the line between classes and individuals is dependent on the application [Brachman 1991]. The GALEN group states that "one of the earliest observations which led to GRAIL was that, in medicine, such arbitrary distinctions are often untenable even within a single application" [Rector 1997]. Thus, GALEN does not have individuals. Data about individual patients typically are stored in patient databases. Thus, there is a goal in medicine to separate concepts, which belong in a controlled vocabulary, from information about individuals, which belongs in a patient database.

2.2.2.6 Inheritance and Subsumption

Knowledge-representation systems allow concepts to inherit properties from concepts higher up in the hierarchy. The *is-a* (i.e., is-subsumed-by) relation is transitive: If concept A is subsumed by concept B and concept B is subsumed by concept C, then concept A is subsumed by concept C. Thus, concepts inherit the *is-a* links of their parents. In addition, concepts inherit roles from their parents and ancestors. In the original KL-ONE system, the role-value type of a role inherited by a concept could be **restricted** by that concept [Brachman 1985]. When the value of the role of a concept is restricted, its role value type is more specific than the role value type of the same role in the concept's parent or ancestor. KRSS and GRAIL both follow this approach. In OKBC, there is a distinction between slots that are inherited and slots that are not inherited.

Inheritance of hierarchical relations and nonhierarchical binary relations cannot be assumed in controlled medical vocabularies that are not rigorous in ensuring the type of hierarchical relationship. In SNOMED and MeSH, there is no implication that cross-references can be inherited. In the UMLS, inheritance of semantic relations cannot be guaranteed [Humphreys 1996a]. Hence, the more rigorous approaches taken by frame-based knowledge-representation languages makes inheritance and subsumption feasible and reliable, whereas the less rigorous approaches of certain controlled medical vocabularies makes it difficult to draw inferences about inherited properties.

2.2.2.7 Conjunction, Disjunction, and Negation

Conjunction, disjunction, and negation are logical *and*, *or*, and *not*, respectively. Most controlled medical vocabularies have no formal way either to combine terms or concepts with conjunction or disjunction, or to negate a term or concept through negation. Names of terms or concepts are precoordinated; thus, any combination must be represented explicitly as a combination, and a combination that is not explicitly represented cannot be considered part of the controlled medical vocabulary.

ICD-9-CM, MeSH, and the UMLS are vocabularies that have only precoordinated terms and do not have conjunction, disjunction, and negation. However, the developers of SNOMED describe SNOMED as a compositional language [Spackman 1998]. They state that terms can be combined to form new legal SNOMED constructs. The problem with this approach, however, is that there is often more than one way to create a new concept. For example, *low back pain* could theoretically be *low* + *back pain* or *low back* + *pain* or *low back pain*. With no constraints or rules for composition, different vocabulary users will generate different concepts that do not have equivalent unique identifiers, and interoperability will be lost.

The developers of the Read codes Version 3 targeted composition as a goal in their design. They used a template approach to constrain possible compositions. When both a precoordinated term and a compositional term have the same meaning, an explicit link between the two representations is stated.

The GALEN developers believe strongly in a compositional approach, and this feature is central to their choice of use of the GRAIL language. GRAIL, like other description logics, does permit conjunction of concepts. The GALEN researchers argue that it is important for the system to be able to recognize concepts with different GRAIL descriptors that are equivalent in meaning. GRAIL does not have disjunction or negation [Rector 1997]. Similarly, CLASSIC is a description logic that has conjunction of concepts, but does not have disjunction or negation. Negation is difficult because *not A* means everything except *A*, and the number of elements in *not A* may approach infinity.

KRSS is a system specification, which does include disjunction and negation as well as conjunction; for a system to be KRSS compliant, however, software developers do not have to implement all features of KRSS [Patel-Schneider 1993].

2.3 Change Models

It makes sense to talk about a change model for a vocabulary only in the context of the structural model of that vocabulary. However, it is instructive to learn from the work and experiences of other developers who have dealt with change in controlled medical vocabularies and who have designed and built knowledge-representation systems, despite the variability of structural models. In this section, I consider similarities and differences among change models in existing systems.

Frame-based knowledge-representation languages are implemented in software systems that permit users to build knowledge bases. The process of developing a knowledge base requires the ability to make changes. Thus, these software implementations must have specific change operations, and in certain cases, user's guides, published articles, or online resources are available that describe the meaning of these operations [Chaudhri 1998a, Rector 1997, Resnick 1993]. For most controlled medical vocabularies, on the other hand, editing software is available to only vocabulary developers, and detailed documentation generally is not available. To understand changes in controlled medical vocabularies, however, we can review the lists, tables, and natural-language text that document completed changes in each release. In this section, I discuss change operations that are either implied, as they are in controlled medical vocabularies, or explicit, as they are in frame-based knowledge-representation systems. Tables 2.5, 2.6, and 2.7 summarize changes in existing systems. Information is based on available literature, but in certain cases, whether or not a feature is present may be a matter of interpretation.

2.3.1 Additions

All systems have ways for users to make additions. The systems presented differ, however, in whether additions are additions of concepts or of terms. Other types of additions are additions of slots, roles, or attributes. Table 2.5 compares additions in the different systems.

For example, an *add* in SNOMED refers to the addition of a new term, which can be either a primary term (preferred term) or a secondary term (synonym). The distinction is specified by the use of a code (01 for preferred term, 02 for synonym). Thus, an *add* takes place if either the concept did not previously exist or if a new name is being added for a concept that did previously exist. In CLASSIC, on the other hand, adding a new concept

Table 2.5. Comparison of features that relate to creation or addition of new entities.

Feature	I	M	U	S	R	CL	GR	OK	KR
Create new concept	+	+	+	+	+	+	+	+	+
Create defined concept						+	+	+	+
Create primitive concept						+	+	+	+
Create disjoint concept	+					+			+
Create binary relation					+	+	+	+	+

Legend:
I: ICD-9-CM; M: MeSH; U: UMLS; S: SNOMED III; R: READ Version 3;
CL: CLASSIC; GR: GRAIL; OK: OKBC; KR: KRSS.

and assigning it an additional name or synonym are clearly separate operations. In CLASSIC, the developer uses the operation *cl-define-concept* to add a new concept and the operation *cl-define-concept-synonym* to add a synonym to an already existing concept. In GRAIL, addition of a new concept is done through a concept-forming operation that results in the creation of a new GRAIL canonical form. GRAIL also offers a separate construct *name* which allows the user to assign an additional name to a concept. In OKBC, the creation of a frame requires the use of one operation *create-frame* and assigning an additional name to that frame requires a different operation *put-frame-pretty-name*. A pretty name does not need to be unique. In KRSS, *define-concept*, *define-primitive-concept*, *define-disjoint-concept*, *define-role*, *define-primitive-role*, and *define-attribute*, provide ways to add concepts, roles, and attributes. *Define-disjoint-concept* permits the user to define a concept in which there is no overlap with another specified concept. An individual can be a member of one of the two disjoint concepts but not of the other. All codes in ICD-9-CM are intended to represent disjoint, nonoverlapping concepts.

2.3.2 Deletions or Retirement

Table 2.6 compares deletions and retirements in different systems. In SNOMED, a *delete* results in the removal of a term (either a preferred term or a synonym) from the set of terms in the vocabulary. In the UMLS, a concept is deleted when its CUI is deleted. A CUI is deleted when there is no longer any concept from any source vocabulary that is linked to that CUI. Deleted CUIs are reported each year in a change file. The MeSH developers also permit deletions, but name changes are more common than true deletions. In the Read codes, obsolete concepts are labeled extinct, but are not deleted from the vocabulary.

Table 2.6. Comparison of features that relate to deletion or retirement of entities.

Feature	I	M	U	S	R	CL	GR	OK	KR
Delete obsolete concept	+	+	+	+			+	+	
Label concept obsolete, but do not delete					+				
Delete binary relation								+	
Label binary relation obsolete, but do not delete									

Legend:
I: ICD-9-CM; M: MeSH; U: UMLS; S: SNOMED III; R: READ Version 3;
CL: CLASSIC; GR: GRAIL; OK: OKBC; KR: KRSS.

In systems that provide automatic classification, it can be more difficult to delete concepts. In CLASSIC, the authors of the manual state that "once a concept has been defined, its definition cannot be modified, and it cannot be deleted." [Resnick 1993]. Bechhofer points out that, to retract a concept in GRAIL, the user or system must remove other concepts that use the retracted concept in their defining or necessary criteria [Bechhofer 1994]. In addition, Bechhofer states that if a concept is retracted, then all its children will be retracted too [Bechhofer 1994].

In OKBC, *delete-frame* and *delete-slot* are valid operations. Using *delete-frame*, the user can delete a class, but there is no way to retire that concept without removing it by labeling it obsolete. The operation *delete-slot* first deletes the slot from all frames that contain the slot, and then deletes the slot from the knowledge base as well [Chaudhri 1998a].

In KRSS, there is no syntax specified for deleting a concept or a role [Patel-Schneider 1993].

2.3.3 Name Changes

Replacing a concept's name without changing the identity of that concept is possible in systems that have a separate unique identifier. In MeSH, the name can be changed, as indicated by the table *Replaced Medical Subject Headings with Replaced-By Headings* [NLM 1996, NLM 1997], but there is no code visible to the user that refers to both the old and new names. In GRAIL, since the GRAIL canonical form is the unique identifier, if that name changes, the concept becomes a new concept. GRAIL handles other unique-name changes through its ***name*** feature.

Table 2.7. Comparison of features that relate to modification of existing entities.

Feature	I	M	U	S	R	CL	GR	OK	KR
Replace unique concept name without changing concept	+	+	+	+	+	+	+		
Add/remove synonym		+	+	+	+	+		+	
Add/remove abbreviation									
Add/remove cross-mapping code			+	+	+		+		
Add parent/child	+[a]	+[a]	+	+[a]	+	+[b]	+[b]	+[c]	+[b]
Remove parent/child	+[a]	+[a]	+	+[a]	+		+	+[c]	
Add binary relation between concept 1 and concept 2		+		+	+		+	+	
Remove binary relation between concept 1 and concept 2		+		+	+		+	+	
Attach binary relation to concept without a value								+	
Replace binary-slot value				+	+			+	
Merge concepts			+						

[a] Modifications to hierarchical codes result in changes to parents and children (ICD-9-CM, MeSH, and SNOMED III).
[b] Parents and children may not be added directly, but new parents may be inferred when new concepts are added (CLASSIC, GRAIL, and KRSS).
[c] Children are not added or removed directly, but result from the addition or removal of parents (OKBC).
Legend:
I: ICD-9-CM; M: MeSH; U: UMLS; S: SNOMED III; R: READ Version 3;
CL: CLASSIC; GR: GRAIL; OK: OKBC; KR: KRSS.

In OKBC, there is an operation *put-frame-name*, which allows the user to change a name, but it is not necessary for the new frame object with the new name to be the same as the frame object with the old name. The philosophy held by the developers of the UMLS lies in direct contrast to this approach. In the UMLS, if the name changes, the concept does not, because the unique identifier remains the same.

In CLASSIC, the operation *cl-change-canonical-name* makes it possible to change a name, but there is no unique identifier available to the user to indicate that the new concept is the same as the old concept. There is an unusual operation in CLASSIC called *cl-remove-concept-names*, which removes all names in the list of names, and if the

canonical concept name is in the list of names to be removed, then the system will pick one of the synonyms as the new concept name. However, "there is no way for the user to know which of the remaining names will become the canonical one" [Resnick 1993]. In KRSS, there is no specific syntax for changing a concept name or role name.

2.3.4 Hierarchical-Relationship Changes

In SNOMED, there are **term-code reassignments**, which have the effect of changing a term's location in the hierarchy. In MeSH, the addition of a tree number has the effect of giving a heading a new parent (and possibly new children) and the deletion of a tree number has the effect of removing a parent (and children). In OKBC, there are operations *add-class-superclasses* and *remove-class-superclass*. There are no comparable operations for making changes to children, because the change is always specified in terms of superclasses, and the resulting subclass relationships in the opposite direction are inferred. In CLASSIC, GRAIL, and KRSS, a concept can acquire a new parent if a new concept is defined that subsumes the first concept. However, redefining an already-defined concept by giving it new parents directly is not possible in CLASSIC or GRAIL. In GRAIL, the addition of a necessary statement to a previously existing concept may have the effect of giving that concept a new parent.

2.3.5 Binary-Relationship Changes

SNOMED change files report modifications to cross-references between terms in lists of **reference corrections**. The MeSH manuals do not include separate tables of changes to cross-references, but such changes would have to occur if a heading to which another heading referred changed its name or was deleted. CLASSIC and KRSS do not permit modifications to roles and role values assigned to a particular concept after that concept has been defined. OKBC provides a number of operations on slots, such as *attach-slot*, *detach-slot*, *put-slot-value*, *put-slot-values*, *remove-slot-value*, *remove-slot-values*, and *replace-slot-value*.

2.3.6 Merges

The UMLS is the only system reviewed here that explicitly supports merges. A merge in the UMLS occurs when two concepts duplicate meaning, and one concept is merged into the other, followed by deletion of the unique identifier of one of the two concepts. It would be possible to perform merges in MeSH, SNOMED, ICD-9-CM, and Read; in those vocabularies, however, a merge is done by a series of other operations, and there

is no notion of merge itself. CLASSIC, GRAIL, OKBC, and KRSS do not have merge operations. However, in recent years, researchers in the knowledge-representation community have recognized the importance of merge operations because they have become interested in merging ontologies [Noy 1999b, Valente 1999]. I discuss that research further in Section 2.5.2.4.

2.4 Comparison of Log Models

There is no standard method for representing completed changes in controlled-medical-vocabulary systems. Change documentation may appear in published printed manuals or in electronic text files, and the choice of data elements and their organization is highly variable. Change descriptions occur in structured formats, such as lists or tables; in unstructured formats such as paragraphs of natural-language text; or in semi-structured formats that combine structured formats and natural-language text. To demonstrate the variety of methods that vocabulary maintainers currently use to represent change, I give examples from MeSH, DSM, SNOMED, and the UMLS. Little has been published by the knowledge-representation community on representation of completed changes.

I make a distinction between a log model and a data format. I use *log model* to indicate what kinds of data elements are in the log, how those elements are organized, and what they mean. In contrast, I use *data format* (or *data interchange format*) to mean the actual file format. Examples of data formats are Elhill format used by the National Library of Medicine, Abstract Syntax Notation 1 (ASN.1), and EXtensible Markup Language (XML) [Connolly 1997]. It is possible to infer a log model by analyzing the data format of a change file.

The highly structured tables of MeSH changes provide a good example of change data. Such tables are included with the annually released printed manuals of MeSH [NLM 1996, NLM 1997]. Since 1999, the change files have been distributed on the Web [NLM 1999]. The text files produced by the College of American Pathology for updates to SNOMED provide an example of changes presented both in structured tables (in tab-delimited format) and in semi-structured text. The changes to DSM are less structured; many of the changes are described in natural-language text. The changes are included at the end of DSM-III-R and DSM-IV, which are published as books [Frances 1994].

Finally, the developers of the UMLS distribute change files with each new version of the UMLS as electronic text files of simple structured tables (in vertical-bar delimited format), but the only changes reported are merges and deletions.

2.4.1 ICD-9-CM

ICD-9-CM , the coding system mandated for reimbursement of health-care services in the United States, is divided into **chapters** that classify diseases and injuries. There are **sections** (groups of three-digit code numbers), **categories** (three-digit code numbers), **subcategories** (four-digit code numbers), and **fifth-digit subclassificatons** (five-digit code numbers) [ICD-9-CM 1993]. A coded term is called a **rubric**. Associated with rubrics are additional explanations that, by convention, use terms such as *excludes*, which specifies conditions that should be coded elsewhere under another code, *code also*, which instructs the user that more than one code is required to describe the condition fully, and *use additional codes*, which suggests additional codes that the user might want to use. Figure 2.1 shows examples of rubrics in ICD-9-CM. Table 2.8 shows examples of changes.

427 Cardiac dysrhythmias
427.0 Paroxysmal supraventricular tachycardia
427.1 Paroxysmal ventricular tachycardia
427.2 Paroxysmal tachycardia, unspecified
427.3 Atrial fibrillation and flutter
427.31 Atrial fibrillation
427.32 Atrial flutter
427.4 Ventricular fibrillation and flutter
427.41 Ventricular fibrillation
427.42 Ventricular flutter
427.5 Cardiac arrest
427.6 Premature beats
427.60 Premature beats, unspecified
427.61 Supraventricular premature beats
427.69 Other
427.8 Other specified cardiac dysrhythmias
427.81 Sinoatrial node dysfunction
427.89 Other
427.9 Cardiac dysrhythmia, unspecified

Figure 2.1. Rubrics in ICD-9-CM from the cardiac-dysrhythmias category.

Source: *International Classification of Diseases, 9th Revision, Clinical Modification*, Fourth Edition, Practice Management Information Corporation, Los Angeles, 1993, pp. 212–213.

Table 2.8. Selected examples of changes made to ICD-9-CM.

Code	Description	Change Made
010	Primary tuberculous infection	Exclusions added
038	Septicemia	Exclusion added
077.9	Unspecified disease of conjunctiva due to viruses and Chlamydiae	Description revised Code invalid for Medicare after October 1, 1993
077.98	Due to Chlamydiae	Code added
077.99	Due to viruses	Code added
250	Diabetes mellitus	Exclusion added, and fifth-digit subclassfications revised and added
250.2	Diabetes with hyperosmolarity	Code revised
250	Diabetes mellitus with other coma	Description revised and exclusion added
438	Late effects of cerebrovascular disease	Use additional codes deleted, code also added

Source: *International Classification of Diseases, 9th Revision, Clinical Modification*, Fourth Edition, Practice Management Information Corporation, Los Angeles, 1993. From section entitled "Summary of Additions, Deletions, and Revisions to Volume 1," pp. 1367–1368.

2.4.2 MeSH

MeSH is the thesaurus of terms that is used to index the medical literature in Medline. MeSH has **headings**, which are the controlled terms, **print entry terms**, which are alternate terms that are listed with each heading, **non-print entry terms**, which are alternative terms that are not listed directly, **cross-references**, which point to other related MeSH headings, **scope notes**, which are text definitions, and **tree numbers**, which indicate where a heading is located in the hierarchical tree structures. A heading can appear in more than one tree and in more than one place in a given tree.

MeSH changes are presented in three text files: (1) new MeSH headings with scope notes, (2) replaced MeSH headings with replaced-by headings, and (3) MeSH tree-number changes. The first file, in Elhill format [NLM 1998], includes a variety of data elements, such as MeSH heading, MeSH tree number, MeSH scope note, annotation, history note, and backward cross reference (Figure 2.2). Data elements in the second file include the old heading, the new heading, and a flag that specifies the fate of the old term (Table 2.9).

Cloning, Organism

The formation of one or more genetically identical organisms derived by vegetative reproduction from a single cell. The source nuclear material can be embryo-derived, fetus-derived, or taken from an adult somatic cell.

X Cloning, Embryo
X Cloning, Human
X Embryo Cloning
X Human Cloning

Previous indexing:
Cell nucleus/transplantation (94-97)
Genetic Engineering (94-97)
Reproduction Techniques (94-97)

Figure 2.2. Example of a new MeSH heading with its scope note and previous indexing.

Legend:
X: Cross-references
Source: National Library of Medicine. *Medical Subject Headings, Annotated Alphabetic List, 1998*. Bethesda, MD, August 1997.

Table 2.9. Examples of replaced Medical Subject Headings with their replacements.

Replaced Heading	New Status	Replacement
Delta Agent	(P)	Hepatitis Delta Virus
Delta Infection	(P)	Hepatitis D
Dementia, Senile#	(P)	Dementia
Dementia, Presenile#	(P)	Dementia
Disabled		Disabled Persons
Heart, Mechanical		Heart-Lung Machine
Hodgkin's Disease	(N)	Hodgkin Disease
Zaire	(P)	Democratic Republic of the Congo

Legend:
#: deleted record
(P): replaced term continues to exist as a print entry term
(N): replaced term continues to exist as a non-print entry term.
Source: National Library of Medicine. *Medical Subject Headings, Annotated Alphabetic List, 1998*. Bethesda, MD, August 1997.

Table 2.10. Sample MeSH tree-number changes. These changes occurred in the 1997 release of MeSH. By deleting a previous tree number and adding a new tree number, *Mumps* was moved from one part of the tree to another. See Figure 2.3 for a hierarchical view of this change.

Heading	1996 Deleted Tree Number	1997 Added Tree Number
Mumps	C2.782.580.600.600.500	C2.782.580.600.680.500

Source: National Library of Medicine. *Medical Subject Headings, Tree Structures, 1997*. Bethesda, MD, August 1996.

1996	
Virus Diseases	C2
RNA Virus Infections	C2.782
Mononegavirales Infections	C2.782.580.
Paramyxoviridae Infection	C2.782.580.600
Paramyxovirus Infections	C2.782.580.600.600
Mumps	C2.782.580.600.600.500
1997	
Virus Diseases	C2
RNA Virus Infections	C2.782
Mononegavirales Infections	C2.782.580.
Paramyxoviridae Infections	C2.782.580.600
Rubulavirus Infections	C2.782.580.600.680
Mumps	C2.782.580.600.680.500

Figure 2.3. Change in hierarchical location of *Mumps* in MeSH between 1996 and 1997. See Table 2.10 for the MeSH representation of this change.

Sources: National Library of Medicine. *Medical Subject Headings, Tree Structures, 1996*. Bethesda, MD, August 1995.
National Library of Medicine. *Medical Subject Headings, Tree Structures, 1997*. Bethesda, MD, August 1996.

Data elements in the third file include the MeSH heading, added tree numbers, and deleted tree numbers. Table 2.10 shows an example, and Figure 2.3 shows a hierarchical view of that example.

The MeSH developers report the frequency with which certain changes are made. In 1997 (with a total of 17,895 terms), there were 350 descriptors added, 71 descriptors replaced with more up-to-date vocabulary, 60 descriptors deleted, and 560 print entry

terms added [NLM 1996]. The reports do not quantify the number of deleted tree numbers and added tree numbers.

2.4.3 DSM

DSM is a classification system that organizes mental disorders. DSM has **categories**, which are classes of mental disorders that are based on criteria sets with defining features [Frances 1994, Spitzer 1987]. Categories are associated with ICD-9-CM codes. In *Diagnostic and Statistical Manual of Mental Disorders, Third Revision Revised*, Appendix D is entitled "Annotated Comparative Listing of DSM-III and DSM-III-R" [Spitzer 1987]. This appendix lists the categories in the order in which they appear in the *old* edition, DSM-III, and identifies the corresponding new categories in DSM-III-R. If there has been a change, an explanation is given in natural-language text. Similarly in *Diagnostic and Statistical Manual of Mental Disorders, Fourth Revision*, Appendix D is entitled "Annotated Listing of Changes in DSM-IV" [Frances 1994]. In this appendix, disorders are listed in the order in which they appear in the *new* edition, DSM-IV, and annotates them with explanations of the ways in which the new categories were changed.

For example, in DSM-III-R, the category *Autistic Disorder* was annotated as follows:

> The distinction between the two specific DSM-III categories of *Infantile Autism* and *Childhood Onset Pervasive Developmental Disorder*, on the basis of age at onset, was judged to be not valid. Therefore, these two categories have been combined into the single category in DSM-III-R of Autistic Disorder. [Spitzer 1987]

In DSM-IV, the category *Anorexia Nervosa* was annotated as follows:

> This disorder has been moved from the *Disorders Usually First Diagnosed in Infancy, Childhood, or Adolescence* section to the *Eating Disorders* section of the Classification. [Frances 1994]

In these examples, two types of changes are identifiable within the natural-language text: (1) " ... two categories have been combined into the single ...," and (2) "This disorder has been moved ... " Other examples of phrases used to express the types of changes include "the name has been changed," "this integrates into one overarching category what were two categories," "this new category was added," "the duration

requirement has been reduced from 6 months to 3 months," "this has been omitted as a separate category," "the criteria set has been changed," "this disorder is new," "a specifier has been added," and "the frequency criterion of at least three times a week was dropped" [Frances 1994].

Because there is no consistent nomenclature used to identify change types, it is difficult to determine the frequency of each type of change. However, in DSM-IV, there is a listing of "New Disorders Introduced into DSM-IV," which lists 13 disorders, and a listing of "DSM-III-R Disorders Deleted from DSM-IV or Subsumed into Other DSM-IV Categories," which lists eight disorders [Frances 1994].

2.4.4 SNOMED

SNOMED is a systematized nomenclature that broadly covers human and veterinary medicine. SNOMED has **terms**, which may be **preferred terms** or **synonyms**, **term codes**, which indicate where terms are in the hierarchy, **ENOMENs**, which are integers that classify types of terms, and **cross-references**, which link SNOMED terms that are related by meaning. An ENOMEN may be *01* to indicate a preferred term, *02* to indicate a synonym of the preferred term, *03* to indicate a term with a more specific characteristic of the primary term, or *05* to indicate an adjectival form (e.g., *hepatic* for *liver*).

Types of changes in SNOMED include: **adds**, **deletes**, **text corrections**, **term-code reassignments**, **reference corrections or adds**, and **ICD adds or corrections**. Change data are presented in tables and lists that, in certain cases, include natural-language text. Tables 2.11 and 2.12 and Figures 2.4, 2.5, and 2.6 give examples.

Table 2.11. Examples of *adds* in SNOMED. Structured data for additions include term codes, date, ENOMEN (English term classification data item), term, and cross-references to other SNOMED codes. *Adds* refer to either preferred terms or synonyms.

Term Code	Date	ENOMEN	Term	Cross-References
D2-54126	01/96	01	Sick building syndrome	(T-20000) (C-00220)
D2-61140	01/96	02	Pulmonary haemorrhage	(T-28000) (M-37000)
D2-61100	01/96	02	Pulmonary oedema, NOS	(T-28000) (M-36300)

Legend:
01: a primary or preferred term
02: a synonym of the primary term
Source: Adapted with permission from file ADDS96.TXT on floppy disk SNOMED International Update No. 3 for Complete Dataset (July 1996).

Table 2.12. Examples of *text corrections* in SNOMED. Text corrections include removal of words from a term and changes of spelling (e.g., "Creutzfeld" -> "Creutzfeldt").

SNOMED Code	ENOMEN	Text Correction
D3-31532	01	"Atrial" flutter-fibrillation[1]
DE-3Bl020	01	Jakob-"Creutzfeldt" disease[1]
DE-3B020	02	"Creutzfeldt"-Jakob disease[1]
T-175B3	01	Deep flexor tendon of middle "third" finger (Remove "third")[2]

[1]Update No. 3 January 1996
[2]Update No. 4 August 1997
Legend:
01: a primary or preferred term
02: a synonym of the primary term
Sources: Adapted with permission from file CORR96.TXT on floppy disk SNOMED International Update No. 3 for Complete Dataset (July 1996) and from file CORR34.DOC (TEXT) on floppy disk SNOMED International Update No. 4 for Complete Dataset (August 1997)

Replace L-10038 - 03: Clostridium botulinum toxin for C-36304 - 01
Replace L-10038 - 03: Diphtheria toxin for C-36310 - 01

Figure 2.4. Examples of *term-code reassignments* in SNOMED. Documented data include term codes, ENOMENs, and terms.

Source: Adapted with permission from file CORR96.TXT on floppy disk SNOMED International Update No. 3 for Complete Dataset (July 1996).

DE-11310 01 Botulism
Replace (L-10038) for (C-36304)

Figure 2.5. Example from semi-structured list of *reference corrections* in SNOMED. Documented data include term codes, ENOMENs, terms, and structured text stating the change.

Source: Adapted with permission from file CORR96.TXT on floppy disk SNOMED International Update No. 3 for Complete Dataset (July 1996).

F-51530 01 Expectoration of sputum
Delete, confusing with sputum (T-20270)

D8-04400 02 Failed attempted legal abortion
Delete confusing

Figure 2.6. Examples of *deletes* in SNOMED. Documented data include term codes, ENOMENs, terms, and brief explanations of what change was made and why.

Source: Adapted with permission from file CORR96.TXT on floppy disk Update No. 3 for Complete Dataset (July 1996). © 1996 College of American Pathologists. All rights reserved.

2.4.5 UMLS

The UMLS contains multiple sources vocabularies and mappings among terms that have the same meaning. Terms that share the same conceptual meaning are linked by a concept unique identifier, or CUI.

With each new release of the UMLS, files called DELETED.CUI and MERGED.CUI are distributed. With these two files, users can determine whether a concept unique identifier that is no longer present in the new version was removed due to a deletion of the concept, or due to a merging of the concept with another concept. There is no explanation of why the changes were made. Examples of such updates are shown in Figures 2.7 and 2.8.

C0119634|Gly-Pro-Arg-Pro-Tyr|
C0119651|Gly-Tyr-Phe-Phe-Phe-Arg-Pro-Arg-Asn-NH2|
C0121341|hemoglobin Cleveland|
C0121366|hemoglobin La Roche-sur-Yon|
C0124739|KRRG|
C0124757|KT 6124|
C0146163|tobacco vein mottling potyvirus 34k protein|
C0146164|tobacco vein mottling potyvirus 42k protein|
C0150366|Surgical Assistance: Circulating|
C0150367|Surgical Assistance: Scrubbing|
C0150620|Hyperalimentation, NOS|
C0152022|BRAIN, HEMORRHAGE|
C0170962|unk gene product|
C0173192|SIOP protocol|

Figure 2.7. Examples showing the representation of deletions in the UMLS, as listed in the file named DELETED.CUI.

Source: National Library of Medicine. *UMLS Knowledge Sources*, 7th Experimental Edition (CD-ROM) January 1996, Disc 1.

C0000105|C0001964|
C0000267|C0000266|
C0000776|C0008489|
C0001070|C0014848|
C0001123|C0011880|
C0001167|C0000889|
C0001194|C0001193|
C0001336|C0086066|
C0001440|C0014177|

Figure 2.8. Examples showing the representation of merges in the UMLS, as listed in the file named MERGED.CUI.

Source: National Library of Medicine. *UMLS Knowledge Sources*, 7th Experimental Edition (CD-ROM) January 1996, Disc 1.

2.5 Management of Shared and Local Versions

Because ICD-9-CM is mandated for reimbursement of health-care services in the United States, there is high motivation for clinicians to conform to this coding system, and there has been little incentive to alter it for local needs. The Read coding system in the United Kingdom has been mandated for reporting data to the National Health Service. However, in the United Kingdom, since the Read codes are also used by clinicians for managing local clinical data, clinicians find it useful to make local modifications. The National Health Services permits local modification and reviews requests for changes that are submitted by local sites.

MeSH was created for the purpose of indexing the medical literature, and although it—or parts of it—may have been adopted and modified for other uses, there are few well-publicized examples of local modification. Thus, MeSH is a vocabulary that is shared by many people worldwide, but in the majority of cases, it is used for the same purpose—retrieval of articles from Medline. SNOMED has been used by pathologists, but is not yet in widespread use for electronic medical records. So far, little has been published about local modification at health-care delivery sites.

The developers of the UMLS have distributed the UMLS annually since 1990, and have encouraged its use for research purposes. The emphasis on research encouraged use of the UMLS for a variety of purposes at a variety of sites. The developers recognized that the UMLS would not cover all users needs—especially when it was first released—and would need enhancements.

Tuttle and colleagues wrote about the local update problem in an early paper entitled “Adding your terms and relationships to the UMLS Metathesaurus” [Tuttle 1991]. They described what they called “the local dilemma,” pointing out the problems that developers would face if they enhanced the Metathesaurus locally. They recognized that local enhancements would increase the burden of maintenance when new versions of the Metathesaurus were released, and predicted that the effort required to integrate local enhancements with Metathesaurus updates could easily exceed the effort required to add the local enhancements in the first place.

The idea of a shared vocabulary that everyone can use is appealing, because people believe that a shared vocabulary would permit everyone to communicate and share data, knowledge, and applications, and pooled resources could go toward the development and maintenance of the vocabulary. However, there are multiple reasons why local sites may differ in their vocabulary needs.

In an ideal world, the shared-vocabulary maintainers could serve the needs of all users. They could take into account all requests by local sites, and make updates to accommodate all users. If there were any discrepancies in this ideal world, the user community could find ways to achieve consensus. Unfortunately, however, no single local site can expect the organization that develops the shared vocabulary to respond to all its needs, and consensus among all potential users for all potential purposes is far from likely. Too much detail in a vocabulary may be costly for an organization, and too little detail does not serve intended objectives.

We may be tempted to conclude that the optimal design and content of a vocabulary depends strictly on the purpose of the application, and we might pessimistically believe that no standard is possible. However, different groups do have common or at least overlapping purposes in health care; if we can identify our common needs for structure and content, we can design and maintain a shared health-care vocabulary that serves the needs of many.

In the remainder of this chapter, I first describe methods for managing local customization that have been proposed or implemented by developers of controlled medical vocabularies, including Read, GALEN, the UMLS, and SNOMED. Subsequently, I explore work done by researchers in the knowledge-representation community, who have recognized the problem of shared use and local customization. Knowledge-representation researchers recognize that the cost of building knowledge bases is high and strive to

develop methods to make knowledge-base reuse possible. In working on problems of reuse, they have dealt with problems with sharing and local modification.

2.5.1 Methods Proposed by Developers of Controlled Medical Vocabularies

ICD-9-CM was initially released in 1975, when few clinical centers had any interest in maintaining a computer-based repository of concepts that would be compatible with ICD-9-CM and its annual releases. In contrast, many health-care vocabularies and corresponding vocabulary-management systems are now being developed to be used in computer-based systems. Vendors of electronic medical-record systems that feature structured data entry find that currently available vocabularies are inadequate, and their customers may demand support for local modification. Developers of the Read Codes and of electronic medical-record products that use the Read codes offer support for local modification, and developers of GALEN, the UMLS, and SNOMED have proposed methods for dealing with local variation as well.

2.5.1.1 Distinct Namespace for Local Codes

In a discussion of how the Read codes are updated in response to user change requests, Robinson and colleagues emphasized the importance of having a code namespace for local terms that does not overlap with the code namespace for official terms [Robinson 1997]. Users of the Read codes may add *temporary codes* or *local codes*. If a temporary code represents a concept that is later adopted in the national set, then a permanent code replaces the temporary code. Local codes accommodate site-specific needs. Such temporary and local additions have codes that are distinguished from official codes by a specific initial character.

2.5.1.2 A Generative Approach to Facilitate Local Changes

Rogers and Rector [Rogers 1997] argued that the generative approach of a vocabulary system such as GALEN offers more principled methods for users to make local extensions to the vocabulary than does an enumerative approach. If it is possible to compose existing concepts to generate new concepts, while following sanctions and other rules enforced by the system, then the local site may use new concepts immediately, rather than wait for updates from the shared vocabulary.

2.5.1.3 Central Coordination

In an earlier article about GALEN changes, Rector and colleagues speculated on methods for locally extending the GALEN CORE model [Rector 1995]. They envision a central coordinator who monitors local changes at the various sites to identify conflicts. The authors' basic premise is that changes that simply increase the level of detail could be made locally, but the local site should notify the central coordinator, whereas more drastic changes should be done only with careful control and coordination at the central site. Although the specifics of which situations would or would not require central control and what decisions the central coordinator would be expected to make are not fully delineated, their point is that if all sites share the vocabulary despite local changes, certain basic aspects of the overall framework must be retained.

2.5.1.4 Clarification of Missing Codes as Deletes or Merges

Olson and colleagues described the five change files distributed with the annual release of the UMLS Metathesaurus [Olson 1996]. These change files provide information that helps local sites to update their local copies of the Metathesaurus. Three change files list deleted codes: DELETED.CUI (for concepts), DELETED.LUI (for terms), and DELETED.SUI (for strings). The remaining two files list merged codes: MERGED.CUI (for concepts) and MERGED.LUI (for terms). The problem for local sites is that, if they compare the old version with the new version and find a former code missing, they do not know (without being told) whether the loss is due to a deletion of the code or to a merge of the original code into another code.

2.5.1.5 An Open Sharing Approach

Suarez-Munist and colleagues depicted a scenario in which multiple sites communicate openly with one another as they make local updates [Suarez-Munist 1996]. This approach assumes that all sites want to distribute their local changes and to adopt other sites' local changes. The MEME-II vocabulary-management software developed by the makers of the UMLS could support the updates. However, although this approach could work well for truly collaborative development with a few interacting groups or individuals who could also communicate in other ways, it might not be appropriate if there were hundreds or thousands of local sites. With many sites, differences in opinion may not be easy to resolve. Also, not every site will want every change from every other site, especially if there are conflicts. In addition, because the expense of updating vocabularies is high, groups that spend the most resources may not be willing to share

their work if they do not receive compensation. Alternatively, a group that spends few resources and is not conscientious about quality control could do sloppy work and produce errors that are propagated. Without a single organization in charge, there may be nobody to take responsibility for quality, and without a mechanism for distributing costs, there may be no incentive to produce good work.

2.5.1.6 Collaborative Development

Campbell and colleagues, in a study of collaborative development by physicians working in distributed locations, addressed conflict resolution [Campbell 1998a]. Unlike the maintainers of the UMLS, Campbell and colleagues strove to maintain a single coherent hierarchical structure that involved all concepts in the system, despite development by more than one maintainer. They identified two types of conflicts: (1) multiply defined term conflict (same term, different structured definitions) and (2) non–unique-definition conflict (same structured definition, different terms). They emphasized collaborative development of a *convergent* medical vocabulary, and used SNOMED as the basis for their work. In their approach, software supports identification of conflicts, and then developers must communicate among themselves to reach consensus when conflicts occur.

2.5.2 Methods Proposed by Researchers in the Knowledge-Representation Community

Work to design and build health-care vocabularies grew out of needs for managing terms for patient data in databases, for assigning standard codes for billing, and for retrieving literature from indexed systems such as Medline. As it has become clear that long lists of terms, with strict hierarchies, inconsistent types of parent–child relationships, and unlabeled nonhierarchical relationships, are not adequate for many clinical purposes, medical-informatics investigators have recognized the overlap with research in knowledge representation.

While developers of MeSH, SNOMED, and the UMLS were creating large sets of terms with valuable content during the past few decades, knowledge-representation researchers were focusing on concepts rather than on terms, and the hierarchies they designed were typically formal taxonomic structures to which algorithms for determining subsumption or for performing classification could be applied. In the early 1970s, Brachman designed the KL-ONE knowledge representation language, which employed concepts, roles, role values, and role restrictions in its approach to concept representation

[Brachman 1985]. Numerous other researchers followed his work by designing variants of KL-ONE in the 1980s. Each offered a slightly different approach, emphasizing one aspect or another, and a significant body of research accumulated. Examples of knowledge-representation languages are Krypton [Brachman 1983], BACK [Peltason 1991], CLASSIC [Brachman 1991], Loom [MacGregor 1991], and K-Rep [Mays 1991, Mays 1996].

In 1994, Cimino argued for following a knowledge-based approach to managing controlled medical vocabularies [Cimino 1994]; in 1997, Rector and colleagues described their implementation of the description logic GRAIL, which they populated with medical terms to form GALEN [Rector 1997]; and in 1998, Campbell and colleagues populated K-Rep with SNOMED terms [Campbell 1998a]. Gradually, the medical-informatics community has recognized the desirability of approaches more formal than those used in MeSH, SNOMED, ICD-9-CM, and the UMLS, but the goal to create systems that hold hundreds of thousands of medical concepts—as are found in the UMLS—far exceeds the scale of knowledge bases built in KL-ONE–like systems in the early days.

Description logics include systems descended from KL-ONE, such as CLASSIC, Loom, GRAIL, and K-Rep. Research on description logics continues as improvements in scalability, increased functionality, and methods for knowledge sharing are sought [Campbell 1998a, Horrocks 1999, Mays 1996, Swartout 1996, Valente 1999].

The word *ontology* has become popular, although its meaning is often vague. The term was borrowed from the discipline of philosophy where it means, according to *Webster's Dictionary*, "a particular theory about the nature of being or the kinds of existents" [Webster's 1991]. For the purposes of the knowledge-representation community, Gruber defined an ontology as a specification of a conceptualization [Gruber 1993], where a conceptualization comprises the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them [Genesereth 1987]. More simply, a set of concepts in a domain and the relationships among them constitute an ontology.

Swartout and colleagues recognized problems that arose with local use of a shared ontology in a DARPA-funded research initiative [Swartout 1996]. The goal of the project was to integrate systems developed by teams of researchers working in a common domain to produce a single larger and more capable system. The resulting ontologies apparently were not widely used.

Rather than growing with the systems in a tightly coupled fashion, ontologies were either released once and not extended, or versions of an ontology were made available at periodic intervals. If a particular system builder wanted to extend the ontology, it took a long time for his extension to be reflected in the shared ontology. Furthermore, when a new version of an ontology was released, developers were reluctant to take on the burden of updating their software to track changes in the ontology. As a result, the ontology and the implementations diverged, thus negating the advantages of a shared ontology [Swartout 1996].

Knowledge-representation researchers have found that the high cost of building knowledge bases impedes the development of large systems [Neches 1991]. Therefore, the problems of sharing, reuse, and local modification have become prominent research topics. Solutions proposed include translating between different knowledge representations, creating a common application programming interface (API), sharing a common syntax and semantics, merging and aligning ontologies, assembling modular ontologies by ontology inclusion, and comparing differentiated ontologies by employing description-compatibility measures. I discuss each of these approaches briefly.

2.5.2.1 Translating Between Different Knowledge Representations

Gruber and colleagues confronted the problem of heterogeneity of knowledge-representation languages by creating Ontolingua [Gruber 1993], a frame-based language based on the Knowledge Interchange Format (KIF). Translators have been written from Ontolingua to other representations, such as Loom, and the Interface Definition Language (IDL) of the Common Object Request Broker Architecture (CORBA) [Farquhar 1997]. Although the original goal of Ontolingua was to support translation between different knowledge-representation languages, automatic translation was sometimes difficult [Uschold 1998]. Perhaps a more important contribution of the work on Ontolingua has been the development of Web-based services for storing knowledge bases in a sharable repository on the Internet [Farquhar 1997].

2.5.2.2 Creating a Common Application Programming Interface

Chaudhri and colleagues developed OKBC as an API, to permit client applications to interact with different knowledge-representation systems in a standard way [Chaudhri 1998a, Chaudhri 1998b]. OKBC comprises explicit operations for creating, querying, and modifying an ontology. Its knowledge model makes use of frames, classes, slots, slot values, and facets. OKBC change operations were discussed in Section 2.3. If a knowledge

base is OKBC compliant, then an application can send OKBC-compliant queries to the knowledge base to retrieve knowledge. As long as the underlying knowledge base is OKBC compliant, it does not matter whether that knowledge base is written, for example, in Protégé [Musen 1998], Loom, or Ontolingua.

2.5.2.3 Sharing a Common Syntax and Semantics

Description-logic researchers dealt with the knowledge-sharing problem by specifying the syntax and semantics of an expressively powerful description logic, incorporating constructs that were generally accepted [Patil 1992]. This specification was KRSS [Patel-Schneider 1993]. In the specification, they included a minimal set of interface functions that permitted construction and querying of knowledge bases. Features included representation of defined and primitive concepts and roles; disjointness of concepts; conjunction, negation, and disjunction of concepts and roles; existential and universal role quantification; maximum, minimum, and exact cardinality of roles; role inverses; and assertions about individuals. It is not possible to design a system that contains all features and that runs efficiently, and therefore not all features must be implemented for a system to be KRSS compliant. However, the designers of the specification identified a core set of constructs and inferences that should be implemented, and provided additional constructs that could be implemented if desired.

2.5.2.4 Merging and Aligning Ontologies

Valente, Russ, and Swartout, using the description logic Loom, merged two ontologies that contained knowledge about air-campaign planning [Valente 1999]. They found that "the merging proved trickier than expected, because there were substantial structural differences between the two ontologies." They succeeded, however, in integrating the different viewpoints into an ontology that incorporated both views.

Noy and Musen developed an algorithm for merging and aligning ontologies, using the frame-based system Protégé [Noy 1999a, Noy 1999b]. In their approach, merging results in a single merged version of the original ontologies, whereas alignment results in the persistence of the two original ontologies, with links between them.

Mitra and colleagues developed a set of algebraic operators that take ontologies represented as graphs as inputs [Mitra 2000]. Operators include union, intersection, and difference. The union operator generates a unified ontology graph from two original ontology graphs.

2.5.2.5 Assembling Modular Ontologies

So that they could assemble collections of ontologies to build larger ontologies, Fikes and colleagues promoted the idea of *inclusion* of one ontology in another [Fikes 1997]. They implemented their approach using the Ontolingua formalism, in which an ontology is a vocabulary of nonlogical symbols and a set of axioms. Including an ontology A in an ontology B requires specifying a translation of the vocabulary of A into the vocabulary of B, applying that translation to the axioms of A, and adding the translated axioms to the axioms in the specification of B. Each ontology provides a local name space for the symbols defined in it, to avoid symbol conflicts.

2.5.2.6 Comparing Concepts by Description-Compatibility Measures

Weinstein defined *differentiated ontologies* as local versions that result when diverse participants adhere to a standard vocabulary on a general level, but are given the freedom to develop specialized meanings in local communities [Weinstein 1999]. In his work on description-compatibility measures, Weinstein assumed that an ontology would be represented in a description logic or in some other formalism that supports subsumption inference. Using a language that contained a subset of features from Loom and CLASSIC, he developed a number of measures that resulted in numeric scores to indicate semantic closeness of two concepts in two differentiated ontologies. He based his comparison methods on the structure of the ontology in the area of the concept of interest. Although he did not use the symbols assigned to concepts (e.g., concept names and synonyms) in his comparison algorithms, he believed that symbols were a potential source of additional information.

2.6 Summary

In this chapter, I surveyed a variety of controlled medical vocabularies and frame-based knowledge-representation techniques. I emphasized the ideas of structural models, change models, and log models. Controlled medical vocabularies typically do not follow formal models with clear semantics that are described explicitly and followed consistently. Nonetheless, many controlled medical vocabularies are rich in content, are in widespread use, and have been highly successful for the purposes for which they were designed. In contrast, knowledge-representation languages that can represent knowledge content in diverse domains, protocols that specify application-programming interfaces, and system specifications that describe common syntax and semantics for knowledge-representation languages in a particular family do follow formal models with clear

semantics. The languages, protocols, and specifications are described explicitly, and computer-based implementations follow the declared models rigorously. Nevertheless, medical ontologies built according to these models have not found widespread use in the delivery of health care.

When controlled medical vocabularies are designed for a particular purpose, such as for literature retrieval or reimbursement for health-care services, and there has been no goal to conform to a model that is useful for other applications, it is difficult to merge or share content. For example, the use of parent–child relationships that could mean *is-a*, *part-of*, or *located-in* in MeSH makes it difficult to reuse MeSH for purposes that require explicit subsumption relationships in a conceptual hierarchy. The use of a code that indicates a concept's location in the hierarchy and that also serves as a unique identifier makes it impossible to permit more than one classification of a concept. Such an approach works for ICD-9-CM if users use the hierarchy only to help them find particular codes, and get accustomed to where codes they commonly use are located. However, if an application needs to know that *streptococcal pharyngitis* is both a disease caused by streptococcus and a pharyngitis, then the vocabulary system cannot be reused for this other application that depends on multiple classification.

It is complicated to sort out these kinds of differences between vocabularies if their structures and underlying assumptions are not clearly expressed in a uniform manner. Hence, formally expressed structural models make it possible to understand differences in how controlled medical vocabularies are designed; were there agreement on structural models, it would be easier to merge and share content as well as to share software browsers and editors that manage content.

Because language reflects current thinking and behavior and our medical knowledge and health-care services are constantly changing, the language of medicine is constantly evolving. Therefore, even if we start with common structural models and are able to share content, we will run into trouble if we cannot share changes. A formal change model that specifies change operations and their semantics permits a shared understanding of change. A formal log model that specifies data elements for completed changes facilitates the sharing of log files. With agreement on a change model and a log model, different groups can write their own vocabulary editors and applications that incorporate shared changes.

Analysis of change files for ICD-9-CM, MeSH, SNOMED, and DSM shows that there is little consistency among change models and log models for these vocabularies. Data

formats for change files are also highly variable. Although knowledge-representation researchers have been careful to express their structural models and change models formally, they have little experience in representing shared changes for medical ontologies that are in actual use. Thus, we can learn from the history of changes made to controlled medical vocabularies about the types of changes that are needed and about how frequently such changes occur, and we can learn from the principles followed by the knowledge-representation community regarding formal approaches to change.

Problems of local modification have been recognized both by developers of controlled medical vocabularies and by knowledge-representation researchers. Developers and users of the Read codes probably have had the most experience in this area for actual patient care. However, changes to the Read codes at the local level are limited primarily to changes in names of concepts, and changes to the hierarchy generally remain under central control at the National Health Service [Robinson 1997]. Cimino also faced the problem in dealing with actual patient care and reported his solution [Cimino 1996a]. Other researchers in the medical-informatics community have speculated on or studied methods for handling local modification. UMLS, GALEN, and SNOMED researchers have published on this topic.

The high cost of creating knowledge bases and the widely accepted goal among computer scientists and programmers of developing software that can be reused have led knowledge-representation researchers to confront problems of sharing and local modification. Methods for translating between languages, using a common API, agreeing on syntax and semantics, merging or aligning ontologies, and predicting semantic closeness of concepts in differentiated ontologies are topics that have received attention.

Based on this analysis of structural models, change models, log models, and local modification, I created the CONCORDIA model, which I describe in detail in Chapter 3.

3 CONCORDIA Model

In this chapter, I describe CONCORDIA. The CONCORDIA model fulfills many of the ideas that health-care-vocabulary developers currently find useful in vocabulary design, but I do not claim that it is the ideal model. Vocabulary developers still have years of work ahead—some of which will be by trial and error played out in the marketplace—to acquire experience with applications and evolving health-care vocabularies. CONCORDIA is just one piece of the work that needs to be done. In particular, it introduces the idea of a local extension to a vocabulary model; it emphasizes the importance of explicit change models [Oliver 1999]; and it permits us to explore the process of synchronization.

In the design of CONCORDIA, I chose not to use the vocabulary model of the UMLS, because the UMLS has no subsumption hierarchy that links all its concepts. I chose not to use the models of MeSH, ICD-9-CM, and ICD-10 for the same reason. SNOMED III and Read codes Version 2 were not appropriate because each of these vocabularies has a strict hierarchy, and the code identifier determines a concept's location in the hierarchy. To be consistent with modern concepts in vocabulary design, I needed a hierarchy that permits multiple parents per concept, and unique identiers that do not determine hierarchical position.

GALEN, SNOMED RT, and Read codes Version 3 have subsumption hierarchies, allow for multiple parents, and have concept codes that are independent of hierarchical location. In addition, they allow for the specification of structured definitions of concepts. I wanted to include these features in CONCORDIA. However, for simplicity, I chose a somewhat simpler model than that of GALEN. SNOMED RT did not exist when I began my work. The vocabulary model of Read codes Version 3 is similar to mine; however, the Read system has templates that permit compositional concept generation [O'Neil 1995], and my model does not.

Based on the analysis of existing vocabularies and knowledge-representation methods, I have selected a set of concept data elements and constraints for the CONCORDIA structural model, a set of change operations and their semantics for the CONCORDIA change model, and a set of log-file data elements and constraints for the CONCORDIA log model. I describe those choices in this chapter, then conclude with an analysis of alternative choices that were considered.

3.1 CONCORDIA Structural Model

The CONCORDIA structural model consists of a shared-vocabulary structural model and a local-vocabulary structural model. First, I describe the data elements in the structural model. Then, I describe the organization of and constraints on those data elements.

3.1.1 Data Elements in the Structural Model

Table 3.1 shows the data elements in the shared-vocabulary structural model. CONCORDIA uses certain data elements to specify concepts and other data elements to specify attributes.

Every concept must have a **concept unique identifier** that remains unique and constant in meaning. In CONCORDIA, the unique identifier is a meaningless alphanumeric string (frequently called a **code**), rather than a meaningful word or phrase. Words or phrases that people use to denote a concept may change over time, but the concept itself does not change. If a patient data element is stored in a database using the concept identifier, the representation of the concept will always be valid, even if the name changes. Unique identifiers in CONCORDIA do not indicate the location of the concept in a hierarchy.

Table 3.1. Data elements in the CONCORDIA shared-vocabulary structural model.

	Data Elements
Concept Data	Concept unique identifier Concept name Concept text definition Synonyms Abbreviations UMLS code Parents Children Defining attribute set Concept usage status Retired parents Retired children
Attribute Data	Attribute unique identifier Attribute name Attribute text definition Attribute usage status

Every concept has a meaningful **concept name** that can be a word or a phrase. A concept name should be as explicit and as unambiguous as possible. Terms such as *cold<1>* and *cold<2>*, which are found in the UMLS [Humphreys 1996a], should be avoided and replaced, for example, with *upper respiratory infection* and *cold temperature*. Terms should be meaningful to a person who is knowledgeable about the subject matter. The concept name is unique at any particular moment: There are no two concepts that have the same meaning at the same time. However, the name of a concept may change over time, while its code stays the same.

Synonyms are optional and may be assigned to concepts. A synonym is an alternate name for the concept. Synonyms help users to search for coded concepts. Therefore, near-synonyms are permitted. Abbreviations are similar to synonyms in that they are used to facilitate search. However, they are distinct from synonyms and are maintained in a separate data structure. A synonym or abbreviation may be used for two different concepts. For example, *cold* could be a synonym for *upper respiratory infection*, and *cold* could also be a synonym for *cold temperature*. Similarly, *LAD* could be an abbreviation for *left anterior descending artery* as well as for *left axis deviation*.

A **concept text definition** is optional but desirable. A text definition is presented in unstructured natural language, like a dictionary definition.

A **translation code** serves as a link from the CONCORDIA-based vocabulary to another system that performs translation services. Implemented software based on the CONCORDIA model is not required to perform translation to other coding schemes. For medical vocabularies, the UMLS currently offers such services, and the translation code would be a **UMLS code**.

A subsumption hierarchy is defined in CONCORDIA by the specification of **parents** and **children**. The relationship between parents and children is always *is-a*. That is, if concept *P* is a parent of child concept *Ch*, then *Ch* is a kind of *P*. For example, *cardiovascular disease* is a parent of *coronary artery disease*, and *coronary artery disease* is a child of *cardiovascular disease*. Although children could be inferred from parents, CONCORDIA includes children in the concept model to facilitate more efficient retrieval of hierarchical information by an implemented system. Subsumption relationships are inherited down the hierarchy (i.e., subsumption is transitive). Ancestors and descendants are not explicitly included in the concept model, because, when necessary, ancestors and descendants can be computed recursively from parents and children by the implemented system.

The **defining attribute set** is a set of attribute–value pairs that define the concept. The set must contain necessary conditions, but it does not have to contain necessary and sufficient conditions. Attribute–value pairs are inherited down the hierarchy. The values can be restricted further at lower levels of the hierarchy as described in Section 2.2.2.6. What constitutes *defining* attribute–value pairs is a matter of judgment. The attribute–value pairs should contain information about the concept that is noncontroversial and that is inherently true.

Each attribute has a code that serves as an **attribute unique identifier**, and that remains constant in meaning over time. It also has an **attribute name** that is unique at any given moment, but that may change over time. The CONCORDIA model currently does not specify a hierarchy for attributes. However, GALEN researchers have shown that such a feature is useful, and we there are potential benefits to such an approach, especially if the set of attributes is large. In CONCORDIA, an attribute also has an optional **attribute text definition**.

So that change can be managed, a concept has a **concept usage status** and an attribute has an **attribute usage status**. The usage status may take on the value of *current* or *retired*. If a concept has a parent that is retired, that parent is removed from the concept's list of parents and placed in its list of **retired parents**. A similar construct is available for **retired children**.

The local-vocabulary structural model also includes the data elements **concept site of origin** and **attribute site of origin**. The site of origin indicates whether the concept originated at the shared-vocabulary level or at the local-vocabulary level. Data elements for shared-vocabulary parents (**SV parents**) and shared-vocabulary children (**SV children**) are also included to track required subsumption links.

3.1.2 Organization and Constraints

The **shared-vocabulary structural model** is specified by definitions for a shared-vocabulary concept, a shared-vocabulary root concept, a shared-vocabulary attribute, and a shared vocabulary; it also includes a set of assumptions that form constraints (Appendix A).

A **shared-vocabulary concept** *C* is a structure that contains the following required elements: (1) concept unique identifier (2) concept name, (3) concept usage status, and (4) set of parents that contains at least one element. *C* may also contain the following elements, but they are not required: (1) concept definition, (2) synonyms, (3)

abbreviations, (3) translation code, (4) children, (5) attribute–value pairs, (6) retired parents, and (7) retired children. Table 3.2 shows an example of a shared-vocabulary concept, *Ebola hemorrhagic fever*. Figure 3.1 shows a hierarchical view.

A **shared-vocabulary root concept** *R* is like a shared-vocabulary concept, but its set of parents is empty, and its set of retired parents is empty. An example of a root concept is a concept named "entity." All other concepts are hierarchical descendants of the root. There is exactly one root.

A **shared-vocabulary attribute** *A* is a structure that contains the following required elements: (1) attribute unique identifier, (2) attribute name, and (3) attribute usage status. *A* may also contain an attribute definition, but a definition is not required. There is no attribute hierarchy. An attribute–value pair of shared-vocabulary concept *C* is

Table 3.2. Representation of a CONCORDIA concept, *Ebola hemorrhagic fever*.

Concept Data Element	Value
Concept unique identifier	1000
Concept name	Ebola hemorrhagic fever
Concept text definition	a highly fatal hemorrhagic fever, clinically very similar to Marburg virus disease, caused by the Ebola virus (family Filoviridae), and occurring in the Sudan and adjacent areas in northwestern Zaire; the natural reservoir and mode of transmission of the virus are unknown, but secondary infection is by direct contact with infected blood and other body secretions and by airborne particles.[1, 2]
Synonyms	Ebola virus disease[2], Ebola disease[1]
Abbreviations	—
UMLS code	C0282687
Parents	viral hemorrhagic fever
Children	—
Defining attribute set	caused-by: Ebola virus
Concept usage status	current
Retired parents	—
Retired children	—

[1] *Dorland's Illustrated Medical Dictionary*, 28th edition. Philadelphia, PA: W.B. Saunders; 1994. Ebola disease, Ebola virus disease; p. 481.

[2] *Dorland's Illustrated Medical Dictionary*, 28th edition. Philadelphia, PA: W.B. Saunders; 1994. Ebola hemorrhaghic fever; p. 619.

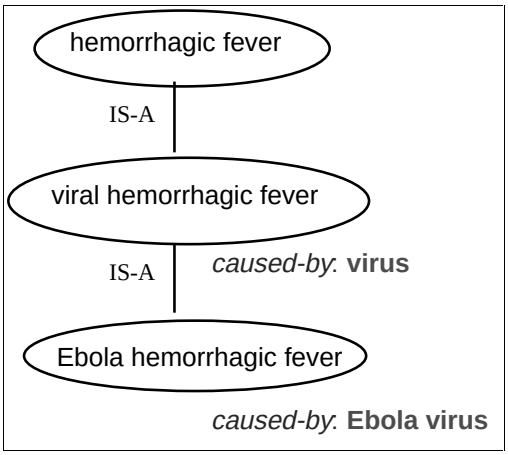

Figure 3.1. Hierarchy showing where the concept *Ebola hemorrhagic fever* would be located in a hierarchy. Hierarchical links are *is-a*; attribute–value pairs are shown.

a two-element set containing a shared-vocabulary attribute *A* and a shared-vocabulary concept *V* such that *C* is related to *V* by attribute *A.* An attribute of a concept may have more than one value (cardinality ≥ 1).

A **shared vocabulary** *SV* is the set $\{R, C_1, C_2, C_3, \ldots C_n, A_1, A_2, A_3, \ldots A_m\}$, where

1. *R* is a shared-vocabulary root concept.
2. $C_1, C_2, C_3, \ldots C_n$ are shared-vocabulary concepts.
3. $A_1, A_2, A_3, \ldots A_m$ are shared-vocabulary attributes.
4. For all concepts C_i belonging to *SV, R* belongs to the set of ancestors of C_i.
5. All shared-vocabulary assumptions hold true. These assumptions are
    1. A concept name is unique.
    2. A concept unique identifier is unique, and its meaning is constant.
    3. A concept may have a translation code (UMLS code) assigned to it, and more than one concept may have the same translation code.
    4. A synonym (or alternate name) may be shared by one or more concepts.
    5. The concept name is not the same as any synonym for the same concept.
    6. The usage status of a concept or attribute must be either *current* or *retired*.

7. There are no cycles between parent and child or between ancestor and descendant.

8. The union of attribute–value pairs of *C* and inherited attribute–value pairs of *C* is a superset of the union of attribute–value pairs of a parent *P* and inherited attribute–value pairs of *P*.

9. If an attribute–value pair of a concept *C* has the same attribute as an inherited attribute–value pair of *C*, then the value of that attribute–value pair is a descendant of the value of the inherited attribute–value pair. (Also see assumption 10.)

10. If a child concept shares the attribute *has-location* with its parent concept, then the value of *has-location* in the child may be related by a *part-of* relation to the value of *has-location* in the parent.

Figure 3.2 shows three types of parent–child relationships, which exemplify assumptions 8, 9, and 10 above. In the first example, *fracture of bone* is a child of *fracture*. The child has an additional attribute–value pair, *has-location:bone,* which is not shared by its parent, and assumption 8 holds. In the second example, since *fracture of*

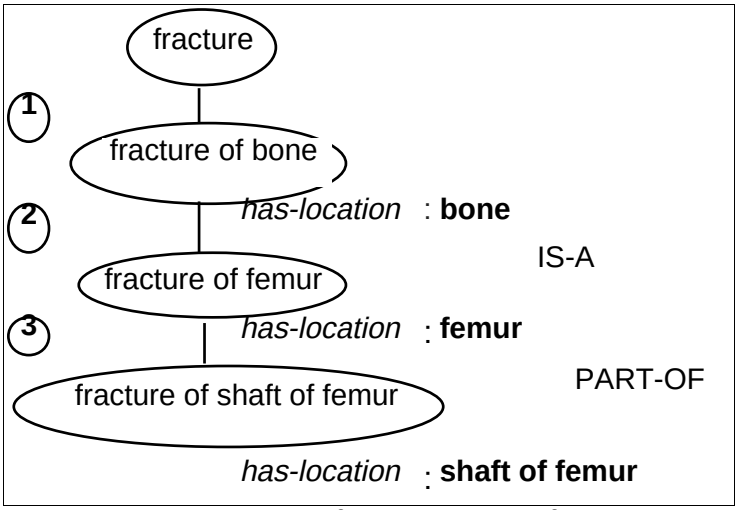

Figure 3.2. Examples of three types of parent–child relationships permitted in the CONCORDIA structural model: (1) a child has an additional attribute–value pair that its parent does not have; (2) a child has an attribute value that is related by an *is-a* relationship to the attribute value of the same attribute in its parent, and (3) a child may share the attribute *has-location* with its parent, and the value in the child is related by a *part-of* relation to the value in that parent.

femur shares the attribute *has-location* with its parent *fracture of bone*, and the attribute value *femur* is subsumed by *bone*, attribute value *femur* is subsumed by *bone*, assumption 9 holds, and a legal parent–child relationship is present. In the third example, *fracture of shaft of femur* shares the attribute *has-location* with its parent *fracture of femur*, and the attribute value *shaft of femur* in the child is related to the attribute value *femur* in the parent by a *part-of* relationship.

The **local-vocabulary structural model** is specified by definitions for a **local-vocabulary concept**, a **local-vocabulary root concept**, a **local-vocabulary attribute**, and a **local vocabulary**; it also includes a set of assumptions that form constraints (Appendix B). A local-vocabulary concept, a local-vocabulary root concept, and a local-vocabulary attribute are analogous to their shared-vocabulary counterparts, but each also has a **site-of-origin flag**. The site of origin indicates whether the concept originated at the shared site and has not been modified locally (a **shared concept**), the concept originated at the shared site but has been modified locally (a **locally modified shared concept**), or the concept originated at the local site (a **local-only concept**). A **local modification** is any change of concept name, definition, synonyms, abbreviations, translation code, parents, children, or attribute–value pairs. Ancestors, descendants, and inherited attribute–value pairs may change without turning a shared concept into a locally modified shared concept.

A local-vocabulary concept that originated in the shared vocabulary also keeps a list of its parents and children in the shared vocabulary: **SV parents** and **SV children**. The lists may differ from the local concept's own lists of parents and children.

A **local vocabulary** follows the rules of a shared vocabulary, but additional assumptions hold true. Local-vocabulary assumptions are

1. The usage status of a local concept is *current*, *retired*, *hidden*, or *preserved*.
2. The site of origin of a local concept is *shared*, *local only*, or *locally modified shared*.
3. If the site of origin of a local concept is *shared* or *locally modified shared*, then the usage status is *current*, *retired*, *hidden*, or *preserved*.
4. If the site of origin of a local concept is *local only*, then the usage status is *current* or *retired*.
5. There exist two distinct namespaces: (1) a local namespace and (2) a shared namespace. For each concept *LC* belonging to local vocabulary *LV*, if *LC*'s site

of origin is *shared* or *locally modified shared*, then *LC*'s concept identifier belongs to the shared namespace, and if *LC*'s site of origin is *local only*, then *LC*'s concept identifier belongs to the local namespace. For each concept *SC* belonging to shared vocabulary *SV*, *SC*'s concept identifier belongs to the shared namespace.

3.2 CONCORDIA Change Model

The CONCORDIA **change model** comprises a shared-vocabulary change model and a local-vocabulary change model.

The **shared-vocabulary change model** consists of a set of change operations (Appendix C). A change operation is a **valid shared-vocabulary change operation** if it is one of the change operations listed in Table 3.3. The complete specification of change operations requires a declaration of assumptions, input parameters, constraints, and effects. Appendices C and D describe all change operations in the shared and local vocabularies.

There are three types of change operations: (1) vocabulary change operations, (2) concept change operations, and (3) attribute change operations. A **vocabulary change operation** is a change operation that affects the structure of the vocabulary. It affects the presence or absence of concepts and attributes in the vocabulary. Additions, retirements, merges, and splits fall into this category. A **concept change operation** is a change operation that affects the values of data elements contained in a particular concept. For example, *replace concept name*, *add synonym*, *add parent*, and *delete attribute–value pair* are concept change operations. An **attribute change operation** is a change operation that affects the values of data elements contained in a particular attribute. *Replace attribute name* and *replace attribute definition* are the only attribute change operations.

Several change operations involve one or more steps. *Retire concept*, for example, involves labeling the concept as retired, leaving the retired concept where it is in the hierarchy when it was last used, and relinking the parents of the retired concept to the children of the retired concept. The retired concept is stored in the set of retired children of its parents and in the set of retired parents of its children. Figure 3.3 shows an example of *retire concept*.

Table 3.3. Valid shared-vocabulary change operations.

Type of Operation	Name of Change Operation
Vocabulary change operation	Add concept
	Retire concept
	Merge two concepts into one of the two concepts
	Merge two concepts into a new concept
	Split concept into two new concepts
	Add attribute
	Retire attribute
	Merge two attributes into one of the two attributes
	Merge two attributes into a new attribute
Concept change operation	Replace concept name
	Correct concept name
	Replace concept definition
	Add translation code (UMLS code)
	Delete translation code (UMLS code)
	Add synonym
	Delete synonym
	Add abbreviation
	Delete abbreviation
	Add parent
	Remove parent
	Add child
	Remove child
	Add attribute–value pair
	Delete attribute–value pair
	Replace attribute value
Attribute change operation	Replace attribute name
	Replace attribute definition

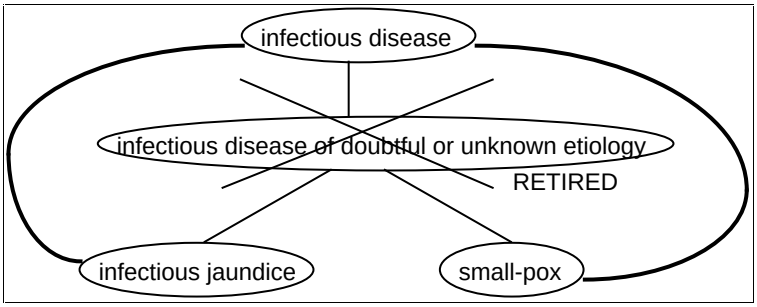

Figure 3.3. Process for *retire concept*. (1) Label concept *retired*. (2) Add retired concept's parents to list of parents of retired-concept's children. (3) Add retired concept's children to list of children of retired-concept's parents. (4) Remove retired concept from list of children of each parent of retired concept. (5) Remove retired concept from list of parents of each child of retired concept. (6) Add retired concept to list of retired children in each parent of retired concept. (7) Add retired concept to list of retired parents in each child of retired concept.

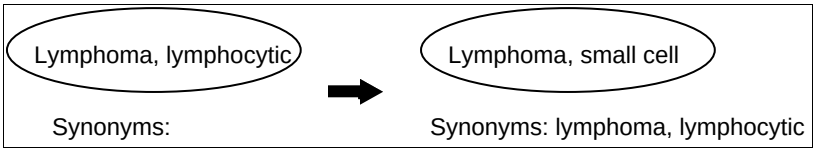

Figure 3.4. Process for *replace concept name*. (1) Change concept name to new name. (2) Add old concept name to list of synonyms.

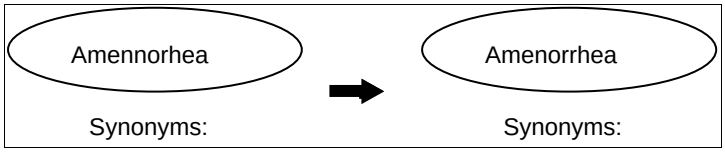

Figure 3.5. Process for *correct concept name*. (1) Change concept name to new name. (Do not add old concept name to list of synonyms.)

The operation *replace concept name* differs from *correct concept name* in that *replace concept name* automatically includes an *add synonym* operation to add the former name to the synonym list. Thus if the name changes, the old name is still accessible. However, if the old name was misspelled or was otherwise incorrect, it may not be appropriate to keep the old name in the synonym list. The change record will store the former incorrect name, but the change operation, *correct concept name,* will not

perpetuate the error in the synonym list. Figure 3.4 shows an example of *replace concept name*. Figure 3.5 shows an example of *correct concept name*.

A merge involves merging two concepts into one of the two concepts or into a new concept. In the former case, one concept is kept and one concept is retired; in the latter case, both of the original concepts are retired. The merged concept acquires the union of the parents, children, synonyms, abbreviations, and attribute–value pairs of the merged concepts. A split divides a concept into two new concepts, and the original concept is retired. For a split, the user must specify which parents, children, attribute–value pairs, synonyms and abbreviations go with which concept. Figures 3.6 and 3.7 show examples of the two types of merges. Figure 3.8 shows an example of a split.

In Figure 3.6, the two concepts *human T-cell lymphotropic virus type III* (*HTLV-III*) and *lymphadenopathy-associated virus* (*LAV*) were initially created as distinct concepts with distinct unique identifiers. When it became clear in the scientific community that

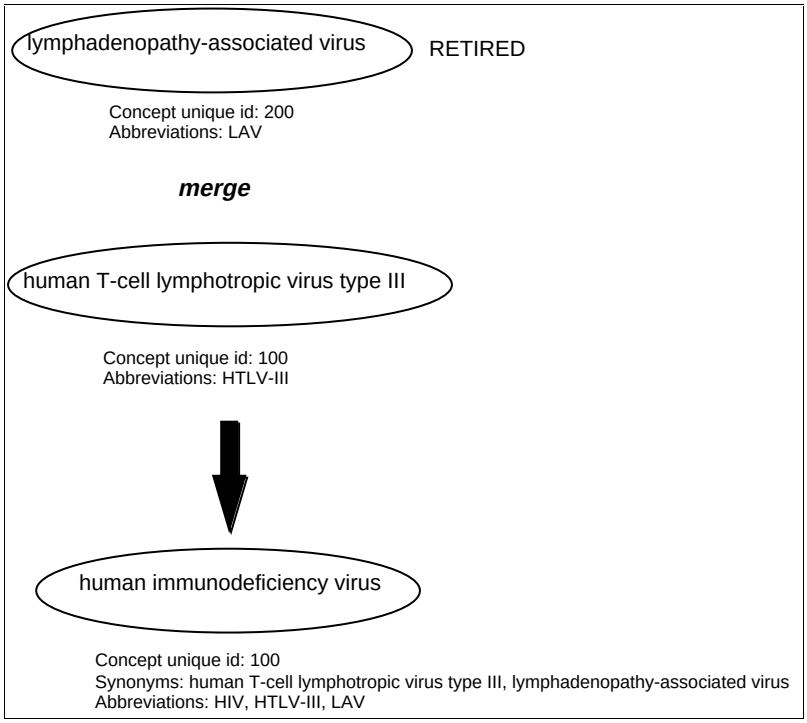

Figure 3.6. Process for *merge two concepts into one of the concepts*. (1) Select one of the two concepts to retain. (2) Label other concept *retired*. (3) Add retired concept's synonyms, abbreviations, parents, and children to retained concept. (4) Add retired concept's name to synonyms of retained concept. This example also shows the effect of an additional change operation that replaces concept name.

these two viruses were actually the same virus, the concepts had to be merged. In this sample case, the choice was to retain the concept with unique identifier 100 and to retire the concept with unique identifier 200. Hence, *lymphadenopathy-associated virus* was merged into *human T-cell lymphotropic virus type III*. The name *lymphadenopathy-associated virus* was added as a synonym, and *LAV* was added as an abbreviation. The merge could have been done in the opposite direction because the concepts were identical, and ultimately neither of the original names was retained. The virus was renamed *human immunodeficiency virus*. In the CONCORDIA model, the renaming step requires an additional change operation. The operation *replace concept name* is applied, and *human T-cell lymphotropic virus type III* becomes a synonym automatically. This type of merge is called *merge two concepts into one of the two concepts*. If it is appropriate to retain one of the original names, then the merge is not followed by *replace concept name*. References to the retired concept in attribute–value pairs are changed to references to the merged concept.

In Figure 3.7, a different type of merge is chosen. In this example, *acidosis, diabetic* is merged with *ketosis, diabetic* to create a new concept *diabetic ketoacidosis*.

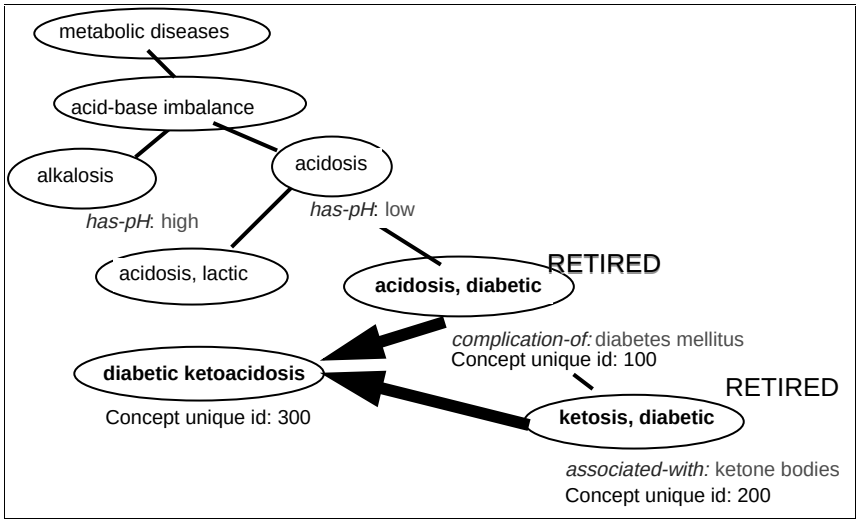

Figure 3.7. Process for *merge two concepts into a new concept*. (1) Label the two concepts *retired*. (2) Add new concept, selecting one of the parents of one of the retired concepts as the parent of the new concept, and select a new name. If one of the retired concepts was a parent of the other, then select a parent of the more general retired concept to be the parent of the new concept. (3) Add attribute–value pairs, remaining parents, and children to new concept. (4) Add retired concepts' names and synonyms to synonym list of new concept.

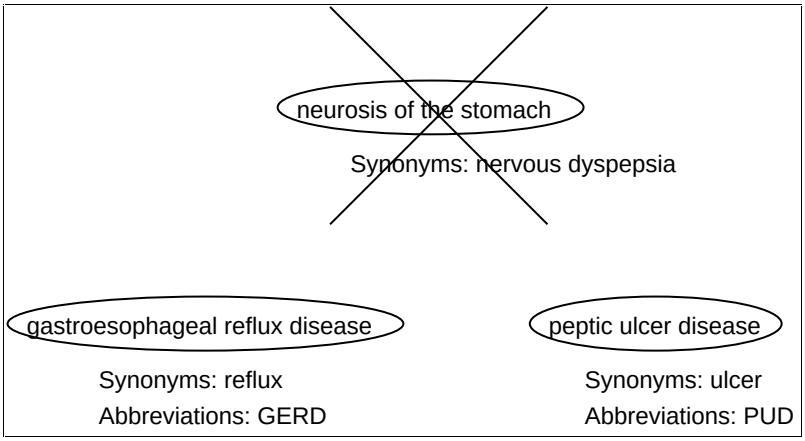

Figure 3.8. Process for *split one concept into two new concepts*. In this case, the synonym was not passed down to either of the two new concepts. This example shows the results of a split followed by application of *add synonym* and *add abbreviation* to each of the new concepts.

This type of merge is called *merge two concepts into new concept*. Whenever a new concept is formed in the CONCORDIA model, one parent must be assigned. A name and a unique identifier are selected for the new concept. The remaining parents, children, synonyms, abbreviations, and attribute–value pairs are added to the new concept.

Figure 3.8 shows a split. There is only one type of split: *split one concept into two new concepts*. In 1923, Osler and McCrae described a condition called *neurosis of the stomach*, or *nervous dyspepsia* [Osler 1923]. They stated that the condition presents a varying picture. In one form, the following symptoms occur:

> ... there are some symptoms apparently associated with hyperacidity especially in nervous individuals. They do not, as a rule, immediately follow the ingestion of food, but occur one to three hours later, at the height of digestion. There is a sense of weight and pressure, sometimes of burning in the epigastrium, commonly associated with acid eructations. [Osler 1923]

Other symptoms of neurosis of the stomach are as follows:

> The attack is usually independent of the taking of food, and may recur at definite intervals, a periodicity which has given rise to the supposition in some cases that the affection is due to malaria. The most marked periodicity, however, may be in the gastralgic attacks of ulcer. They

frequently come on at night. Vomiting is rare; more commonly the taking of food relieves the pain. [Osler 1923]

The former description paints a clinical picture similar to what clinicians today would call *gastroesophageal reflux disease*, and the latter description is more like *peptic ulcer disease.*

Suppose a vocabulary maintainer were updating the vocabulary. If *neurosis of the stomach* had become obsolete, and characteristics of the conditions *gastroesophageal reflux disease* and *peptic ulcer disease* had become clearer, it would be reasonable to perform a split. The concept *neurosis of the stomach* would be split into two new concepts, *gastroesophageal reflux disease* and *peptic ulcer disease*, and the original concept would be retired. There are no attribute–value pairs in this example, but if there were, the user would have to specify whether one or the other or both of the new concepts acquired the pairs. The user also would have to specify that the synonym *nervous dyspepsia* would not be passed to either of the new concepts.

The **local-vocabulary change model** consists of a set of change operations, where that set is a superset of the set of shared operations (Appendix D). A change operation is a **valid local-vocabulary change operation** if it is included in the list in Box 3.1.

A local maintainer can apply the operation *hide concept* or *hide attribute* to a concept or attribute that comes from the shared vocabulary, when that concept or attribute is not useful at the local site. Retiring a shared or locally modified shared concept is not permitted, if that concept is still current in the shared vocabulary. If a concept or attribute has been retired at the shared site, but is still needed by the local site, the local maintainer can apply the operation *preserve concept* or *preserve attribute*.

Box 3.1. Valid local-vocabulary change operations.

1. A valid shared-vocabulary change operation
2. Hide concept
3. Preserve concept
4. Hide attribute
5. Preserve attribute

3.3 CONCORDIA Log Model

The **shared-vocabulary log model** is a data model for completed changes in a shared vocabulary (Appendix E). If a valid shared-vocabulary change operation is applied to the shared vocabulary, then the **change record** contains a set of data elements that corresponds to that particular change operation. A **log** is an ordered sequence of change records. The model assumes that only one developer is making changes to the vocabulary at a time.

The set of data elements recorded includes **universal data elements** that are recorded for all change operations (change-operation name, timestamp, author, explanation, and assigned sequence number), and additional elements depending on the type of operation. As described in Section 3.2, there are vocabulary change operations, concept change operations, and attribute change operations. In addition to universal data elements, a vocabulary change operation requires only **change-specific data elements**. For example, a change record for the vocabulary change operation *add concept* would include the unique identifier of the added concept, current name of the added concept, unique identifier of the parent of the added concept, and current name of the parent of the added concept. However, a concept change operation also requires **concept-change data elements** to identify the concept (concept unique identifier and current concept name), in addition to universal data elements and change-specific data elements. The current concept name is included because the name may change from time to time, and the maintainer must have easy access to an identifier that is readily understandable. Therefore, a change record for the concept change operation *add synonym* would include concept unique identifier, current concept name, and added synonym. Similarly, an attribute change operation requires **attribute-change data elements** to identify the attribute (attribute unique identifier and current attribute name) in addition to universal data elements and change-specific data elements. A change record for the attribute change operation *replace attribute definition* would include attribute unique identifier, current attribute name, old attribute definition, and new attribute definition.

Examples of data elements recorded for particular change operations are given in Table 3.4 and Figure 3.9. Table 3.4 shows data elements recorded for *replace attribute value*, and groups those elements by type. Figure 3.9 shows a sample change record for *add child*. Appendix E gives the complete log model.

Table 3.4. Data elements in the CONCORDIA log model for the concept change operation *replace attribute value*.

Type	Name of Data Element
Universal data elements	Change-operation name
	Timestamp
	Author
	Explanation
	Sequence number
Concept-change data elements	Concept unique identifier
	Current concept name
Change-specific data elements	Attribute identifier
	Current attribute name
	Old value identifier
	Old value name
	New value identifier
	New value name

Type of change: Add child

Author: Diane E. Oliver

Timestamp: November 11, 1998, 10:30 a.m.

Explanation: *Dengue hemorrhagic fever* is classified as a viral hemorrhagic fever because it is caused by a virus (the dengue virus), and because, like other viral hemorrhagic fevers, it has clinical manifestations of fever, hemorrhagic manifestations, shock, thrombocytopenia, and neurologic disturbances. (See *Dorland's Illustrated Medical Dictionary*, 14th Edition. Philadelphia, MA: W.B. Saunders, 1994. Hemorrhagic Fevers; p. 619.)

Concept unique identifier: S3046

Current concept name: viral hemorrhagic fever

New child concept unique identifier: L5112

New child concept name: dengue hemorrhagic fever

Figure 3.9. Sample change record for concept change operation *add child*.

3.4 Data Interchange Format

Any data format that can be produced easily by vocabulary developers and that can be read easily by software at local sites should serve the data-sharing needs of the community. Essential factors for such a data interchange format are that the format must be agreed on, the format must faithfully follow the accepted data model, and the data must be structured consistently in ways that are readily computer processable.

Tab-delimited files that can be read into relational databases are easy to share, but there is no standard way to specify which fields correspond to which data elements. The LOINC vocabulary is distributed in tab-delimited files. Abstract Syntax Notation 1 (ASN.1) is another format that has been used to share data [Larmouth 1999]. The UMLS was distributed in ASN.1 format for a few years, but this practice was discontinued. The majority of users probably preferred the relational format, probably due to familiarity with and easy access to relational databases. Formats designed for a specific use, such as the Elhill format from the National Library of Medicine, are sharable if the format is explained to users, and if programs can be written easily to process these files. MeSH files are distributed in Elhill format. Currently, XML (EXtensible Markup Language) is a language that is popular and is becoming widely known [Connolly 1997]. Using XML, data-format developers can express the format specifications by means of a document type definition (DTD). If a community agrees on a DTD for a particular domain, data can be shared easily.

Due to the widespread acceptance of XML, and the ease with which a data format can be specified and communicated with a DTD, I chose to use XML in my implementation of the CONCORDIA model. I created a DTD that conforms to the CONCORDIA vocabulary structural model, and a DTD that conforms to the CONCORDIA log model.

Appendix F shows the DTD for the shared vocabulary. Appendix G shows the DTD for the shared-vocabulary log.

3.5 Summary

In this chapter, I described the CONCORDIA model. The design of CONCORDIA was directly influenced by the design of existing controlled medical vocabularies and frame-based knowledge-representation languages. In Chapter 2, I looked at the design of ICD-9-CM, MESH, SNOMED, Read, GALEN, DSM, CLASSIC, GRAIL, KRSS, and OKBC. There is no single design that encompasses all of these systems, nor do I believe it to be desirable for a single design to do so. Instead, I carefully selected features that appear to be beneficial

based on current trends. For example, in the transition from SNOMED III to SNOMED RT or from Read Version 2 to Read Version 3, the developers abandoned the use of codes to specify a concept's location in a hierarchy, they made the change from single classification to multiple classification of concepts, and they followed the trend toward specifying structured definitions for concepts using attributes and attribute values. The latter two features have been present for years in frame-based knowledge-representation systems, and have been important in GALEN since its inception.

I made other design choices for CONCORDIA that another designer might not have preferred. Certain choices were made for the sake of simplicity, and other choices were made simply because a choice had to be made, and the ideal choice is not yet clear.

For the sake of simplicity, I chose not to include data structures and associated change operations that would represent and facilitate changes for translation of terms in one natural language, such as English, to another natural language, such as French or Spanish. Clearly, support for such translation services would be essential for international communication.

In addition, I chose not to guarantee that automatic classification can be performed in systems based on the CONCORDIA model. In GRAIL and in other description logics, a concept must have necessary and sufficient conditions specified if it is to be classified automatically. In CONCORDIA, there is no way to represent whether a set of attributes and values associated with a concept is a set of necessary and sufficient conditions, or simply a set of necessary conditions. The set must contain necessary conditions in CONCORDIA; but if those conditions are also sufficient, there is no way to say so. Thus, the burden of classification lies to a greater extent on the user. Constraints in CONCORDIA limit the types of attribute–value pairs that can be assigned to a child concept, given the attribute–value pairs of its parent concept. These constraints facilitate classification, but do not guarantee that automatic classification can be performed. The benefit of this approach is that there are fewer restrictions on changes that can be made to the system. However, it is up to the user to be sure that changes made do not adversely affect the correctness of other concepts in the hierarchy.

The approach taken in CONCORDIA is similar to the approach taken in OKBC in that the OKBC model does not guarantee that an OKBC-compliant system will perform automatic classification. Because the burden lies with the user to make sure that the semantics dictated by the model are adhered to, change operations available in OKBC are relatively flexible, and it is easy to make changes to parents, children, slots, and slot

values. In contrast, the changes available in GRAIL, CLASSIC, and KRSS are more limited because it is computationally expensive to reclassify the entire knowledge base automatically whenever a small change is made.

Also for simplicity, I chose not to include features that would permit automatic concept generation from existing knowledge in the vocabulary. Templates in the Read codes provide a straightforward approach for specifying concepts that can be generated from component parts. Use of grammatical statements, sensible statements and other structures in GRAIL make it possible for the system to determine if a concept generated from component parts is a valid GALEN concept or not. I did not follow either of these approaches in the CONCORDIA model, primarily because of the difficulty in assigning unique identifiers to concepts that have been generated compositionally. However, this area is one where more research is needed. The huge numbers of concepts in medicine make it desirable to find ways to represent these concepts from their component parts, rather than to enumerate all concepts directly.

A feature of the GALEN model that is not shared by other controlled medical vocabularies or description logics is the ***specialised by*** construct. This feature is the generalization of the constraint on inheritance of an attribute that indicates anatomic location, in which the value of the attribute in the child is a *part of* the value of the attribute in the parent. An example of a situation in which this feature applies was the example of *fracture of shaft of femur* shown in Figure 3.2. Bernauer also wrote about the need for this type of construct for subsumption hierarchies in medical domains [Bernauer 1998]. The GALEN group generalized the effect for attributes other than anatomic location. I took a simpler approach to accommodate only anatomic location, but additional research would be valuable in this area.

I chose not to have a representation for an attribute hierarchy in CONCORDIA, not only because I wanted simplicity, but also because there is minimal experience with attribute hierarchies in controlled medical vocabularies. The GALEN developers have found an attribute hierarchy useful for medical concepts, and the vocabulary community may find such a feature necessary in the future as the number of attributes users need grows large.

In CONCORDIA, I chose to make the restriction that a unique identifier is a meaningless string. There are three reasons for this requirement: (1) if the unique identifier contains meaning about hierarchical location, there are disadvantages in classification, (2) if the code is composed of words that have meaningful connotations

and those connotations change with time, the code itself has an implied meaning that becomes obsolete, and (3) if a code contains characters or digits that contain meaning based on their position in the string, then certain codes could be invalid, and certain meanings could have no valid codes.

First, as I discussed in Section 2.2.1.1, use of the identifier to indicate location in the hierarchy has the disadvantage of limiting placement of a concept to only one position in the hierarchy. It also limits the number of levels in the hierarchy.

Second, if a word or phrase is the unique identifier and that word or phrase term becomes obsolete due to changes in commonly used language, then the meaning of the concept may be confusing. For example, if the term *Lues which hasComplication NeurologicImpairment* (following the GRAIL format for a hypothetical concept) is the unique identifier for neurosyphilis, the fact that *lues* is no longer a term that is readily recognized by the medical community makes the unique identifier confusing. For another example, suppose the term *non–insulin-dependent diabetes mellitus* is the unique identifier assigned to one of the two most common types of diabetes mellitus. Later, the term is changed to *type 2 diabetes mellitus* because the original term is confusing: although patients with this condition do not depend on insulin for immediate survival, they may depend on insulin for glucose control. If the medical community begins to use of the newer term, the vocabulary looks out of date if the older term must be retained as the unique identifier.

Third, if a code contains characters that have meaning in certain positions, then particular combinations of characters could be invalid if combinations of the implied meanings make no sense. Also, if meaning is specified by the choice of alphanumeric character in a particular position in a code, then the number of possible meanings is limited by the finite number of possible characters. Therefore, certain codes might not have meanings, and certain meanings might not have codes.

Due to the relative lack of experience with formal change operations in the medical-vocabulary community, and lack of standards, I made several arbitrary choices in the change model. For example, I chose to make two separate operations: *replace concept name*, which includes automatic addition of the old name to the synonym list, and *correct concept name*, which does not. With no standards for what to do with obsolete concepts, I had to make choices for *retire concept* regarding the fate of descendants of the retired concept. There were two options I considered: (1) retire all descendants of the retired concept, or (2) relink the parents of the retired concept with all

the children of the retired concept. I chose the latter. In addition, I had to make a decision about the fate of the attribute–value pairs of the retired concept. Again I considered two options: (1) push the attribute–value pairs of the retired concept down to all the children, or (2) effectively delete those attribute–value pairs so that they are no longer associated with the descendants, although previously they had been inherited by the descendants. I chose to delete them from the sets of inherited attribute–value pairs of descendants.

Because there are no standards for merge or split operations, and because such operations have received little attention in the literature, I arbitrarily chose two types of merges and one type of split that I believed sounded reasonable. If two concepts being merged are identical in meaning, then theoretically, either concept could be retained and the other retired. A vocabulary developer, however, might prefer not to select one concept over the other to retain, and might prefer to create a new unique identifier for the merged concept. Therefore, I offered both possibilities.

Initially, I had two split operations. The first split operation resulted in the concept being split into two new concepts. The second split operation made it possible to split a new concept off the original concept and retain the original concept. This process occurs in MeSH when there become too many Medline articles in one category, and it is prudent to create another classification of a subset of those articles. In this case, the original MeSH heading is retained and a new MeSH heading is created for the subset. However, I chose not to include this type of operation because it conflicts with the notion that the meaning of a concept should never change. Therefore, I included only one split operation.

Another arbitrary choice was the decision to include a single translation code in a concept's specification. An alternative would be to map to a single coding system such as the UMLS as I have done, but to permit more than one UMLS concept per CONCORDIA concept. Such an approach would be reasonable because there often is not a one–to–one mapping between concepts in different coding systems. Another option would be to permit specification of an unlimited number of coding systems to which translation could be performed. For example, there could be a data structure associated with each concept that permits a user to store a set of names of multiple coding systems, and the equivalent code for that concept in each coding system. However, I chose to assume that another service, such as the UMLS, would be available where translation to multiple health-care coding systems would be performed.

In the CONCORDIA log model, inclusion of both a timestamp and a sequence number is not necessary. A computer program could generate a sequence number according to timestamp. Due to this duplication of information, I would prefer to remove the sequence number from the model. However, maintaining a record of the sequence number made it unnecessary to include a timestamp-sorting routine in the implementation. Therefore, for ease of implementation, I included both timestamp and sequence number.

Sharing a structural model, change model, and log model, such as I have described for CONCORDIA, makes it possible for a shared-vocabulary site and a local-vocabulary site to share vocabulary content and communicate about changes. Synchronization of a local version of a shared vocabulary with the shared version from which it is derived requires a shared model like CONCORDIA. In the next chapter, I discuss synchronization methods and show that the chosen methods fulfill the goals of synchronization.

4 Synchronization Methods

Researchers who have investigated methods for sharing, modifying, converging, translating, merging, and differentiating vocabularies or ontologies—as discussed in Section 2.5—have dealt with a number of the same problems that arise in managing local divergence. For example, these investigators have faced problems with different representations of concepts and relationships that make integration difficult, problems that result from different perspectives and goals of content developers working with the same subdomain, and difficulties in determining automatically if two concepts have the same intended meaning. However, despite advances in these related areas, the problem of local divergence remains challenging.

In this chapter, I propose methods for coordinating divergent versions of a shared vocabulary through a process that is based on the CONCORDIA model. I define **synchronization** as the application of shared-vocabulary changes to the modified local version of a shared vocabulary to reach a target state. I describe the target state, synchronization change operations, the synchronization process, and the difference between automated and supported changes in a synchronization-support tool.

4.1 Synchronized State

The **target state**, or **synchronized state,** for local vocabulary *LV* and shared vocabulary *SV*, occurs when the following three requirements are met:

1. Every concept that is represented in *SV* is also represented in *LV*, and it has the same unique identifier (preservation of concept existence and identity).
2. Every subsumption relationship that exists in *SV* also exists in *LV* (preservation of subsumption relationships).
3. Every attribute–value pair in the defining attribute set of a concept in *SV* is also in the defining attribute set of that concept in *LV* (preservation of attribute–value pairs).

According to the first requirement for the synchronized state, if a concept unique identifier represents a concept in *SV*, then that concept unique identifier exists in *LV* and represents the same concept. Ideally, the concept in the shared vocabulary and the corresponding concept in the local vocabulary have the same meaning (in the opinion of

experts). If the two meanings are different, the situation would be considered an error, but if the requirements for a target state hold, then the system would still be in a synchronized state. The local vocabulary can include concepts that are not in the shared vocabulary, but all concepts in the shared vocabulary must be in the local vocabulary.

The second requirement for the synchronized state implies that, if concept *A* subsumes concept *B* in *SV*, then concept *A* subsumes concept *B* in *LV*. The relationship between *A* and *B* may be a parent−child relationship in both the *SV* and *LV*, but other states are also possible. For example, *A* could be a parent of *B* in *SV*, but *A* could be a parent of a parent (grandparent) of *B* in *LV*. Another way to state the second requirement is that, if *A* is a parent or ancestor of *B* in *SV*, then *A* is a parent or ancestor of *B* in *LV*. It follows that, if *A* is a child or descendant of *B* in *SV*, then *A* is a child or descendant of *B* in *LV*.

The third requirement guarantees that defining-attribute information will not be lost, but may be enhanced.

Formally, the set {*LR*, LC_1, LC_2, LC_3, ... LC_n, LA_1, LA_2, ... LA_m} is a **synchronized local vocabulary** *LV* associated with a shared vocabulary *SV* = {*SR*, SC_1, SC_2, SC_3, ... SC_p, SA_1, SA_2, ... SA_q} if:

1. *LV* is a local vocabulary.

2. *SV* is a shared vocabulary.

3. *LR* is the root concept of *LV*.

4. *SR* is the root concept of *SV*.

3. *LR*.concept_id = *SR*.concept_id.

4. *LR*.site_of_origin = “shared.”

5. If concept *SC* is a shared-vocabulary concept in the shared vocabulary *SV*, then there exists a local-vocabulary concept *LC* in the local vocabulary *LV* such that *SC*.concept_id = *LC*.concept_id.

6. If SC_1 and SC_2 are shared-vocabulary concepts that belongs to shared vocabulary *SV*, LC_1 and LC_2 are local-vocabulary concepts that belong to local vocabulary *LV*, SC_1.concept_id = LC_1.concept_id, SC_2.concept_id = LC_2.concept_id, and SC_1 is a parent or ancestor of SC_2 in *SV*, then LC_1 is a parent or ancestor of LC_2 in *LV*.

7. If *SC* is a shared-vocabulary concept that belongs to shared vocabulary *SV*, *LC* is a local-vocabulary concepts that belong to local vocabulary *LV*, and *SC*.concept_id = *LC*.concept_id, then for each attribute–value pair *avpair* belonging to the defining attribute set of *SC*, *avpair* belongs to the defining attribute set of *LC*.

Statements 5 through 7 are the three requirements for a synchronized state.

4.2 Synchronization Change Operations

A change operation is a **valid synchronization change operation** if it is either a valid local-vocabulary change operation (Section 3.2), or one of the following:

1. Merge local concept into shared concept

2. Merge local attribute into shared attribute

The operation *merge local concept into shared concept* is the same as the operation *merge concept into one of the two concepts* in the local-vocabulary change model, except constraints differ. The local-vocabulary operation permits the merge of two local-only concepts, and either concept may be retired. Alternatively, it allows a local concept to be merged into a shared concept, and the local concept is retired. In contrast, the synchronization operation dictates that the merge must be a merge of a local concept into a shared concept, with retention of the shared concept and retirement of the local concept. The reason for making a distinct operation that merges a local concept into a shared concept is to indicate explicitly that there was duplication of concepts in the two versions of the vocabulary. It emphasizes the importance of this task in the synchronization process. The purpose of *merge local attribute into shared attribute* is similar.

Changes that do not affect the hierarchy are relatively easy to implement; changes that do affect the hierarchy are more complex because they must not violate the parent–child relationship rules that affect defining attribute sets. Therefore, it is helpful to distinguish those changes that do not affect the hierarchy from those changes that do. Boxes 4.1 and 4.2 show the synchronization change operations, grouped accordingly.

Note that *add attribute* refers to adding the attribute as a registered attribute to the terminology, rather than adding it to a particular concept's attribute set. Also, *hide attribute* and *preserve attribute* refer to the attribute's existence in the vocabulary, rather

Box 4.1. Synchronization change operations that affect the concept hierarchy.

Add concept
Retire concept
Add parent
Remove parent
Add child
Remove child
Merge two concepts into one of the two concepts
Merge two concepts into new concept
Merge local concept into shared concept
Split concept into two new concepts
Hide concept
Preserve concept

Box 4.2. Synchronization change operations that do not affect the concept hierarchy.

Replace concept name
Replace concept definition
Add synonym
Delete synonym
Add abbreviation
Delete abbreviation
Add attribute–value pair
Remove attribute–value pair
Replace attribute value
Add attribute
Retire attribute
Merge two attributes into one of the two attributes
Merge two attributes into new attribute
Merge local attribute into shared attribute
Replace attribute name
Replace attribute definition
Hide attribute
Preserve attribute

than to its presence in the defining attribute set of a particular concept. Therefore, *add attribute*, *hide attribute*, and *preserve attribute* have no effect on the concept hierarchy.

The semantics of the synchronization change operations are the same as the semantics for the corresponding local-vocabulary operations, except for important differences in constraints related to site of origin. Certain change operations can be performed only on local-only concepts or attributes in the local editor, whereas those same change operations can be performed on only shared or locally modified shared concepts or attributes during synchronization.

In the local-vocabulary change model, the operation *add concept* adds a local-only concept to the vocabulary. It is not possible to add a shared concept during a local-vocabulary editing session. In contrast, during synchronization, the opposite is true: *add concept* adds a shared concept to the vocabulary, and it is not possible to add a local-only concept. Similarly, the operation *retire concept* can be performed on local-only concepts in the local-vocabulary change model, but is performed on shared concepts during synchronization. The analogous situation is true for *add attribute* and *retire attribute*.

Because the two concept merge operations and the split operation result in retirement of one or two concepts, the constraints on site of origin that apply to *retire concept* must be considered for merges and splits. On the one hand, in the local-vocabulary change model, merges and splits can take place only if concepts being retired are local only. On the other hand, during synchronization, merges and splits being processed affect shared concepts, because these operations reproduce operations that were performed on the shared vocabulary, which contains only shared concepts.

The remainder of the change operations have no constraints on site of origin in the local-vocabulary change model that differ from constraints on those same operations in synchronization. Those operations are *add synonym*, *delete synonym*, *add abbreviation*, *delete abbreviation*, *replace concept definition*, *replace UMLS code*, *add attribute–value pair*, *delete attribute–value pair*, *replace attribute value*, *hide concept*, *preserve concept*, *hide attribute*, and *preserve attribute*.

4.3 Synchronization Process

Before defining the synchronization process, I describe variables used in the definition. Let the following assertions hold:

1. *SV* is a shared vocabulary.
2. *LV* is a local vocabulary.
3. *LV* is synchronized with *SV* at time *t*.
4. *SVLog* is a shared-vocabulary log (i.e., a record of change operations applied to *SV* between time *t* and *t+1*).
5. *n* is the number of shared-vocabulary change operations applied to *SV* between time *t* and time *t+1*.
6. sv_op_i is the *i*th shared-vocabulary change operation applied to *SV* (*i*=1 to *n*).

7. $\{sv\_op_1, sv\_op_2, ... sv\_op_n\}$ is the ordered set of n shared-vocabulary change operations applied to *SV*.

8. CR_i is the ith change record in *SVLog*, which documents sv_op_i.

9. $SVLog = \{CR_1, CR_2, ... CR_n\}$.

10. *action* is an action performed on LV to process a change record *CR*.

11. *synch_op* is a synchronization change operation.

For each shared-vocabulary change operation (*sv_op*), there is exactly one change record (*CR*). For each change record, the user or system selects exactly one action (*action*). For each action, there are one or more steps. A valid action is a sequence of steps permitted for that action. For example, Tables 4.1 and 4.2 show the steps in valid actions for *add concept*, and *add parent*, respectively. Each step in an action is a synchronization change operation (*synch_op*). Therefore, n shared-vocabulary change operations correspond to m synchronization operations where $m \geq n$.

Table 4.1. Action choices for *add concept*.

Action	Steps
Action 1	Step 1: Add concept
Action 2 (if local name conflicts)	Step 1: Rename local concept with conflicting name Step 2: Add concept
Action 3 (if local name conflicts)	Step 1: Rename local concept with conflicting name Step 2: Add concept Step 3: Rename new concept Step 4: Rename local concept back to original name
Action 4 (if local concept has same meaning)	Step 1: Add concept Step 2: Merge local concept into shared concept Step 3: Retain local name as synonym
Action 5 (if local concept has same meaning)	Step 1: Add concept Step 2: Merge local concept into shared concept Step 3: Rename concept with local name Step 4: Retain shared name as synonym
Action 6 (if local concept has same meaning)	Step 1: Add concept Step 2: Merge local concept into shared concept Step 3: Rename concept with local name
Action 7	Step 1: Add concept Step 2: Hide concept

Table 4.2. Action choices for *add parent*.

Action	Steps
Action 1	1: Add parent 2: Add parent to list of SV parents
Action 2 (if a cycle would result from adding the parent)	1: Break cycle 2: Add parent 3: Add parent to list of SV parents
Action 3 (if concept gaining new parent is already subsumed by parent)	1: Add parent to list of SV parents

The **synchronization process** is the step-by-step application of a sequence of valid actions {$action_1$, $action_2$, ... $action_n$} to the local vocabulary *LV* that corresponds to the sequence of change records {CR_1, CR_2, ... CR_n} in *SVLog*. When there is one or more valid action for a given type of change record, the selected action is based on either the state of the local vocabulary, or on user preference. The result is a sequence of synchronization operations applied to the local vcoabulary determined by the sequence of actions. The goal at the end of synchronization is for the local vocabulary to be in a synchronized state.

The actions must be performed on *LV* in the same order specified by the order of the change records. That is, for $i = 1$ to n and $j = 1$ to n, such that $i < j$, if $action_i$ corresponds to CR_i, and $action_j$ corresponds to CR_j, then $action_i$ is applied to *LV* before $action_j$ is applied to *LV*.

4.3.1 Valid Actions

Appendix H gives valid actions for all the shared-vocabulary change operations. Allowable actions must be implemented so that the requirements for a synchronized state and for a local vocabulary are fulfilled. Important local-vocabulary constraints are

1. Concept names are unique.
2. There are no cycles.
3. If a concept *C* has attribute–value pair made up of attribute *A* and value V_1, a descendant of *C* may have an attribute–value pair made up of attribute *A* and value V_2 if V_1 subsumes V_2.

Change operations that involve additions of concepts or attributes are *add concept*, *add attribute*, and the merge and split operations. Action choices for these operations must guarantee that there is no local concept or attribute with the same name as the new concept or attribute, and must guarantee that the new concept or attribute is added to the local vocabulary.

Change operations that affect retirement of a concept are *retire concept*, *retire attribute*, and the merge and split operations. Action choices allow the concept or attribute that was retired in the shared vocabulary to be either retired or preserved in the local vocabulary.

Change operations that result in addition of a parent or a child (*add parent* and *add child*) must take into account the possibility that a cycle will occur after the parent or child is added. Because cycles are not permitted, the system must take steps to avoid creating a cycle before adding a parent or child.

Change operations that result in removal of parents and children are *remove parent* and *remove child*. If a parent or child was removed in the shared vocabulary, the local-vocabulary maintainer may choose whether to remove it from the local vocabulary, or to leave it as is.

Change operations that affect attribute–value pairs are *add attribute–value pair*, *delete attribute–value pair*, and *replace attribute value*. Deleting an attribute–value pair cannot cause conflicts, but addition of an attribute–value pair or changing the value of an attribute may do so. Action choices must guarantee that the new or modified attribute–value pair is added to the local vocabulary, but the requirement on attribute values for concepts that share the same attribute as an ancestor must not be violated. The system must take steps to make sure that the maintainer removes attribute–value pairs that were added locally if they would conflict after the change from the shared vocabulary is made.

Change operations that do not affect the hierarchy, but that affect name changes are *replace concept name* and *correct concept name*. Action choices must take into consideration the requirement for unique names. If there is a conflict, then one of the conflicting names must be changed. If there is no conflict, then the user has the choice of whether to change the name. In the case of *replace concept name*, the user also has the choice of whether to add the old name to the synonym list.

Change operations that do not affect the hierarchy, and that do not affect uniqueness of the concept name are *replace concept definition, replace UMLS code, add synonym, delete synonym, add abbreviation,* and *delete abbreviation*. These operations do

not cause potential conflicts in the local vocabulary, and are not required during synchronization. Therefore, action choices permit the local-vocabulary maintainer to decide whether to make the corresponding change in the local vocabulary.

4.3.2 Claims

By definition of the requirements of a local vocabulary (Section 3.1.2) and by definition of synchronization change operations (Section 4.2), the following claim holds.

> Claim 1. If LV_i fulfills the requirements of a local vocabulary, and *synch_op* is a valid synchronization change operation, which causes the local vocabulary to be transformed from an initial state, LV_i, to a subsequent state, LV_{i+1}, then LV_{i+1} also fulfills the requirements of a local vocabulary.

In other words, after a single synchronization change operation is applied to *LV*, *LV* is still a local vocabulary, according to the definition of *local vocabulary*. Figure 4.1 depicts the process.

By transitivity of Claim 1, the following claim is also true.

Claim 2. If LV_i fulfills the requirements of a local vocabulary, $synch\_op_1$, $synch\_op_2$, ... $synch\_op_n$ are valid synchronization change operations, and LV_{i+1}, LV_{i+2}, ...LV_{i+n} are the states of the local vocabulary after each change operation respectively, then LV_{i+n} also fulfills the requirements of a local vocabulary. In other words, after a series of synchronization change operations is applied to *LV*, *LV* is still a local vocabulary, according to the definition of *local vocabulary*. Figure 4.2 depicts the process.

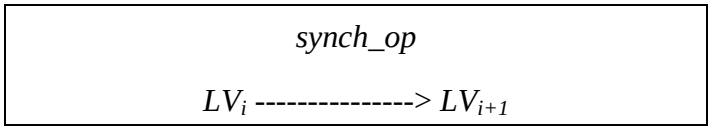

Figure 4.1. Application of one synchronization operation to a local vocabulary.

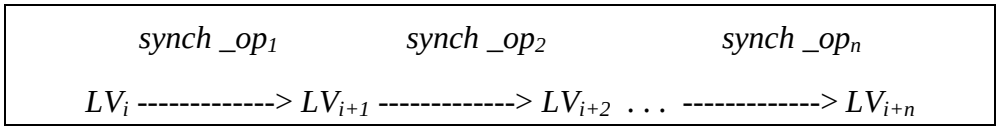

Figure 4.2. Application of a series of synchronization change operations to a local vocabulary.

These two claims partially support the final claim.

Claim 3. Suppose that

1. LV_t is a synchronized local vocabulary that is synchronized with shared vocabulary SV_t at time *t*.
2. SV_t is transformed to SV_{t+1} by a series of shared-vocabulary change operations.
3. LV_t is transformed to LV_i by a series of local-vocabulary change operations. LV_i is not synchronized with SV_{t+1}.
4. $\{action_1, action_2, ... action_n\}$ is the sequence of actions that the user or system selects and that corresponds to the sequence of change records $\{CR_1, CR_2, ... CR_n\}$.
5. $\{synch\_op_1, synch\_op_2, ... synch\_op_m\}$ is the sequence of synchronization change operations that are applied to LV_i as a result of $\{action_1, action_2, ... action_n\}$.

Then, LV_{i+m} is a synchronized local vocabulary that is synchronized with SV_{t+1}.

In other words, if *LV* is synchronized with *SV* at time *t*, then after a series of local-vocabulary change operations and synchronization change operations, *LV* will be a synchronized local vocabulary that is synchronized with *SV* at time *t+1*. Figure 4.3 depicts the process.

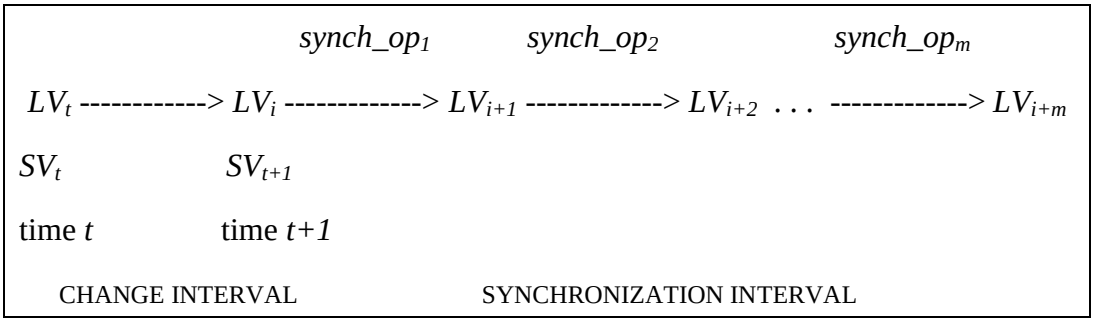

Figure 4.3. Application of changes to a local vocabulary during the change interval followed by application of synchronization operations during the synchronization interval.

Discussion of Claim 3.

By Claim 2, we know that LV_{i+m} will be a local vocabulary. To show that LV_{i+m} is in a synchronized state, we consider the three requirements for synchronized state: (1) preservation of concept existence and identity, (2) preservation of subsumption relationships, and (3) preservation of attribute–value pairs.

For the first requirement of a synchronized state to hold, the following must be true:

A. Every concept that was added to *SV* is also added to *LV* and has the same unique identifier in *LV* that it has in *SV*.

B. No concept in *LV* is retired that was not retired in *SV*.

There are two phases during which LV can change. First, there is the local-vcoabulary editing phase (called "CHANGE INTERVAL" in Figure 4.3). Second, there is the synchronization phase (called "SYNCHRONIZATION INTERVAL" in Figure 4.3).

Statement A is true because during the local-vocabulary–editing phase, no concepts of type *shared* or *locally modified shared* are added. Only *local-only* concepts can be added to *LV*. During the synchronization phase, the only concepts that are added to *LV* are those that were added to *SV* and recorded in the shared log. Inspection of the permissible actions for *add concept* shows that each valid action includes a step that adds the concept. There are seven valid actions permissible for *add concept* (Table 4.2.). When *add concept* is performed, the concept added to *LV* is a replica of the concept added to *SV*, but also includes information about site of origin, SV parents, and SV children. The concept unique identifier is the same when the replicated concept is created, and there is no valid change operation in the local-vocabulary change model that permits a user to change a concept's unique identifier after concept creation. Thus, every concept added to *SV* exists in *LV* and has the same unique identifier.

Statement B is true because during the local-vocabulary–editing phase, no shared concepts may be retired due to constraints on *retire concept*, *merge two concepts into one of the two concepts*, *merge two concepts into new concept*, and *split concept into two new concepts*. During the synchronization phase, shared concepts may be retired only if they were retired previously in *SV*.

For the second requirement of a synchronized state to hold, subsumption relationships must not be removed during local-vocabulary editing or during synchronization if the corresponding relationships have not been removed in the shared

vocabulary. During local-vocabulary editing, the operation *remove parent* and *remove child* could lead to incompatibility with the shared vocabulary if concepts affected are shared or locally modified shared. These operations are permitted only if another subsumption path links the parent and child concepts that are directly involved in the operation. If *remove parent* or *remove child* breaks a subsumption path that affects any other concepts in *LV* that have a known subsumption relationship in *SV*, then the operation is not permitted. During synchronization, if *remove parent* or *remove child* is processed from the shared log, the same operation may be applied to the corresponding shared concepts in *LV*, but does not have to be. Whether or not the change is applied to *LV*, the list of *SV* parents or *SV* children in the corresponding concept in *LV* is updated to maintain information about subsumption relationships in *SV*.

For the third requirement of synchronized state to hold, attribute–value pairs must not be lost during local-vocabulary editing or during synchronization. During local-vcoabulary editing, attribute–value pairs in shared or locally modified-shared concepts cannot be deleted if they are not already inherited. During synchronization, *add attribute–value pair* or *replace attribute value* must be applied to the corresponding shared or locally modified-shared concept in *LV* if the resulting attribute–value pair is not already inherited.

The synchronization process is implemented in the synchronization-support tool, which is discussed in Chapter 5.

5 Implementation

CONCORDIA provides a data model for the shared and local vocabularies, a specification for change operations, and a data model for change logs. Concept Manager is an implementation that demonstrates use of the CONCORDIA model. The purpose of Concept Manager is to support synchronization.

Concept Manager is the name of the overall system; it comprises a suite of the five applications. The five applications are (1) a shared-vocabulary browser, (2) a shared-vocabulary editor, (3) a local-vocabulary browser, (4) a local-vocabulary editor, and (5) a synchronization-support tool. Together these applications form a coordinated concept-management environment for the shared-vocabulary–development organization and for the local organizations that conform to but adapt the shared vocabulary to local needs. In this chapter, I describe the design and functionality of the implemented system. Figure 5.1 shows the components of the Concept Manager system and the flow of data.

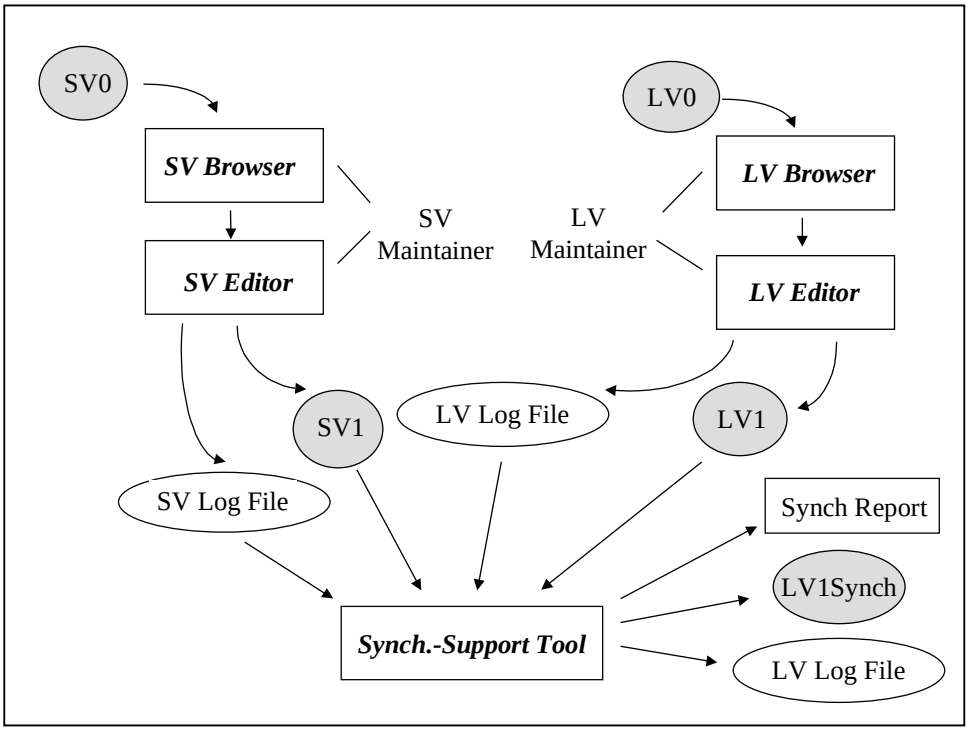

Figure 5.1 Components of Concept Manager and flow of data.

Legend:
SV: Shared Vocabulary; SV0: SV at time 0; SV1: SV at time 1;
LV: Local Vocabulary; LV0: LV at time 0; LV1: LV at time 1;
LVSynch1: LV at time 1 after synchronization

5.1 Functional Requirements

The browser and editor are distinct applications because an end user would have rights to view and navigate the vocabulary, but would not have the authority to make changes. A vocabulary developer, however, would need both browsing and editing functions. An end user would use a stable, static version of the vocabulary, and the vocabulary developer would use a dynamically evolving version of the vocabulary. In most cases, end users will use a browser, and developers will use an editor that includes a browser. It is important to separate the functions required by the two groups.

The browser must enable the user to perform the following tasks: (1) search for a concept or attribute; (2) view information about a selected concept or attribute; (3) view the hierarchy of concepts; (4) navigate through the hierarchy by clicking recursively from parent to child or from child to parent.

In addition to providing support for browsing tasks, the editing application must be able to perform the following services: (1) add, modify, and retire concepts and attributes at the user's request; (2) perform checks on data input when the user requests a change, and guarantee that constraints of the structural model or change model will not be violated; (3) display changes made during the current editing session; (4) save the modified version of the vocabulary; and (5) save changes in a log.

Browsing and editing requirements for the local and shared vocabularies are the same, but the local-vocabulary software must take into account differences in the vocabulary structural model and change model. For example, the local-vocabulary browser must display the site of origin for a concept or attribute (with values of *shared*, *local only, or locally modified shared*), and the local-vocabulary editor must offer the change operations *hide* and *preserve* for both concepts and attributes.

The synchronization-support tool must make it possible for the user to view and understand changes that were made to the shared vocabulary, and must enable the user to make changes to the local vocabulary that correspond to changes made to the shared vocabulary. Tasks that the tool must perform are as follows: (1) display changes made to the shared vocabulary; (2) guide the user through the change process, facilitating decision making by the user, and performing as many tasks automatically as possible; (3) permit the user to browse the shared vocabulary, which is static; (4) permit the user to browse the local vocabulary, which is changing during synchronization; (5) display changes

made to the local vocabulary during the synchronization session; (6) save the modified version of the local vocabulary; and (7) save changes in a log.

5.2 Design Choices

A variety of choices were available regarding programming design, user interface, and input and output mediums. I chose an object-oriented design, made compromises on the user interface, and used text files for basic input and output.

5.2.1 Object-Oriented Design

I chose to use an object-oriented design to facilitate reuse of program code in the five applications (shared-vocabulary browser, shared-vocabulary editor, local-vocabulary browser, local-vocabulary editor, and synchronization-support tool). These applications have numerous overlapping functions and services, and object classes can be reused in their original form or modified by subclassing.

Many of the classes that are subclassed are needed by both the shared and local vocabularies. All the data elements of such a class in the shared vocabulary are inherited by the corresponding class in the local vocabulary, which may have additional data elements. All the methods of the class in the shared vocabulary also are inherited or overridden by the class in the local vocabulary, which may have other methods as well. Classes used to support data entry and data display in the shared-vocabulary browser and editor are subclassed for use in the local-vocabulary browser and editor.

This approach allows distinct separation of editing functionality from browsing functionality. That is, the browser can be packaged as a separate application that makes no reference to the editor. Similarly, the local-vocabulary software is separate from the shared-vocabulary software. The local vocabulary depends on the shared vocabulary but not vice versa. None of the classes required for the shared vocabulary make reference to any classes, data elements or methods that are specific to the local vocabulary. The shared-vocabulary software can be packaged separately and makes no reference to local-vcoabulary features.

5.2.2 User Interface

User-interface design has a significant influence on the usability of a software tool. For a vocabulary of thousands or hundreds of thousands of concepts, the challenge is to provide the user with essential information, but to minimize data overload. It is also

important to minimize the number of mouse clicks and keystrokes needed for navigation and data entry. There is no consensus on the optimal design of human−computer interfaces for browsers and editors for large vocabularies. My goal, however, is to develop models that represent change and methods that support change, rather than to study user interfaces and human-computer interaction. Therefore, I created a user interface that offers the desired services, but tradeoffs and compromises were necessary due to time and resource limitations. Development of elaborate user interfaces and evaluation of different approaches are beyond the scope of this work.

5.2.3 Input and Output

Figure 5.2 summarizes input and output.

Each of the applications reads in an entire vocabulary from a text file in XML format. (Section 3.4 describes the use of XML for the vocabulary data file, and Appendix F gives the DTD for the shared vocabulary.) A browser or editor reads in only one vocabulary at a time. The synchronization-support tool reads in one shared vocabulary and one local vocabulary. The synchronization-support tool also reads in a shared-vocabulary log and a local-vocabulary log. (Appendix G gives the DTD for the log.) Vocabularies and logs are stored in main memory.

The browsers, editors, and synchronization-support tool accept keyboard and mouse-click data entry for user input. For example, when using the browser, the user types in a string, and the program searches for a concept whose name matches the input. Alternatively, the user selects a concept by clicking on a menu item in a pull-down menu that displays concepts hierarchically.

The browser does not produce any output. The editor produces a modified version of the vocabulary in a vocabulary output file, and a record of changes in a log output file. The local-vocabulary editor and synchronization-support tool produce a modified version of the local vocabulary in a local-vocabulary output file, and a record of changes in a log output file. The synchronization-support tool also produces a report of changes that occurred during the synchronization session.

The log output files generated by the local-vocabulary editor and by the synchronization-support tool are not formatted specifically for the local vocabulary; in particular, the log file does not contain information about concept or attribute site of origin. A better design choice would be to generate a local-vocabulary–specific log output file; however, because the additional information was not needed for the

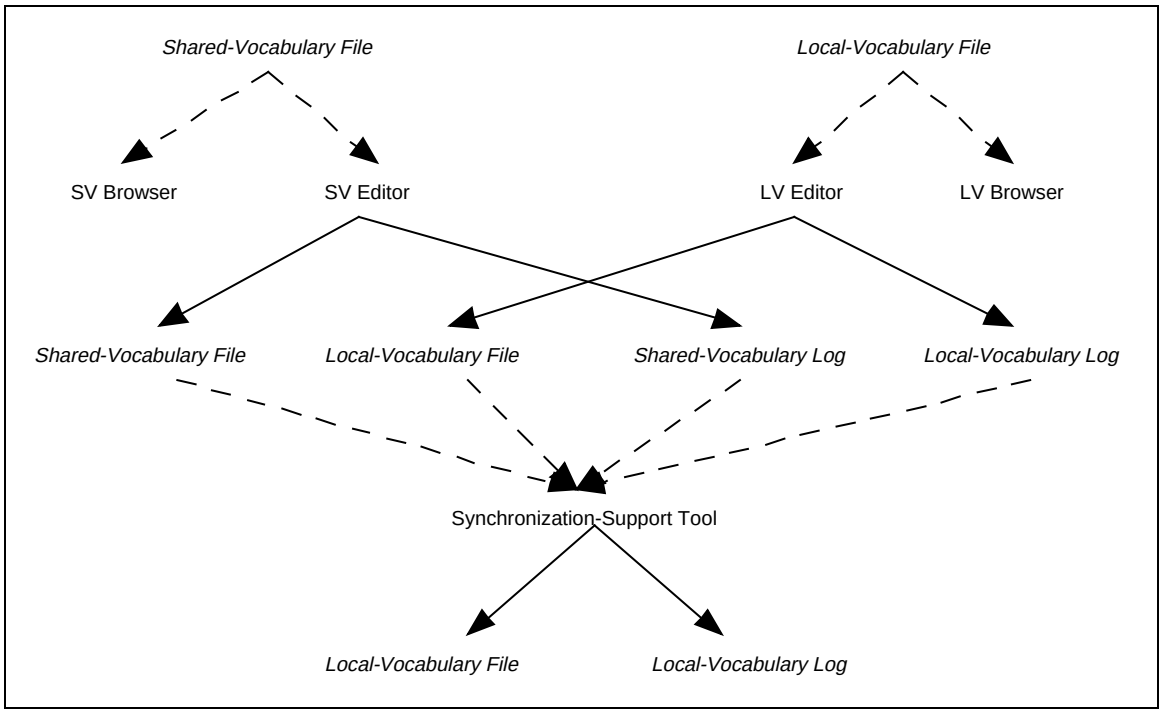

Figure 5.2. Inputs and outputs of applications. Names of input and output files are italicized; names of applications are not. Files at the origins of dashed arrows represent inputs; files at the ends of solid arrows represent outputs.

synchronization task in this research, I chose to reuse the same program code that writes and reads logs for the shared vocabulary.

5.3 Development Environment

I used the Java[1] programming language [Flanagan 1997], Version 1.1. Unfortunately, the Java Foundation Classes—commonly called the Java Swing Set—were not available when I began this work [Topley 1998]. The Java Swing Set includes a class for displaying trees that would have been useful for displaying concept hierarchies. However, since Java 1.1 does not include a tree class, I used pull-down menus to display concept hierarchies instead.

For the software-development environment, I used Metrowerks CodeWarrior[2] Integrated Development Environment (IDE), Release 3 [Metrowerks 1998]. The CodeWarrior IDE supports editing, compiling, and debugging in Java.

[1] Java™ is a trademark of Sun, Inc.

[2] Metrowerks™ and CodeWarrior™ are trademarks of Metrowerks, Inc.

For hardware, I used a Windows NT workstation that runs Version 4.0 of the Windows NT operating system. The computer has a 400 MHz Pentium II processor with 128 MB of random-access memory. The video display has a resolution of 1024 x 768 pixels.

The software that I developed for this project functions as a standalone application that runs on a single computer. Concept Manager currently is not accessible over a network.

5.4 Basic Functionality

Many classes developed for the system are used repeatedly by other classes in the system. I describe several of those classes here to provide an explanation of the basic functionality of the implementation, and to relate function to design. In a shared object model of vocabulary services, interfaces to classes such as these could be standardized and shared. Each standard interface would contain a specification for the data elements of a class and for the methods that can be performed on that class. In this section, I concentrate on data elements and methods in classes that provide basic services in the software. I discuss core vocabulary functions, search methods, change operations, change logs, error checking, data entry and data display.

5.4.1 Core Vocabulary Functions

The basic classes required for populating the vocabulary with concepts and attributes according to the CONCORDIA model are (1) Vocabulary, (2) Concept, (3) Attribute, and (4) Attribute–Value Pair.

The **Vocabulary** class contains a root concept and a set of hashtables that index concepts and attributes in the vocabulary. An instance of Vocabulary is the vocabulary itself. Multiple hashtables are currently implemented, including a table that indexes concepts by concept code, and a table that indexes concepts by concept name. Because a code is unique, a table lookup for a concept, given a concept code, gives a unique concept. Similarly, because a concept name is unique for current concepts, a table lookup of a concept in the current concept-name table, given a concept name, gives a unique concept. However, because the name of a retired concept can be reused for a current concept, the table of all concepts indexed by concept name may yield more than one concept, given a concept name.

Additional tables in the Vocabulary class permit lookup of attributes, given attribute code or attribute name. There is also a synonym table and an abbreviation table for concepts. From the synonym table, one or more concepts can be retrieved given a string that matches a synonym. The abbreviation table functions similarly.

In addition to providing methods for table lookups, the Vocabulary class contains methods for the vocabulary change operations *add concept, add attribute, retire concept, retire attribute,* the concept merge operations, the concept split operation, and the attribute merge operations.

The **Concept** class contains data elements that are used to represent a concept: (1) concept unique identifier, (2) concept name, (3) synonyms, (4) abbreviations, (5) concept definition, (6) UMLS code, (7) parents, (8) children, and (9) defining attribute set. In addition, it contains two data elements that facilitate management of links when parents and children are retired: (1) retired parents and (2) retired children. Methods support queries that get information, and perform updates that set information about a concept. Update methods correspond to concept change operations. Examples of concept change operations supported by methods in the Concept class are *replace concept name, add synonym, add parent,* and *replace attribute value*.

The **Attribute** class contains data elements that are used to represent an attribute: (1) attribute unique identifier, (2) attribute name, and (3) attribute definition. Methods support queries that get information about an attribute, and perform updates that set information about an attribute. Update methods correspond to attribute change operations. The only attribute change operations are *replace attribute name, correct attribute name,* and *replace attribute definition*. The Attribute class also offers a method (*get linked concepts*) that determines all concept pairs that are linked by a particular attribute. For example, if concept C_1 has attribute A with value C_2 in its attribute set, then C_1 and C_2 form a pair of concepts that are linked by attribute A. If no concept pairs are linked by the attribute, the attribute can be retired.

As its name implies, the **Attribute–Value Pair** class contains an attribute and a value. A list of attribute–value pairs associated with a concept forms the attribute set in the Concept class. The Attribute–Value Pair class is used repeatedly, but the way it is used is an implementation choice. The CONCORDIA model does not require that a unique identifier be assigned to each attribute–value pair. Therefore, in this implementation, two attribute–value pairs are equivalent if they contain the same attribute, for which there is an attribute unique identifier, and the same value, for which there is a concept unique

identifier. However, those two attribute–value pairs may have different references, or addresses. Methods for this class include queries about the attribute and value, as well as methods for setting the attribute and value.

5.4.2 Search Methods

The **Search** class provides methods that search for concepts or attributes, given particular inputs. The browser performs the following functions:

1. Retrieves concept or attribute that has the same name as the input string
2. Retrieves concepts that have a synonym that matches the input string exactly
3. Retrieves concepts that have an abbreviation that matches the input string exactly
4. Retrieves concepts that have a name and/or synonym that matches the input string exactly
5. Retrieves concept or attribute that has a unique identifier (i.e., code) that matches the input string
6. Retrieves concepts or attributes whose names share the same first three letters as the three-letter input string

There are many more search methods that could enhance a user's search for concepts and attributes—for example, term completion, morpheme matching, lexical-variant matching, and matching tokens within phrases. Browser developers could incorporate quantitative methods to rank order possible matches. Researchers at the NLM have studied methods for matching input strings to concepts in the UMLS [Divita 1998]; my implementation would benefit from the application of their findings. However, because analysis of search methods is not the emphasis of this research, I have implemented only a small set of basic search methods in this prototype. A broader set of search services would be desirable and necessary in a production system.

5.4.3 Change Operations

For each change operation in the CONCORDIA model, there is one change-operation class. Each change-operation class has methods for performing a temporary change, for performing error checks on the proposed change, for performing the actual change, and for adding a record of the change to the log. For example, the class for the

operation *merge two concepts into new concept* contains a method that performs the proposed merge on copies of the two concepts. The purpose of performing a temporary merge is to permit the user to review the effects of a change before committing to that change. The merge class performs error checks and returns flags if errors occur. The method for completion of the actual change is invoked if the change is confirmed by the user, who makes a decision to confirm after reviewing results of the temporary change. Finally, the class contains a method that creates a change record to document the merge, and adds that change record to the log.

5.4.4 Log

The log is a sequence of log classes. Each log class is specific for a particular change operation, but inherits properties from more general classes. Figure 5.3 depicts the class hierarchy for log classes. The most general class (LogChange**)** is the abstract superclass of each of three other abstract classes that support vocabulary changes, concept changes, and attribute changes (LogVocabularyChange, LogConceptChange, and LogAttributeChange). These three abstract classes are direct superclasses of the non-abstract change-specific log classes (e.g., LogReplaceConceptName, LogAddParent, LogDeleteSynonym). Each change-specific class inherits data elements and methods

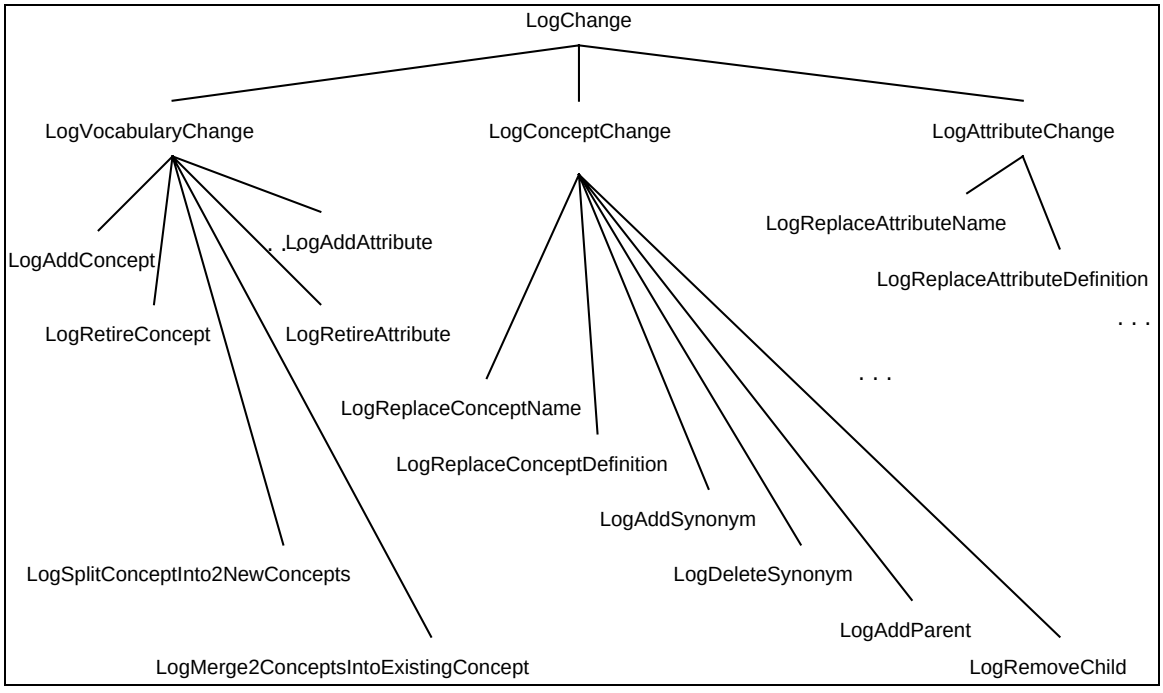

Figure 5.3. Hierarchy of classes for changes in the log.

from its superclasses. For example, the log class for the operation *add parent* (LogAddParent) inherits universal change-data elements *date*, *timestamp*, *author*, *explanation*, and *change-operation name* from the top-level class (LogChange); it inherits concept-change data elements *concept unique identifier* and *current concept name* from its direct superclass (LogConceptChange); and it provides its own change-specific data elements *parent concept unique identifier* and *parent concept name* in its own set of data elements (LogAddParent).

A diagrammatic representation of a portion of a sample log is shown in Figure 5.4. A log is a sequence of change records of different types. A legal change-record type is one of the non-abstract change-specific log classes.

For certain change operations, completion of the change operation generates not only a change record for the log, but also a sublog that goes in the log. Any change operation that involves more than one step requires a sublog. Such operations include *replace concept name, merge two concepts into one of the two concepts, merge two concepts into new concept,* and *split concept into two new concepts*.

For example, *replace concept name* is a composition of *correct concept name* followed by *add synonym*. The concept name is first changed to the new concept name, and then the old concept name is added as a synonym. The resulting additions to the log would be the change record for *replace concept name,* and a sublog that contains two change records—one record for *correct concept name* followed by a second record for *add synonym*.

The merge and split operations have sublogs that document the creation of new concepts; that document addition of parents, children, synonyms, abbreviations, and attribute–value pairs to original or new concepts; and that document retirement of the original concepts. Thus, sublogs for merges and splits may describe many change

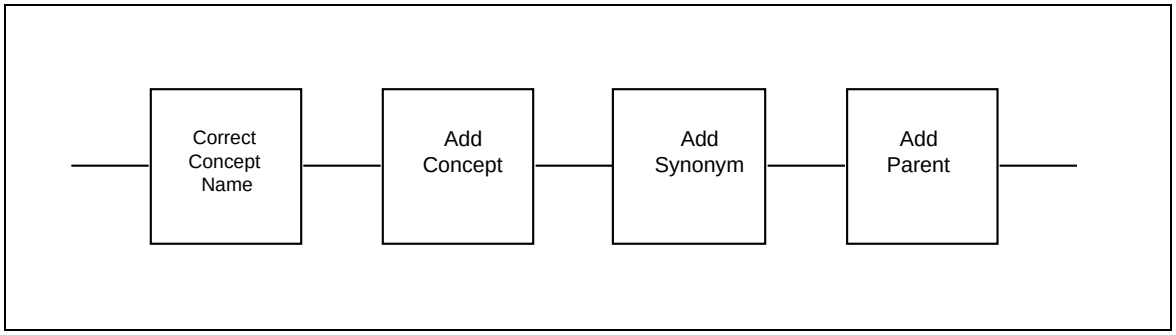

Figure 5.4. Portion of a sample log. The log is a sequence of change records in chronological order.

operations. The sublog will contain many records if the number of parents, children, synonyms, abbreviations, and attribute–value pairs is large.

5.4.5 Error Checking

Constraints in the CONCORDIA model are enforced in Concept Manager through error-checking classes. The error-checking classes are used by the editors and the synchronization-support tool.

In the shared-vocabulary editor, every request for a change to the vocabulary is checked against constraints for that particular change operation. For example, if the user wants to add a concept, the system checks to see whether any current concept already has the same concept name as the concept being added. If one does, an error message pops up with an explanation of the problem. In addition, the system checks to see be sure that the parent specified for the new concept exists and is not retired. If the parent does not exist or is retired, an error message pops up. If a retired concept already has the same name as the name given to the new concept, the system recognizes a potential problem, but issues a warning, rather than an error message. The CONCORDIA model permits reuse of names from retired concepts, but the user may be unaware that the old name exists, and a warning may be helpful. The difference between an error and a warning is that the user can still choose to make the change despite a warning, but if an error is identified, the system will block the change until the error is corrected.

Error checking is the same in the local-vocabulary editor as in the shared-vocabulary editor, but the software checks for additional constraints on site of origin and usage status. For example, *retire concept* has the additional constraint in the local vocabulary that a concept can be retired only if it is a local-only concept. In contrast, in the synchronization-support tool, processing a record for *retire concept* in the shared vocabulary log implies that *retire concept* will be performed on a shared concept in the local vocabulary. Therefore, constraints on changes to the local vocabulary may differ depending on whether the changes are being made from within the local-vocabulary editor or from within the synchronization-support tool. The local-vocabulary editor also performs checks on operations *hide concept*, *hide attribute*, *preserve concept*, and *preserve attribute*, which are operations that do not exist in the shared vocabulary.

5.4.6 Data Entry and Data Display

Many of the classes in Concept Manager support the graphical user interface. There are myriad ways that such classes can be designed; I do not describe the user-

interface classes in detail. I note, however, that many of the classes designed for the browser were readily adaptable by the editor, and that many of the classes designed for the shared vocabulary were readily adaptable by the local vocabulary. In addition, classes that support display of hierarchies through pull-down menus, display of changes made during the editing session, and display of vocabulary summary data are reused by the synchronization-support tool. I used subclassing techniques to reuse these functions.

5.5 Applications

In this section, I summarize each of the applications, as viewed from the user's perspective.

5.5.1 Shared-Vocabulary Browser

The shared-vocabulary browser is shown in Figure 5.5. In the center of the upper panel is a pull-down menu from which the user selects the type of search. Search options

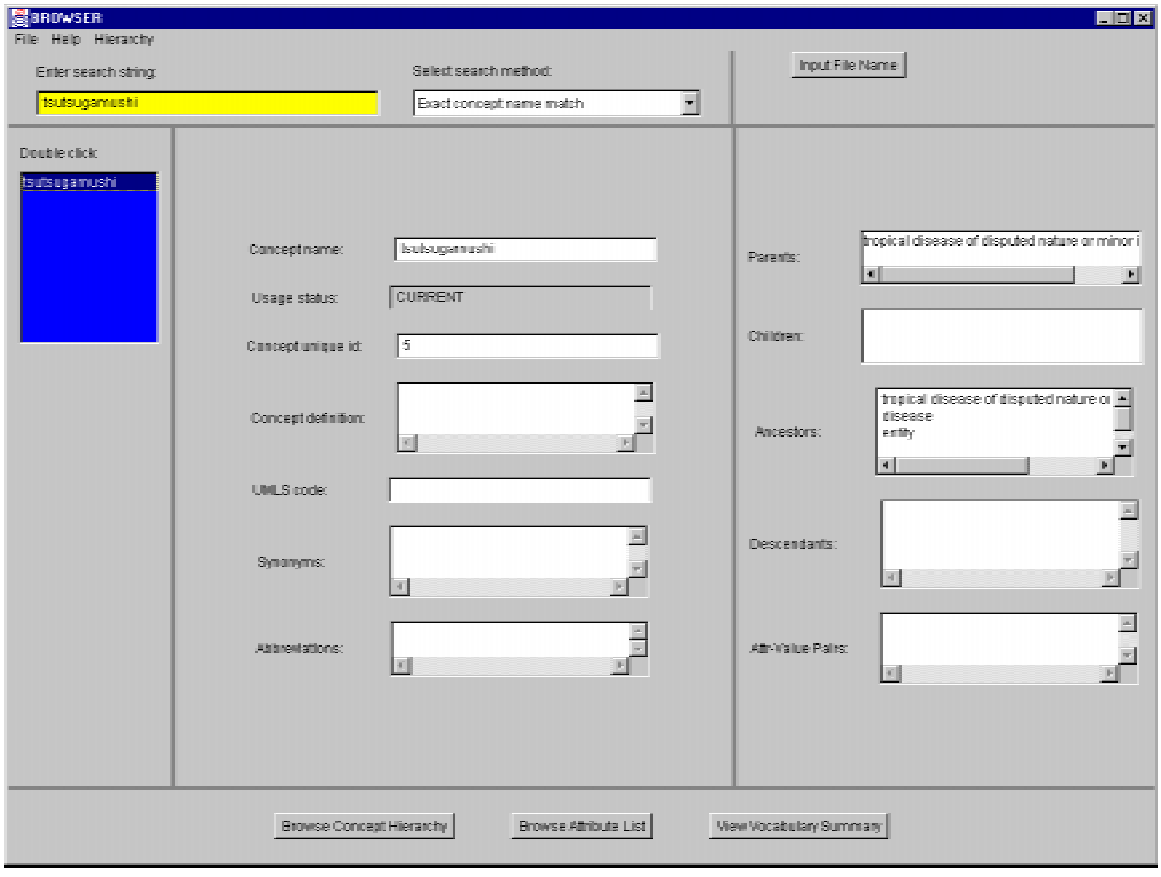

Figure 5.5. Shared-vocabulary browser.

are the functions described previously in Section 5.4.2. To the left of the pull-down menu is a field in which the user enters a search string. The user hits *Return* after entering a search string.

The concept or concepts that match appear in the list box in the left panel. The user double clicks on a selected concept by name, and the information about the corresponding concept appears in the data display. The concept that appears in the data display is the **concept of focus**.

To march up or down the hierarchy from the concept of focus, the user can double click on one of the parents or one of the children listed in the *Parents* or *Children* boxes, respectively. When the user selects a parent or child, the concept of focus switches to that parent or child, and the information about the new concept of focus appears in the display.

Alternatively, the user can navigate up and down the hierarchy by starting at the root on the pull-down menu at the top of the screen labeled *Hierarchy*. Each concept that has children has a submenu that displays those children. The user can navigate the hierarchy by traversing menus and submenus, and can select a concept at any level.

5.5.2 Shared-Vocabulary Editor

When the user launches the shared-vocabulary editor, the browser interface appears first, and the user has access to the full set of browsing services.

To edit the vocabulary, the user clicks on the button *Go to Editor*, and the editor interface appears (Figure 5.6). The concept of focus that the user selected in the browser automatically becomes the concept of focus in the editor. The user may also select a concept from within the editor by typing a string in the text field in the upper panel. For simplicity in software development, the only two search techniques available in this portion of the interface are exact string matches of concepts or attributes. Alternatively, the user can select a concept of focus by navigating the pull-down menu labeled *Hierarchy* and by clicking on a concept.

The left side panel in the editor displays information about the concept. The space available for concept information in this area is limited, however, and therefore a toggle button labeled *Next* (or *Previous* on the next screen) is necessary. The user toggles back and forth between *Next* and *Previous* to see the complete information about the concept.

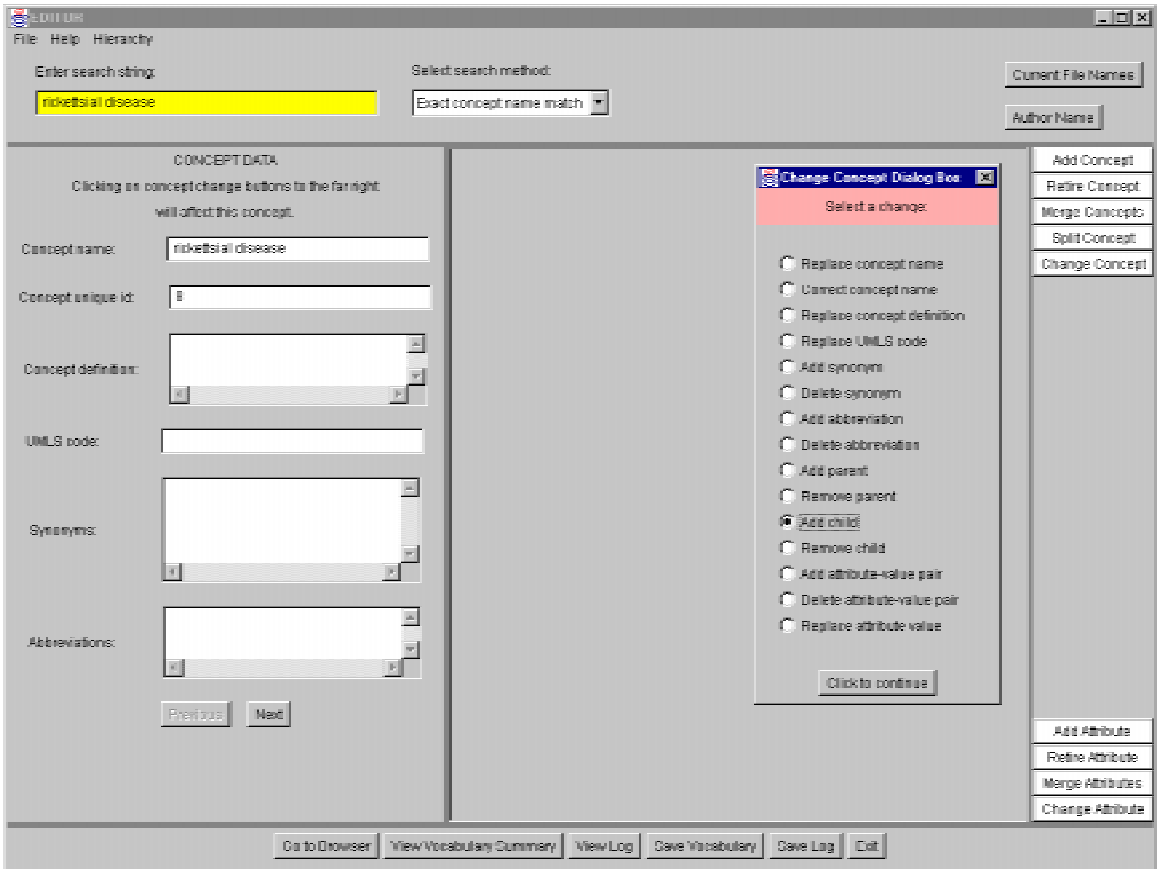

Figure 5.6. Shared-vocabulary editor.

Buttons on the right side panel refer to categories of changes. If the user clicks on *Add Concept*, a panel in the center appears for the user to enter the name of the concept, and the name of the parent concept. The system automatically retrieves and displays the unique identifier for the parent concept. Unique identifiers in the shared-vocabulary editor are integer strings. The system assigns integer unique identifiers in numeric order as it creates new concepts. The built-in root concept has the unique identifier *0*.

If the user clicks on the button *Change Concept*, a dialog box appears with a list of changes from which the user chooses, as shown in Figure 5.6. These changes correspond to concept change operations in the CONCORDIA model. Similarly, if the user clicks on *Change Attribute*, a dialog box appears with a list of changes that correspond to attribute change operations in the CONCORDIA model.

After the user makes a change request and enters the data required, the system performs a temporary change. A temporary change does not alter the vocabulary itself, but demonstrates what the effect of the change will be. The user has a chance to confirm or cancel the change at this point. If the user clicks to confirm, then a display of actual

results appears. The system creates a change record by making an instance of the appropriate log change class, and stores the record in sequence in the log.

The user may view the contents of the evolving log as it exists up to that point at any time during the editing session by clicking on *View Log*. At the end of the editing session, the user clicks on *Save Vocabulary* and *Save Log* to save the modified vocabulary and the log, respectively.

5.5.3 Local-Vocabulary Browser

The local-vocabulary browser looks like the shared-vocabulary browser, with one notable difference. There is an extra data field that displays the concept or attribute site of origin (Figure 5.7).

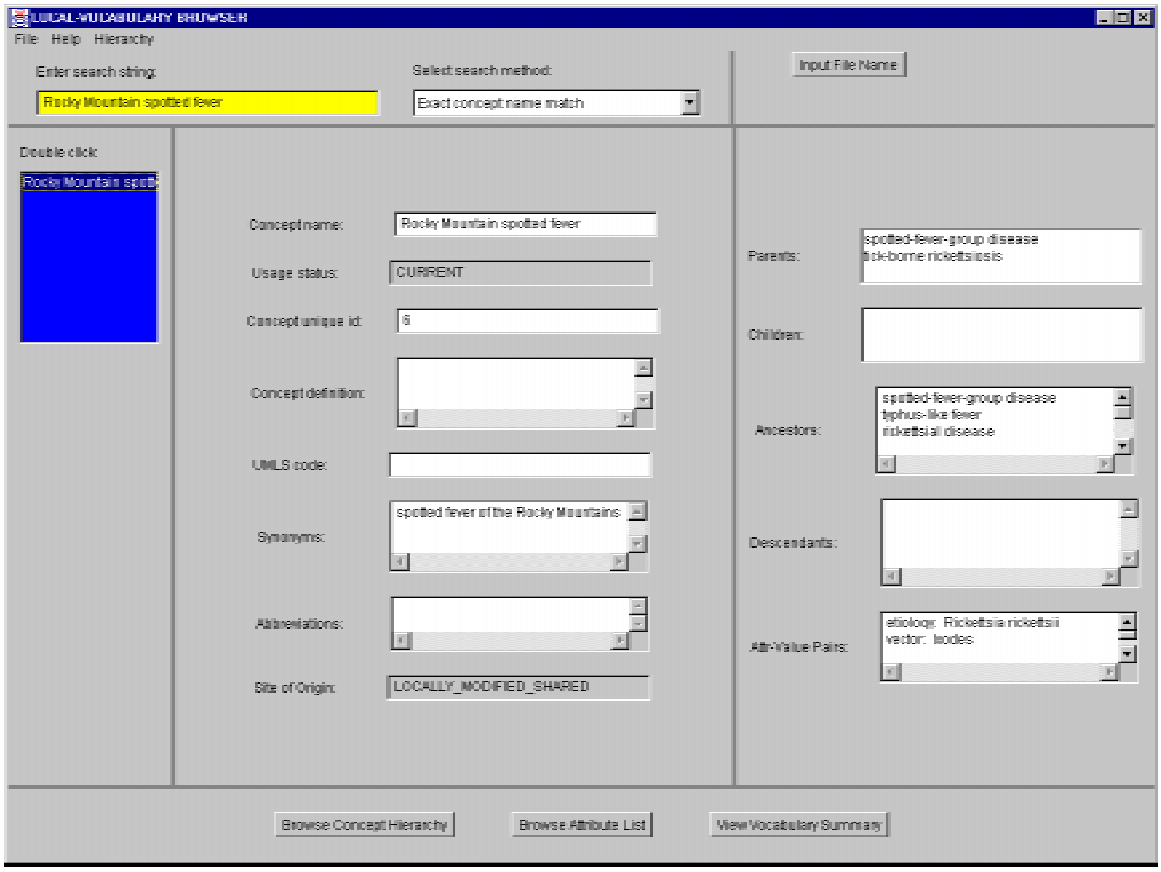

Figure 5.7. Local-vocabulary browser.

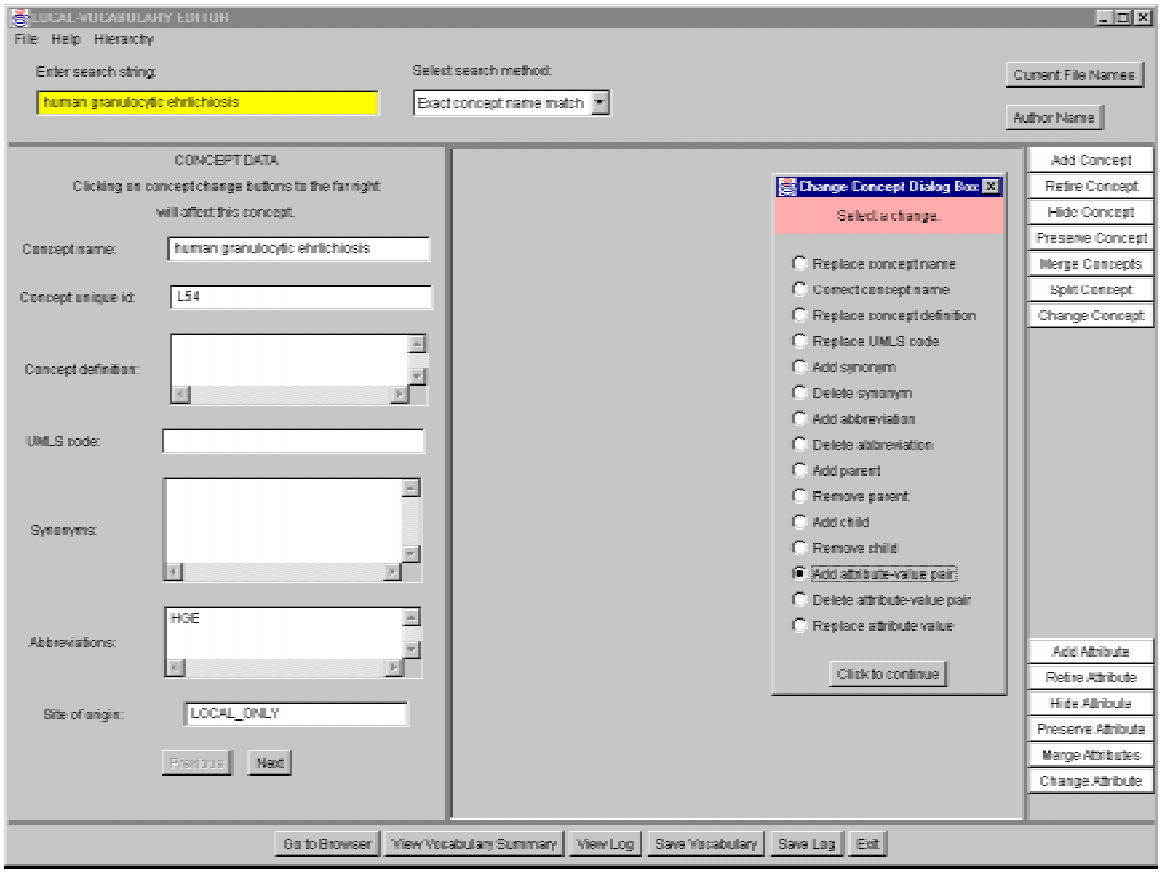

Figure 5.8. Local-vocabulary editor.

5.5.4 Local-Vocabulary Editor

The local-vocabulary editor looks like the shared-vocabulary editor, with several differences. There is a data field that displays the concept or attribute site of origin, and there are several additional change buttons on the right side panel. The additional change buttons are *Hide Concept*, *Preserve Concept*, *Hide Attribute*, and *Preserve Attribute* (Figure 5.8). The local-vocabulary editor enforces constraints according to the local extension of the CONCORDIA model, which in certain cases, differ from constraints on the shared vocabulary.

5.5.5 Synchronization-Support Tool

Figure 5.9 shows the synchronization-support tool. The synchronization-support tool reads in four files: (1) the shared vocabulary, (2) the local vocabulary, (3) the shared log, and (4) the local log. The local vocabulary is modified during the synchronization session, and the shared log provides the raw data that drives the changes. The tool reads in the shared vocabulary so that the user can view the shared vocabulary at any time;

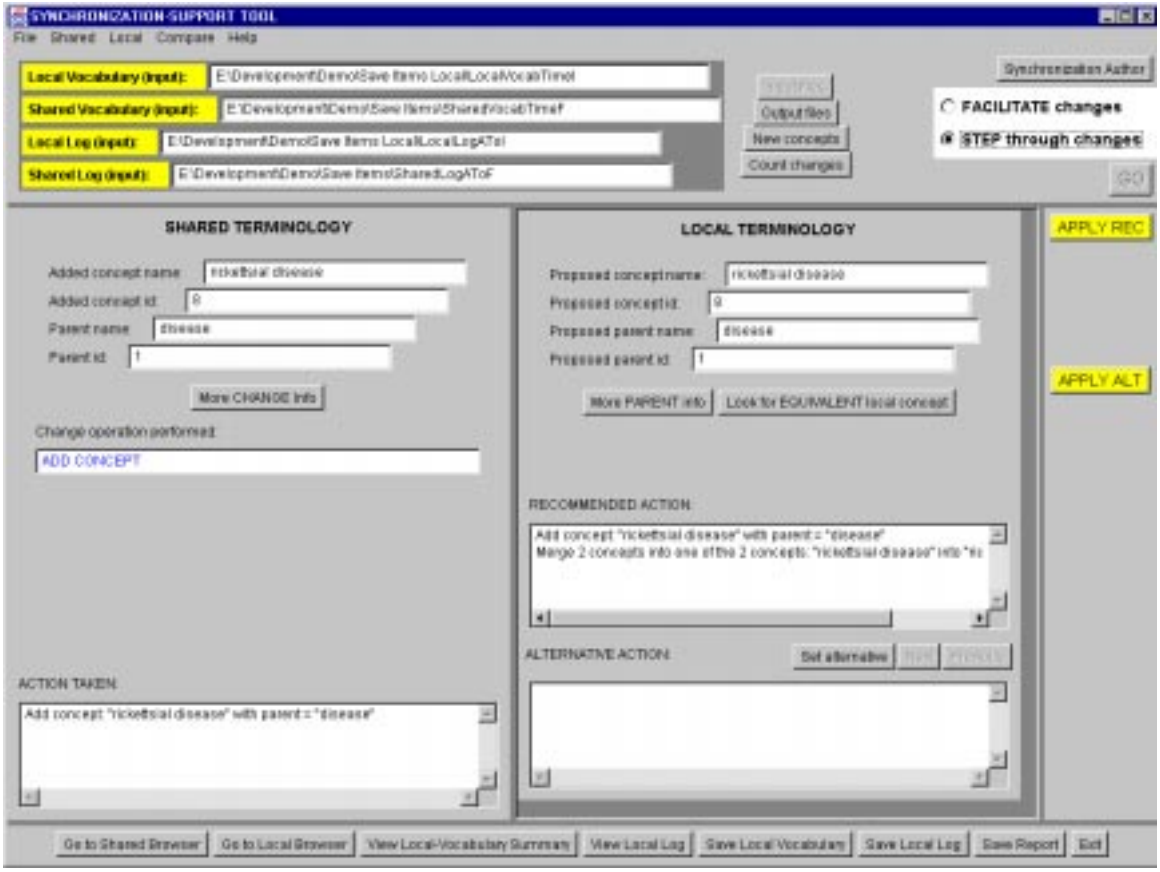

Figure 5.9. Synchronization-support tool.

however, the synchronization process makes no changes to the shared vocabulary. The tool reads in the local log because the local log contains information about changes made to the local vocabulary since the two vocabularies were last synchronized.

There are two choices available to the user. The first option is called *facilitate changes*. If the user selects this option, the system performs certain changes automatically, thereby facilitating the process as much as possible. The other option is called *step through changes*. Under this option, the system displays every change that was made to the shared vocabulary. The system does not perform any changes automatically; rather, the user makes a decision about every change, with the benefit of direct review.

In facilitate mode, the system automatically performs changes that do not affect the hierarchy. Those changes are *replace concept name* (if no duplicate name exists), *add synonym, delete synonym, add abbreviation, delete abbreviation,* and *replace UMLS code*. Other changes can be made automatically only in certain situations. For example, *add parent* and *add child* are performed automatically if there is no violation of constraints for attribute–value pairs, and no cycles will occur. *Remove parent* and *remove child* may

be performed automatically if no concept is left without at least one parent. *Add attribute–value pair* and *replace attribute* value may be performed automatically if no constraints are violated. *Delete attribute–value pair* can be performed automatically because this change cannot violate any constraints.

A change that occurs frequently in existing controlled vocabularies is *add concept*. In general, this change operation is difficult to automate. There are only two cases in which the system, as it is currently implemented, takes the liberty of handling the change without consulting the user. In each of these cases, the system adds the shared concept and merges the local concept into the shared concept.

The requirements for the first case of automated concept addition are as follows: (1) a concept that was added to the local vocabulary during the change interval has the same name as the concept that was added to the shared vocabulary, and (2) the concept that was added to the local vocabulary has the same parents, the same children, and the same attribute–value pairs that the new concept has in the shared vocabulary. These requirements are restrictive to reduce the risk of an error by the system.

The requirements for the second case are as follows: (1) the concept added to the shared vocabulary has the same concept name as the synonym of a concept that was added to the local vocabulary, and (2) the concept in the local vocabulary has the same concept name as a synonym of the concept in the shared vocabulary. This case assumes that two independent developers might add two concepts with the same name, but intend two different meanings, but would probably not add two concepts with two of the same names if those concepts have different meanings.

A more common situation occurs when the system encounters a change record for *add concept* in the log, but neither of the requirements for automatic addition is fulfilled. In this case, the system takes steps to retrieve information that will help the user to make a decision. First, the system checks to be sure that no other concept exists in the local vocabulary with the same name as the concept being added. If such a concept does exist, the system asks the user to determine whether the two concepts have the same meaning. If they do not, then the user is asked to change the name of the local concept.

If no concept by the same name exists in the local vocabulary, the system could try to identify concepts that have been added to the local vocabulary during the change interval that are most likely to have the same intended meaning as the added shared concept. It could use a combination of measures that assess similarity of names and synonyms, similarity of location in the hierarchy, and similarity of attribute sets to rank

order candidate concepts. As a last resort, the system could ask the user to review all the remaining concepts that were added to the local vocabulary since the previous synchronization to verify that no duplicate concept exists. In the current implementation of the synchronization-support tool, the system checks for exact name matches, but does not assess or rank order potential matches. The system gives the user the opportunity to review concepts added to the local vocabulary during the previous change interval to make sure that no concept with the same meaning, but different name, exists in the local vocabulary.

For each change in the shared log that is presented to the user, the system gives a recommended action and a set of zero or more alternatives. The possible actions, including recommended and alternative actions are those actions discussed in Chapter 4 and listed in Appendix H. The recommended action for the local vocabulary is the change that most closely resembles the change made to the shared vocabulary.

When a recommended action or alternative action contains more than one step, each step is displayed. The user reviews the options. If more than one alternative is available, the user may click buttons labeled *Next* and *Previous* to review each alternative. Finally, the user applies the recommendation or an alternative by clicking on one of two buttons: *Apply Rec* or *Apply Alt*. If the user chooses to apply an alternative, the system performs the alternative that is currently displayed.

The system performs the change, displays data about the change on the screen, and stores a change record in the local log. At any point during the session, the user may review the local log that has been created so far. Finally, when all change records from the shared log have been processed, the user clicks to save the local vocabulary, save the local log, save the report, and exits. The report is a printout of the changes that have been made to the local vocabulary during this synchronization session; it permits the user to review all the changes that were made.

5.6 Summary

Concept Manager is a set of software tools that permit the shared vocabulary maintainer to browse and edit the shared vocabulary, the local vocabulary maintainer to browse and edit the local vocabulary, and the local-vocabulary maintainer to synchronize the local vocabulary with the shared vocabulary.

The system is designed in such a way that the shared-vocabulary browsing components are part of the shared-vocabulary editor, the local-vocabulary browsing

components are part of the local-vocabulary editor, the shared-vocabulary browsing components are part of the local-vocabulary browser and editor, and the shared-vocabulary editing components are part of the local-vocabulary editor. In addition, the shared-vocabulary and local-vocabulary browsing components are part of the synchronizer.

The CONCORDIA model lends itself well to an object-oriented approach. In addition, the XML-based format for vocabulary files and log files works well for an object-oriented design. The log was designed in such a way that each type of change operation could have a slightly different set of data elements recorded in the log.

In Chapter 6, I describe the evaluation, which was performed using the Concept Manager software.

6 Evaluation

The purpose of this evaluation is to provide a proof-of-concept demonstration of the utility of establishing formal models to facilitate the sharing of changes. Different applications use change information in different ways; sharability is facilitated by a common understanding of data models for content and change, and by agreement on methods that operate on the data in those models. In this demonstration, I show that CONCORDIA can serve as such a common data model for content and change, and that it can provide sharable change methods.

6.1 Restatement of Hypothesis

In Chapter 1, I stated the hypothesis of this research. I restate the hypothesis here:

> Communication of changes between an organization that develops a shared vocabulary and a local site that uses and adapts that vocabulary requires a shared understanding of an explicit formal vocabulary structural model, change model, and log model; the utility of such formal models can be demonstrated by their implementation in a synchronization-support tool that enables a vocabulary developer to synchronize a local version of a shared vocabulary with the evolving shared version from which it was derived.

6.2 Evaluation Approach

To demonstrate the use of a shared model that supports change, I implemented Concept Manager, which is a suite of tools that depend on the CONCORDIA model and that support carefully controlled local divergence. I generated a test set of shared and local versions of a small medically oriented vocabulary (80 concepts in the final vocabulary), and used the tools to synchronize the local version with the shared version. For the vocabulary test set, I obtained content from three different textbooks of medicine: a textbook from 1917 for the initial vocabulary, and two contemporary textbooks by different publishers for divergent versions of the initial vocabulary. I discuss the results by considering the following questions: (1) Were the synchronization criteria fulfilled? (2) Was the model effective for this test set? (3) Did automation of certain tasks and support of other tasks facilitate synchronization for this test set?

Box 6.1. Domain-modeling rules for transferring content from a source textbook to a vocabulary.

1. Name concepts according to the names used in the text.
2. If the name of a concept in the text is plural, make it singular.
3. If the name of a concept is in a compound form, using the conjunction *and*, create two distinct concepts, and make each of those concepts singular.
4. If a concept is referred to by multiple names and one name is used more frequently than the other(s), choose the name that is most frequently used to be the concept name.
5. If a concept is referred to by two different names, and one name is followed by an alternate name in parentheses, choose the name that is not in parentheses to be the concept name.
6. If a concept is referred to by two different names, and one name is followed by an alternate name in parentheses, make the name that is in parentheses a synonym.
7. If a pattern for naming emerges, but is not consistently followed in the text, select a pattern and be consistent in the vocabulary, even if another one of these rules is violated.
8. Classify concepts in the vocabulary according to the hierarchy implied in the text. For example, the following hierarchical relationships may be adapted to the vocabulary: (1) a chapter title is superordinate to a section title; (2) a section title is superordinate to a subtitle; (3) a subtitle is superordinate to a term used in the body of the text; and (4) one concept that is described in the text as being more general than a second concept is superordinate to that second concept.
9. Classify concepts in the vocabulary according to the hierarchy implied in tables. For example, the following hierarchical relationships may be adapted to the vocabulary: (1) a concept in a column header is superordinate to concepts listed under that header; (2) a concept that is indented and located under another concept is subordinate to that other concept.
10. If a relationship between two concepts is described in the text, create an attribute that corresponds to the relationship.
11. If an attribute is referred to by more than one name and one name is used more frequently than the other(s), choose the name that is used most frequently to be an attribute name. If no name is used more frequently that the others, arbitrarily select one of the names to be the attribute name.
12. If information in the text implies that a relationship between two concepts exists, create an attribute–value pair for the first concept, where the relationship is the attribute, and the second concept is the value.
13. If a concept selected for the vocabulary is referred to by an abbreviation in either the text or tables, add that abbreviation to the concept.
14. If a set of related concepts can be grouped together semantically to facilitate management of the hierarchy, create a concept as a superconcept to the related concepts, even if that concept is not explicitly discussed in the text.
15. Avoid any concept whose name contains the word “Other,” if the concept means everything not included in other concepts defined in the vocabulary.

6.2.1 A Case Study

This evaluation is a case study that requires domain modeling of a particular subdomain and synchronization of vocabularies encoded for that subdomain. A case study can lead to new hypotheses and further testing, and therefore provides a basis for future work.

This evaluation is not a large-scale study to determine the validity of the model for all users or for all purposes, and it is not a comprehensive study of the usability of the software that I have developed. Because I have not studied the usefulness of the model for a broad variety of subdomains and for a broad variety of vocabulary services, the conclusions that I can draw regarding generalizability of the model are limited. In addition, because I have not studied the use of the software by many users, I cannot offer conclusive evidence about the usability of the tools. Nevertheless, testing the software with medical content from existing textbooks yields information about the model, the software, and the synchronization process that contributes to our understanding of the synchronization process.

6.2.2 Methods

In this section, I list the steps that I followed in the experiment. I state the goal of each step, and explain what I did to carry out each step. In Box 6.1, I state the rules that I followed to transfer knowledge from the textbooks to the vocabularies. Figures 6.1, 6.2, and 6.3 show the initial vocabulary, the modified shared vocabulary, and the modified local vocabulary, respectively. My evaluation followed the following steps:

1. *Implement software that enables communication of changes between a shared-vocabulary–development organization and local sites, and that makes use of a common understanding of shared and local models*. I implemented the Concept Manager suite of tools, which I described in Chapter 5. Concept Manager is faithful to the CONCORDIA model, which includes both shared and local models.

2. *Select sources of medical content in a limited subdomain of medicine from two points in time, including two sources by different authors from the second point in time*. I chose rickettsial diseases to be the subdomain, and I chose widely separated points in time: the second and last decades of the twentieth century. The sources of medical content were

1. *The Diagnostics and Treatment of Tropical Diseases* [Stitt 1917] (used for the initial versions of the shared and local vocabularies)
2. *Harrison's Principles of Internal Medicine* [Fauci 1998] (used for the modified version of the shared vocabulary)
3. *Cecil Textbook of Medicine* [Bennett 1996] (used for the modified version of the local vocabulary)

The current editions of *Cecil* and *Harrison's* were only two years apart, and I regarded them as contemporary sources written by different authors at approximately the same point in time. Because the different authors made different choices about the naming and classification of concepts, the authors are analogous to vocabulary maintainers at different sites. I refer to the three textbooks listed above as *Source 1*, *Source 2*, and *Source 3*, respectively.

3. *Select one of the two contemporary textbooks to be the source of content for the modified shared vocabulary, and the other to be the source for the modified local vocabulary*. I arbitrarily chose *Harrison's* (Source 2) to be the source of the shared vocabulary, and *Cecil* (Source 3) to be the source of the local vocabulary.
4. *Create the content of the initial shared vocabulary*. I modeled content knowledge contained in Source 1, according to the CONCORDIA shared-vocabulary structural model.
5. *Create the content of the initial local vocabulary*. I made the content of the initial local vocabulary the same as the content of the initial shared vocabulary, but adapted it to the CONCORDIA local-vocabulary structural model.
6. *Create the content of the modified shared vocabulary*. I modeled content knowledge contained in Source 2 according to the CONCORDIA structural model for a shared vocabulary, and identified changes that would complete the transition from the initial shared vocabulary to the modified shared vocabulary. The changes complied with the CONCORDIA shared-vocabulary change model.
7. *Create the content of the modified local vocabulary*. I modeled content knowledge contained in Source 3 according to the CONCORDIA local-vocabulary structural model, and identified changes that would complete the

transition from the initial local vocabulary to the modified local vocabulary. The changes complied with the CONCORDIA local-vocabulary change model.

8. *Enter the content of the initial shared vocabulary into the system using the shared-vocabulary editor, and save the resulting file in the shared-vocabulary format*. I selected concepts from *The Diagnostics and Treatment of Tropical Diseases* (Source 1) that were discussed in the same chapter as "spotted fever of the Rocky Mountains." Rocky Mountain spotted fever is now known to be a rickettsial disease. I entered those concepts into the shared-vocabulary editor. I saved the shared vocabulary at time 0 as *SV-0*. SV-0 contains 8 concepts.

9. *Read the shared vocabulary at time 0 into the local-vocabulary editor, and save the resulting file in the local-vocabulary format to create the local vocabulary at time 0*. I loaded SV-0 into the local-vocabulary editor, and saved the local vocabulary at time 0 as *LV-0*. Like SV-0, LV-0 contains 8 concepts.

10. *Make changes to the shared vocabulary at time 0, using content from the first contemporary source*. I made changes to SV-0, using the shared-vocabulary editor, according to changes I identified to create the modified shared vocabulary. I saved the resulting version as the shared vocabulary at time 1 (*SV-1*). SV-1 contains 62 concepts.

11. *Make changes to the local vocabulary at time 0, using content from the second contemporary source*. I made changes to LV-0 using the local-vocabulary editor, according to changes I identified to create the modified local vocabulary. I saved the resulting file as the local vocabulary at time 1 (*LV-1*). LV-1 contains 59 concepts.

12. *Synchronize the local vocabulary with the shared vocabulary, noting choices made*. The resulting local vocabulary represents the local version at time 1 after synchronization (*LV-1-synch*). LV-1-synch contains 80 concepts.

13. *Save the printed results of the synchronization session, and analyze the results*. The printed results produced by the system are reproduced, in part, in Figure 6.9, and are reproduced in full in Appendix K. I analyze the results in Section 6.5.

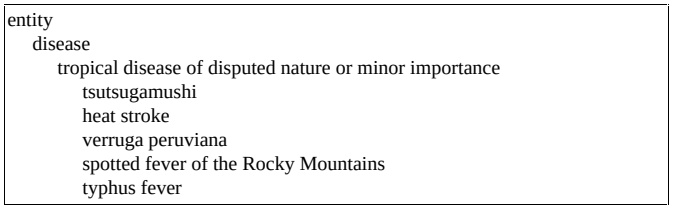

Figure 6.1. Initial shared vocabulary (SV-0). The concept hierarchy for the initial local vocabulary (LV-0) is the same as the concept hierarchy for SV-0.

Figure 6.2. Shared vocabulary (SV-1). (The second column is a continuation of the first.)

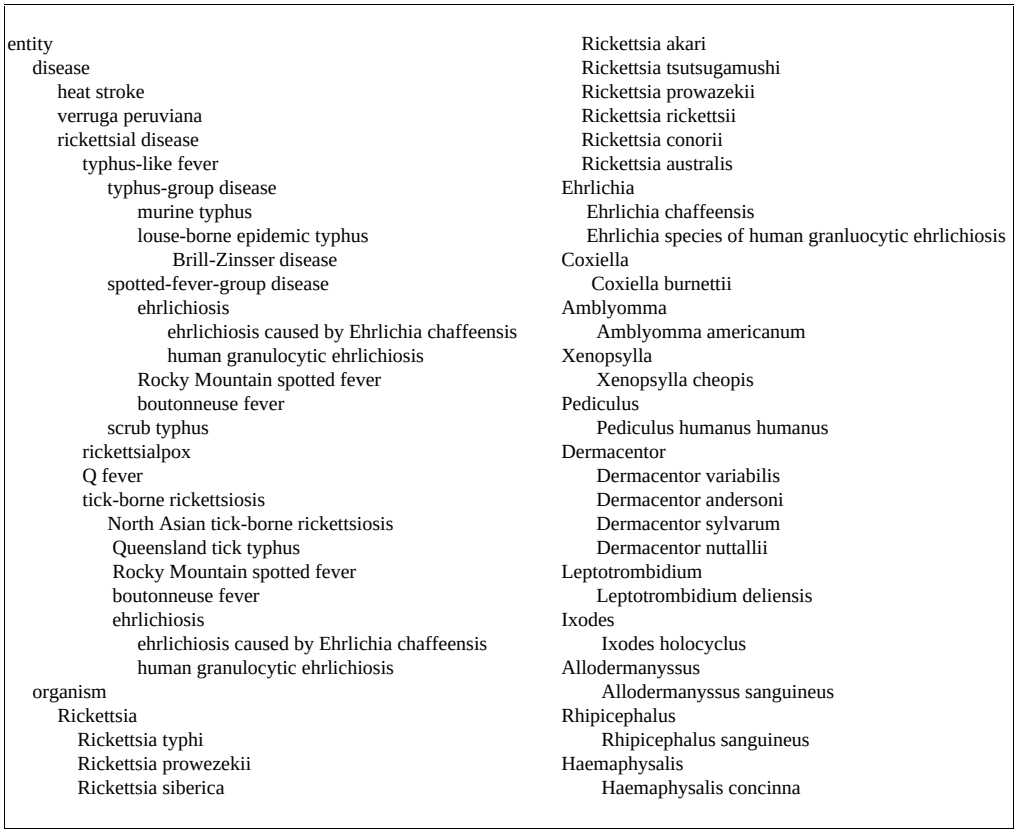

Figure 6.3. Local vocabulary (LV-1). (The second column is a continuation of the first.)

6.3 Vocabulary Test Set

The vocabulary test set comprises the initial shared vocabulary, the initial local vocabulary, the modified shared vocabulary, and the modified local vocabulary. Note that for this test set, I did not assign UMLS codes to the concepts, and I did not create text definitions. I describe here how I reconstructed knowledge contained in the textbooks to create the controlled vocabularies according to the CONCORDIA model.

6.3.1 Initial Shared Vocabulary (SV-0)

There are eight concepts in SV-0, as shown in Figure 6.1: *entity, disease, tropical disease of disputed nature or minor importance, heat stroke, tsutsugamushi, verruga peruviana, spotted fever of the Rocky Mountains,* and *typhus fever*. The 1917 textbook includes a section called *Tropical Diseases of Disputed Nature or Minor Importance*. In this section, the chapters are named *Verruga Peruviana and Oroya Fever, Heat Stroke and Heat Prostration, Tsutsugamushi or Japanese River Fever, Spotted Fever of the Rocky Mountains,* and *Typhus Fever*.

In CONCORDIA, a vocabulary requires a root concept. I called the root *entity*, which is the system's default name for the root. I changed the section title from plural to singular, and used it as the name of a new concept in the hierarchy. Then, I mapped each chapter to a new concept, where each concept represented only one entity, and each concept name was in the singular, lower-case form. Therefore, the chapter *Verruga Peruviana and Oroya Fever* became the concept *verruga peruviana*. *Heat Stroke and Heat Prostration* became *heat stroke*. *Tsutsugamushi or Japanese River Fever* became *tsutsugamushi*. I did not change *Spotted Fever of the Rocky Mountains* or *Typhus Fever*, except for converting them to lower case. The book explained that *verruga peruviana* and *Oroya fever* are the same disease, and I made *Oroya fever* a synonym of *verruga peruviana*. Because the book recognized heat stroke and heat prostration as different conditions, I did not make *heat prostration* a synonym of *heat stroke*. Because the text explained that *Japanese River fever* was another name for tsutsugamushi, I made *Japanese River fever* a synonym of *tsutsugamushi*.

The children of *tropical disease of disputed nature or minor importance* became *verruga peruviana, heat stroke, tsutsugamushi, spotted fever of the Rocky Mountains,* and *typhus fever* because the chapters were subordinate to the section. I wanted to include a concept that was more specific than *entity* to group the disease concepts. Therefore, I

added *disease* as a child of *entity*, and made *disease* a parent of *tropical disease of disputed nature or minor importance.*

Another chapter in this section was entitled *Climatic Bubo, Ainhum, Goundou,* and *Rat Bite Disease*. Because it was difficult to map these items to currently known medical conditions concepts with certainty, I chose not to include them in the initial test set.

6.3.2 Initial Local Vocabulary (LV-0)

LV-0 contains the same concepts as SV-0, but each concept in LV-0 has an additional data element—the site of origin. Since all concepts originated in the shared vocabulary, the value of the site of origin for each concept is *shared*. The concept hierarchy for the shared vocabulary is the same as the concept hierarchy for the local vocabulary at time 0 (Figure 6.1).

6.3.3 Modified Shared Vocabulary (SV-1)

Using *Harrison's Principles of Internal Medicine* to guide the creation of the modified vocabulary, I created SV-1. I followed the rules for transfer of content from the text to the vocabulary (Box 6.1).

I applied 129 changes to SV-0 to create SV-1. I list those changes in part in Box 6.2 and in their entirety in Appendix I. I explain next how I used the names and classification structure from the text of *Harrison's* to select changes and to generate the modified shared vocabulary. The concept hierarchy of the resulting shared vocabulary is shown in Figure 6.2.

6.3.3.1 Classification of Concepts (Shared Vocabulary)

The title of the relevant chapter in *Harrison's* is "Rickettsial Diseases" [Walker 1998]. Following the convention of naming concepts in their singular forms (modeling rule 2 in Box 6.1), I created the concept *rickettsial disease*. I added *rickettsial disease* to the vocabulary as a child of *disease*.

The chapter has five sections. The names of the sections are in capitals as follows:

1. TICK- AND MITE- BORNE SPOTTED FEVERS

2. FLEA- AND LOUSE-BORNE RICKETTSIAL DISEASES

3. CHIGGER-BORNE SCRUB TYPHUS

Box 6.2. A subset of changes made to the shared vocabulary, SV-0, to create the modified shared vocabulary, SV-1. The complete list of changes is given in Appendix I.

1. Add concept: CONCEPT rickettsial disease, PARENT disease
2. Add concept: CONCEPT mite-borne spotted fever, PARENT rickettsial disease
3. Add concept: CONCEPT tick-borne spotted fever, PARENT rickettsial disease
4. Add concept: CONCEPT flea-borne rickettsial disease, PARENT rickettsial disease
5. Add concept: CONCEPT louse-borne rickettsial disease, PARENT rickettsial disease
6. Add concept: CONCEPT ehrlichiosis, PARENT rickettsial disease
7. Add concept: CONCEPT Q fever, PARENT rickettsial disease
8. Replace concept name: OLD tsutsugamushi, NEW scrub typhus
9. Add parent: CONCEPT scrub typhus, PARENT rickettsial disease
10. Remove parent: CONCEPT scrub typhus, PARENT tropical disease of disputed nature or minor importance
11. Replace concept name: OLD spotted fever of the Rocky Mountains, NEW Rocky Mountain spotted fever
12. Add parent: CONCEPT Rocky Mountain spotted fever, PARENT tick-borne spotted fever
13. Remove parent: CONCEPT Rocky Mountain spotted fever, PARENT tropical disease of disputed nature or importance
14. Add concept: CONCEPT Mediterranean spotted fever, PARENT tick-borne spotted fever
15. Add concept: CONCEPT African tick-bite fever, PARENT tick-borne spotted fever
16. Add concept: CONCEPT Japanese spotted fever, PARENT tick-borne spotted fever
17. Add concept: CONCEPT Queensland tick typhus, PARENT tick-borne spotted fever
18. Add concept: CONCEPT Flinders Island spotted fever, PARENT tick-borne spotted fever
19. Add concept: CONCEPT rickettsialpox, PARENT mite-borne spotted fever
20. Replace concept name: OLD typhus fever, NEW epidemic typhus
21. Add parent: CONCEPT epidemic typhus, PARENT louse-borne rickettsial disease
22. Add concept: CONCEPT Brill-Zinsser disease, PARENT epidemic typhus
23. Add concept: CONCEPT murine typhus, PARENT flea-borne rickettsial disease
24. Remove parent: CONCEPT epidemic typhus, PARENT tropical disease of disputed nature or minor importance
25. Add concept: CONCEPT human monocytic ehrlichiosis, PARENT ehrlichiosis
26. Add concept: CONCEPT human granulocytic ehrlichiosis, PARENT ehrlichiosis
27. Add parent: CONCEPT verruga peruviana, PARENT disease
28. Remove parent: CONCEPT verruga peruviana, PARENT tropical disease of disputed nature or minor importance
29. Add parent: CONCEPT heat stroke, PARENT disease

4. EHRLICHIOSES

5. Q FEVER

From the section title TICK- AND MITE- BORNE SPOTTED FEVERS, I created two concepts in their singular forms, *tick-borne spotted fever* and *mite-borne spotted fever* (modeling rules 2 and 3). Similarly, from FLEA- AND LOUSE-BORNE RICKETTSIAL DISEASES, I created *flea-borne rickettsial disease* and *louse-borne rickettsial disease*. The section entitled CHIGGER-BORNE SCRUB TYPHUS contains a discussion of only one disease, which is most frequently referred to as *scrub typhus*. Since the terms *chigger-borne scrub typhus* and *scrub typhus* refer to the same entity, I created only one concept, and named it *scrub typhus*. The last two section titles of the chapter are EHRLICHIOSES and Q FEVER. From the title EHRLICHIOSES, I created the concept *ehrlichiosis*; from Q FEVER, I created *Q fever*.

The section TICK- AND MITE- BORNE SPOTTED FEVERS has three subtitles:

1. ROCKY MOUNTAIN SPOTTED FEVER

2. MEDITERRANEAN SPOTTED FEVER (BOUTONNEUSE FEVER) AND OTHER SPOTTED FEVERS

3. RICKETTSIALPOX

The concept *Rocky Mountain spotted fever* already existed in the old version of the shared vocabulary, although by a slightly different name. Therefore, using *replace concept name*, I changed *spotted fever of the Rocky Mountains* to *Rocky Mountain spotted fever*. I created a new concept that corresponded to the subtitle *Mediterranean Spotted Fever (Boutonneuse Fever) and Other Spotted Fevers*. Based on the authors' use of parentheses, I made *Mediterranean spotted fever* the concept and *boutonneuse fever* a synonym (modeling rules 5 and 6). I removed the conjunction *and* (modeling rule 3), and to avoid concepts whose names contain *Other* (modeling rule 15), I created only *Mediterranean spotted fever*. The conditions discussed in the text that are grouped as "other spotted fevers" are African tick-bite fever, Japanese or Oriental spotted fever, Queensland tick typhus, and a spotted fever observed on Flinders Island. I created concepts for each of these diseases, and named them *African tick-bite fever, Japanese spotted fever, Queensland tick typhus,* and *Flinders Island spotted fever*. I chose *Japanese spotted fever* to be the name of the concept referred to as "Japanese or Oriental spotted fever," and made *Oriental spotted fever* a synonym, because it was named second in the disjunction. The text refers to the spotted fever on Flinders Island as "the spotted

fever observed on Flinders Island (near Tasmania)," but a table used the term *Flinders Island spotted fever*. I chose the latter term.

Other changes included the addition of *murine typhus*, and the concept-name replacement of *typhus fever* to *epidemic typhus*.

6.3.3.2 Synonyms (Shared Vocabulary)

In certain cases, the text explains reasons for alternate names. For example, *Harrison's* discusses various names for Mediterranean spotted fever:

> The names for this disease vary with the region in which it occurs; examples include *Mediterranean spotted fever* (also known as *boutonneuse fever*), *Kenya tick typhus*, *Indian tick typhus*, *Israeli spotted fever*, and *Astrakhan spotted fever*. [Walker 1998]

I added the latter four terms to the synonym list for *Mediterranean spotted fever*. The synonym list already included *boutonneuse fever*.

In other cases, alternate terms occur in parentheses. For example, "*Dermacentor variabilis* (dog tick)" and "*Ixodes scapularis* (deer tick)" are listed as tick vectors in a table that compares human monocytic ehrlichiosis and human granulocytic ehrlichiosis [Walker 1998]. I used the official taxonomic names for concept names, and made the common English names in parentheses synonyms.

However, the description of human monocytic ehrlichiosis says, "The Lone Star tick (*Amblyomma americanum*) is the major vector ..." The name that appears first is the common name *Lone Star tick*, and the name that appears second in parentheses is the taxonomic name *Amblyomma americanum*. However, to be consistent with the preceding examples, I made the taxonomic name the concept name, *Amblyomma americanum*, and the common English term, *Lone Star tick*, a synonym.

6.3.3.3 Obsolete Concept (Shared Vocabulary)

Not surprisingly, the term *tropical disease of disputed nature or minor importance* from the old version of the vocabulary is not used in the 1998 edition of *Harrison's*. However, before I could retire the concept, I had to remove the concept from the list of parents of each of its children. Therefore, I applied *remove parent* to *scrub typhus* (formerly *tsutsugamushi* in SV-0), to *Rocky Mountain spotted fever* (formerly *spotted fever of the Rocky Mountains* in SV-0), to *epidemic typhus* (formerly *typhus fever* in SV-0), to *verruga peruviana*, and to *heat stroke*. Then, I retired the obsolete concept.

6.3.3.4 Additional Concepts to Assist Organization (Shared Vocabulary)

In the description of each rickettsial disease, the text identifies the rickettsial organism that causes each disease. The only concept in the vocabulary that was more general than each organism was *entity*. To group rickettsial organisms so that each organism would not be a sibling of *disease*, I created a concept *living organism*. This concept encompasses organisms that cause disease and that transmit disease.

In addition, I decided to include the concepts *monocyte* and *granulocyte*, which are target cells of Ehrlichia organisms in ehrlichiosis. To form additional subdivisions of the hierarchy, I added the concept *anatomic part* as a child of entity, and added *cell* as a child of anatomic part. I made *cell* a parent of *monocyte* and *granulocyte*.

Two other concepts I added that are not specifically discussed in the textbook are *Rickettsia* and *Orientia*. I added these two concepts to group species that are in the genus. However, I did not create distinct concepts for every genus, which would have been more consistent.

6.3.3.5 Naming Problem (Shared Vocabulary)

A naming problem arose for the organism that causes human granulocytic ehrlichiosis. In *Harrison's*, the organism is referred to as an " E. equi-like organism" in a table, an "Ehrlichia equi-like organism" in one part of the text, and the "agent of HGE" in another part of the text. I preferred to avoid the vagueness of "Ehrlichia equi-like organism," and instead preferred "agent of HGE." However, to be as specific as possible and to avoid using an abbreviation, I selected *agent of human granulocytic ehrlichiosis*.

6.3.3.6 Abbreviations (Shared Vocabulary)

Three abbreviations are introduced in this chapter—*RMSF*, *HGE*, and *HME*. When each abbreviation is introduced, the full term is used in the text, followed by the abbreviation in parentheses. Later in the text, the abbreviation alone is used. I added these abbreviations to the corresponding diseases in the vocabulary.

6.3.3.7 Attribute Names (Shared Vocabulary)

I identified three relationships that would help to distinguish related concepts in the chapter if I used them in attribute–value pairs for concepts. The rickettsial diseases can be distinguished from one another by the organisms that cause the diseases, and by organisms that transmit the organisms that cause the diseases. Therefore, I created

attributes for etiology and transmission. I also created an attribute for major target cell, to distinguish between the two ehrlichioses.

Phrases used in the text suggest names for attributes, but my choices might differ from another person's choices. From the phrase "etiologic agent of scrub typhus" and similar phrases, I selected *has-etiology*. From multiple instances of the phrase "is transmitted by," I chose *transmitted-by*. From the phrase "*Ehrlichia chaffeensis*, which targets mainly macrophages and monocytes" I selected *major-target-cell*.

6.3.3.8 Attribute–Value Pairs (Shared Vocabulary)

An attribute of a concept links that concept to the attribute's value. In the shared vocabulary, there are three attributes that are used in relationships between pairs of concepts. The attribute *has-etiology* links a rickettsial disease and a rickettsial organism. The attribute *transmitted-by* links a rickettsial organism with an organism such as a tick, a mite, or a chigger. The attribute *major-target-cell* links an organism that causes ehrlichiosis with either *monocyte* or *granulocyte*. The following are examples of attributes (underlined) linking pairs of concepts:

1. *Rocky Mountain spotted fever* <u>*has-etiology*</u> *Rickettsia rickettsii*
2. *Rocky Mountain spotted fever* <u>*transmitted-by*</u> *Dermacentor andersoni*
3. *Mediterranean spotted fever* <u>*has-etiology*</u> *Rickettsia conorii*
4. *Mediterranean spotted fever* <u>*transmitted-by*</u> *Rhipicephalus sanguineus*

6.3.4 Modified Local Vocabulary (LV-1)

I used *Cecil* to create LV-1, and followed the rules for the transfer of content from textbook to vocabulary (Box 6.1).

I applied 119 changes to LV-0 to create LV-1. I list those changes in part in Box 6.3, and in their entirety in Appendix J. I explain here how I selected the changes and modified the local vocabulary. The resulting local vocabulary is shown in Figure 6.3.

6.3.4.1 Classification of Concepts (Local Vocabulary)

In *Cecil*, the relevant chapter is entitled "Rickettsial Diseases," which is the same name as the name of the corresponding chapter in *Harrison's*. This chapter employs several classification schemes. It organizes information into sections and subsections, and displays information about the diseases in several tables.

Box 6.3. A subset of changes made to the local vocabulary, LV-0, to create the modified local vocabulary, LV-1. The complete list of changes is given in Appendix J.

1. Add concept: CONCEPT rickettsial disease, PARENT disease
2. Add concept: CONCEPT typhus-like fever, PARENT rickettsial disease
3. Add concept: CONCEPT typhus-group disease, PARENT typhus-like fever
4. Add concept: CONCEPT murine typhus, PARENT typhus-group disease
5. Add concept: CONCEPT louse-borne epidemic typhus, PARENT typhus-group disease
6. Add concept: CONCEPT Brill-Zinsser disease, PARENT louse-borne epidemic typhus
7. Add concept: CONCEPT scrub typhus, PARENT typhus-like fever
8. Add concept: CONCEPT spotted-fever-group disease, PARENT typhus-like fever
9. Add concept: CONCEPT ehrlichiosis, PARENT spotted-fever-group disease
10. Replace concept name: OLD spotted fever of the Rocky Mountains, NEW Rocky Mountain spotted fever
11. Add parent: CONCEPT Rocky Mountain spotted fever, PARENT spotted-fever-group disease
12. Remove parent: CONCEPT Rocky Mountain spotted fever, PARENT tropical disease of disputed nature or minor importance
13. Add concept: CONCEPT boutonneuse fever, PARENT spotted-fever-group disease
14. Add concept: CONCEPT rickettsialpox, PARENT rickettsial disease
15. Add concept: CONCEPT Q fever, PARENT rickettsial disease
16. Add concept: CONCEPT tick-borne rickettsiosis, PARENT rickettsial disease
17. Add concept: CONCEPT North Asian tick-borne rickettsiosis, PARENT tick-borne rickettsiosis
18. Add concept: CONCEPT Queensland tick typhus, PARENT tick-borne rickettsiosis
19. Add parent: CONCEPT Rocky Mountain spotted fever, PARENT tick-borne rickettsiosis
20. Add parent: CONCEPT boutonneuse fever, PARENT tick-borne rickettsiosis
21. Add parent: CONCEPT ehrlichiosis, PARENT tick-borne rickettsiosis
22. Add concept: CONCEPT organism, PARENT entity
23. Add concept: CONCEPT Rickettsia, PARENT organism
24. Add concept: CONCEPT Rickettsia typhi, PARENT Rickettsia
25. Add concept: CONCEPT Rickettsia prowazekii, PARENT Rickettsia
26. Add concept: CONCEPT Ehrlichia, PARENT organism
27. Add concept: CONCEPT Ehrlichia chaffeensis, PARENT Ehrlichia
28. Add concept: CONCEPT Ehrlichia species of human granulocytic ehrlichiosis, PARENT Ehrlichia
29. Add concept: CONCEPT Rickettsia siberica, PARENT Rickettsia

The chapter is divided into six sections, which have the following section titles:

1. The Typhus Group
2. Rocky Mountain Spotted Fever
3. Other Tick-borne Rickettsioses
4. Rickettsialpox
5. Scrub Typhus
6. Q Fever

The Typhus Group has two subtitles in all capitals:

1. EPIDEMIC LOUSE-BORNE TYPHUS
2. MURINE TYPHUS

One table in the chapter divides the diseases into three groups [Hornick 1996c]:

1. Typhus-like fevers
2. Ehrlichiosis
3. Q fever

The first group, *Typhus-like fevers*, contains three subgroups:

1. Typhus group
2. Scrub typhus
3. Spotted-fever group

Another table divides the diseases into five groups [Hornick 1996b]:

1. Typhus group
2. Spotted-fever group (selected examples)
3. Rickettsialpox
4. Scrub typhus (Tsutsugamushi disease)
5. Q fever

The typhus group has three members:

1. Murine typhus
2. Epidemic typhus
3. Brill-Zinsser disease

The spotted-fever group has three members:

1. Rocky Mountain spotted fever

2. Ehrlichiosis

3. Boutonneuse fever

Thus, even in a chapter written by a single author [Hornick 1996a], there are multiple classification schemes. The schemes are similar, but not exactly the same.

I used the structure of the first table to organize upper-level concepts in the vocabulary. I added the members of the typhus group and the members of the spotted-fever group, which were given in the second table. A disease cannot be a direct child of a group of diseases because the relationship would not be a subsumption relationship. Therefore, I created *typhus-group disease* and *spotted-fever–group disease*.

The text includes a section entitled "Other Tick-borne Rickettsioses," which includes discussions of boutonneuse fever, North Asian tick-borne rickettsiosis, and Queensland tick typhus. I avoided a grouping based on an "Other" category (modeling rule 15), and created *tick-borne rickettsiosis*. I added *boutonneuse fever, North Asian tick-borne rickettsiosis*, and *Queensland tick typhus* as children of *tick-borne rickettsiosis*. Because Rocky Mountain spotted fever and ehrlichiosis are also described as tick borne in the text, I included them as children of *tick-borne rickettsiosis*.

In the first table, *Brill-Zinsser disease* is a sibling of *Epidemic typhus*. However, the text says:

> Patients who recover from classic typhus have the opportunity to develop Brill-Zinsser disease and at that time have rickettsemia and are able again to infect body lice. However, this happens rarely, for few cases of Brill-Zinsser disease have been detected among the many hundreds of thousands of soldiers who acquired typhus in World War II; one estimate suggested a rate of 10 per 10,000 cases of primary typhus. More cases may be recognized as the geriatric population continues to increase. [Hornick 1996a]

Based on this explanation, I concluded that Brill-Zinsser disease is a particular form of epidemic typhus, and made *Brill-Zinsser disease* a child of *louse-borne epidemic typhus*. Another vocabulary developer might have reached a different conclusion.

6.3.4.2 Synonyms (Local Vocabulary)

The textbook gives synonyms for certain diseases and organisms. It suggests synonyms through the use of parentheses or commas, and by the explicit identification of alternate names as synonyms.

In the discussion of scrub typhus, the organism *Rickettsia tsutsugamushi* has an alternate name specified in parentheses:

> ETIOLOGY. *Rickettsia tsutsugamushi* (*R. orientalis*) is a small gram-negative, obligate intracellular organism. Unlike other rickettsial infections, infection with *R. tsutsugamushi* does not induce solid protection against additional bouts of scrub typhus. This results from the variable antigenic compositions of the strains. [Hornick 1996a]

I made *Rickettsia orientalis* a synonym of *Rickettsia tsutsugamushi*.

In the discussion of organisms that cause Rocky Mountain spotted fever, alternate names are set off by commas.

> Several species of ticks are commonly involved in transmission of disease: *Dermacentor andersoni*, the wood tick, in the Rocky Mountain states; *D. variabilis*, the dog tick, in the East and Oklahoma; *Amblyomma americanum* in Texas and Oklahoma; and *Rhipicephalus sanguineus* in Texas and Mexico." [Hornick 1996a]

I made *wood tick* a synonym of *Dermacentor andersoni*, and *dog tick* a synonym of *Dermacentor variabilis*.

In the discussion of louse-borne epidemic typhus, synonyms are identified explicitly in the text.

> Synonyms include *classic*, *historic*, and *European typhus*; *jail*, *war*, *camp*, and *ship fever*; *Flichfieber* (German); *typhus exanthematique* (French); and *tifus exantematico* and *tabardillo* (Spanish). Many of these names indicate the location of the outbreaks—military and concentration camps, crowded ships with poor and starved immigrants, outbreaks in persons living in occupied countries during wartime, and so forth. [Hornick 1996a]

I included the explicitly stated synonyms in the synonym list of *louse-borne epidemic typhus*.

The author suggests a contradictory classification of ehrlichiosis. In the section entitled "Other Tick-borne Rickettsioses," there is a subtitle "HUMAN EHRLICHIOSIS (SPOTLESS ROCKY MOUNTAIN SPOTTED FEVER)." I added *spotless rocky Mountain spotted fever* as a synonym of *ehrlichiosis*. Because this name contains the adjective *spotless*, it is difficult to determine if the condition should be grouped with spotted fevers or not. In fact, the author classifies ehrlichiosis in the spotted-fever group in one table, but not in another table [Hornick 1996a]. I put ehrlichiosis in the spotted-fever group.

6.3.4.3 Obsolete Concept (Local Vocabulary)

In *Cecil*, as in *Harrison's*, the concept *tropical disease of disputed nature or minor importance* did not appear. I removed the children of *tropical disease of disputed nature or minor importance*, and retired the obsolete concept.

6.3.4.4 Additional Concepts to Assist Organization (Local Vocabulary)

To assist with the organization of concepts in the hierarchy, I added several additional concepts that are not used explicitly in the text. I subgrouped the etiologic organisms by genus, creating *Rickettsia, Coxiella, Xenopsylla, Pediculus,* and *Ehrlichia*. In addition, I grouped the vectors by genus, creating *Dermacentor, Amblyomma, Ixodes, Leptotrombidium, Allodermanyssus,* and *Haemaphysalis*. I grouped these genera under a concept *organism*. I chose the name *organism* because it is the column header in a table that specifies etiologic organisms. Another domain modeler might have made different choices.

6.3.4.5 Naming Problems (Local Vocabulary)

Two naming problems arose due to the lack of specific names in the text. The author of the chapter in *Cecil* discusses two types of ehrlichioses, but does not give the first type a specific name. He refers to the first type as simply *ehrlichiosis*. The name *ehrlichiosis* is not specific enough for the first type, because both types are ehrlichioses. Also, the author gives no specific name to identify the organism that causes human granulocytic ehrlichiosis.

The following paragraph describes, but does not name, the first type of ehrlichiosis:

> The first reported case of infection with a species of Ehrlichia in the United States occurred in 1986. A patient was found to have

> intracytoplasmic inclusions (morula) in monocytes ... In 1990, the first isolation was made on a continuous cell line of canine macrophage cells using blood drawn from a patient with a 3-day history of fever, headache, pharyngitis, nausea, and vomiting. This isolate was named *E. chaffeensis* because the patient was an Army reservist stationed at Ft. Chaffee, Arkansas. This strain is different from but closely related to *E. canis* and is now used as the antigen for serologic studies. Patients suspected of having ehrlichiosis, but no antibodies when tested with *E. canis*, do have antibodies to this human isolate. The organism can be stained in circulating monocytes. [Hornick 1996a]

The second type of ehrlichiosis is described in the following paragraph, and it does have a specific name—*human granulocytic ehrlichiosis*.

> In 1994, another Ehrlichia species was associated with human disease ... closely related to the *E. equi / E. phagocytophilia* group ... The unique feature of infection with this strain was the location of the morulae. They were found in granulocytes in the circulation and in the bone marrow. Because of this, the current name for the disease is *human granulocytic ehrlichiosis* (*HGE*). [Hornick 1996a]

Thus, the author views these conditions as distinct diseases. To distinguish between the two types, I called the first type *ehrlichiosis caused by Ehrlichia chaffeensis*, and the second type *human granulocyte ehrlichiosis*. In addition, I created concepts for the organisms that cause these diseases. One of these organisms is referred to as *Ehrlichia chaffeensis* in the text. The other organism is not explicitly named, and I named it *Ehrlichia species of human granulocytic ehrlichiosis.*

6.3.4.6 Abbreviations (Local Vocabulary)

Abbreviations used in this chapter were *RMSF* and *HGE*. I added them to the relevant concepts.

6.3.4.7 Attribute Names (Local Vocabulary)

I selected two attributes. I named the first attribute *etiology* because most of the sections describing the diseases have subsections named “Etiology.” I named the second attribute *vector*. A column header in one of the tables is “Arthropod Vector.”

6.3.4.8 Attribute–Value Pairs (Local Vocabulary)

For attribute–value pairs, I identified information in the text that expresses the etiology of diseases and the vectors that transmit organisms causing those diseases. Examples of attributes that link pairs of concepts are as follows:

1. *Queensland tick typhus etiology Rickettsia australis*
2. *rickettsialpox etiology Rickettsia akari*
3. *scrub typhus etiology Rickettsia tsutsugamushi*
4. *Queensland tick typhus vector Ixodes holocyclus*
5. *rickettsialpox vector Allodermanyssus sanguineus*
6. *scrub typhus vector Leptotrombidium deliensis*

6.3.4.9 Merges (Local Vocabulary)

Two of the new concepts that I added to local vocabulary, based on information in *Cecil*, duplicated the meaning of two older concepts in the vocabulary.

Tsutsugamushi, described in the 1917 textbook, is the same concept as *scrub typhus* described in *Cecil*. Therefore, I merged the new concept *scrub typhus* into the old concept *tsutsugamushi* to keep the concept identifier from the shared vocabulary. However, because I kept the old concept, the merge yielded a concept with the old name. Therefore, I had to perform *replace concept name* to update the name of the old concept. The new name became *scrub typhus*. The old name *tsutsugamushi* became a synonym.

Similarly, I merged the local concept *louse-borne epidemic typhus* into the shared concept *typhus fever*, and then replaced the old name with the new name. The new name became *louse-borne epidemic typhus*, and the old name *typhus fever* became a synonym.

6.4 Outputs

The outputs of synchronization are the synchronized local vocabulary, and the synchronization report.

6.4.1 Synchronized Local Vocabulary (LV-1-synch)

I used the synchronization-support tool to process the shared-vocabulary log, and to synchronize the local vocabulary (*Cecil*) with the shared vocabulary (*Harrison's*). Figure 6.4 shows the concept hierarchy after synchronization. In this section, I discuss the

kinds of changes that occurred during synchronization, and the level of support that the system offered when it performed the changes.

Table 6.1 shows the number of occurrences of each type of change operation applied to the local vocabulary during synchronization. These changes reflect the effect of the local maintainer processing the shared log, using the synchronization-support tool. The changes are not exactly the same changes that the shared-vocabulary maintainer applied to the shared vocabulary during the previous change interval. For example, the synchronization changes include the merges of local concepts into shared concepts that result from processing *add concept* change records where the added concept also exists in the local vocabulary; they include the individual changes given in the sublogs of compound changes; and they include choices made by the local-vocabulary maintainer who performs the synchronization. The frequencies given are for the test set only, and do not necessarily reflect frequencies that would typically occur in other test sets, or in the clinical setting.

6.4.1.1 Automated Changes

The most common automated change was *add synonym*. There were 13 cases of *add synonym*. Other alternate-name changes that are performed automatically by the tool are *add abbreviation*, *delete synonym*, and *delete abbreviation*. In the rickettsial-disease example, there were only three cases of *add abbreviation*, no cases of *delete synonym*, and no cases of *delete abbreviation*.

When the change operation *replace concept name* is applied to the shared vocabulary, it appears as a compound operation in the log with a sublog of the two operations, *correct concept name* and *add synonym*. If the local-vocabulary maintainer works with the synchronization-support tool in facilitate mode to perform *replace*

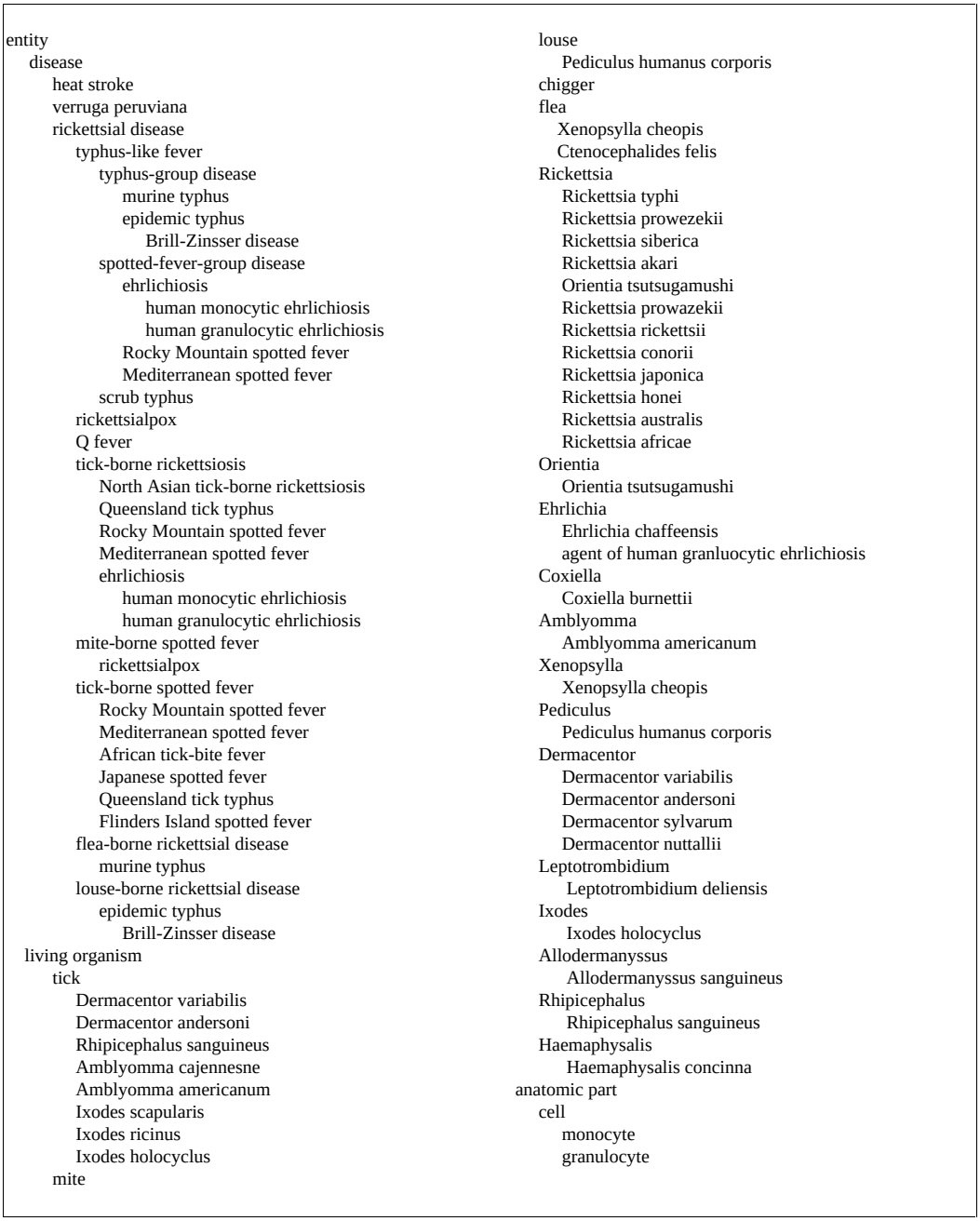

Figure 6.4. Synchronized vocabulary (LV-1-synch). (The second column is a continuation of the first.)

Table 6.1. Number of occurrences of change operations applied to the local vocabulary during the synchronization session.

Change Operation	Number of Occurrences	Automated	Supported
Add concept	55	11	44
Merge local concept into shared concept	32	11	21
Add attribute–value pair	17	17	0
Add synonym	13	13	0
Add parent	6	6	0
Remove parent	4	4	0
Add attribute	3	0	3
Merge local attribute into shared attribute	2	0	2
Add abbreviation	2	2	0
Correct concept name	1	1	0
Retire concept	1	1	0
Delete synonym	0	0	0
Delete attribute–value pair	0	0	0
Replace attribute value	0	0	0
Merge 2 concepts into 1 of the 2 concepts	0	0	0
Merge 2 concepts into new concept	0	0	0
Split concept into 2 new concepts	0	0	0
Merge 2 attributes into 1 of the 2 attributes	0	0	0
Merge 2 attributes into new attribute	0	0	0
Total	136	66	70

concept name, the system applies both operations from the sublog automatically. However, if the maintainer chooses to work in step mode, the system offers the maintainer the choice of changing the concept name to the new name (using *correct concept name*), or not, and of adding the old name to the synonym list (using *add synonym*), or not.

In the experiment, the shared log contained a change record for *replace concept name* applied to the concept *typhus fever*. In the shared vocabulary, the name *typhus fever* was changed to *epidemic typhus*. The same concept was also changed in the local vocabulary during the change interval. The name *typhus fever* in the local vocabulary was changed to *louse-borne epidemic typhus*. The change report, which displays concepts by name instead of by code, shows that *correct concept name* was applied to the concept named *louse-borne epidemic typhus* in the local vocabulary. This local concept acquired the shared name *epidemic typhus*. Thus, in facilitate mode, which aims for maximal consistency between the shared and local vocabularies, the system automatically assigns the new shared name to the local version of the concept, even if that means overriding a name change made at the local level.

In another example of *replace concept name*, the shared log contained a record indicating that the concept *spotted fever of the Rocky Mountains* was changed to *Rocky Mountain spotted fever*. However, the same change had also occurred in the local vocabulary. Therefore, the system recognized that, in this case, no action was required to process the change.

Text definitions and UMLS code were not studied in this experiment, but the addition and deletion of these data elements is similar to the addition and deletion of synonyms and abbreviations. In facilitate mode, the system preferentially assigns choices made by the shared-vocabulary maintainers to shared concepts in the local vocabulary.

The most common change was *add concept*, but only a subset of add-concept changes could be processed automatically. As described in Section 5.5.5, the only situations in which a concept can be added automatically (in this implementation) are (1) the concept names are the same and all parents, children, and attribute–value pairs the same, and (2) the concept name and a synonym of the added shared concept are the same as a synonym and the concept name of a local concept.

In this experiment, there were a number of cases that fit the first situation. For example, six species in the *Rickettsia* genus *(Rickettsia typhi, Rickettsia prowazekii, Rickettsia akari, Rickettsia rickettsii, Rickettsia conorii, Rickettsia australis*) had the

same name in the shared vocabulary that they had in the local vocabulary, and had the same parent *Rickettsia*. Although initially there were two *Rickettsia* concepts—one in the shared vocabulary and one in the local vocabulary—the order of changes in the shared log guaranteed that the parent *Rickettsia* would be added (and merged) first, before the rickettsial species were added as its children.

Subsequently, when the system encountered an *add concept* record for a rickettsial species, such as *Rickettsia typhi*, the system determined that there was a concept *Rickettsia typhi* in the local vocabulary with a different unique identifier, but with the same name and the same parent (*Rickettsia*) that *Rickettsia typhi* had in the shared vocabulary. *Rickettsia typhi* had only one parent, no children, and no attribute–value pairs, and the requirement for automatic merging was met. The system automatically merged the local concept *Rickettsia typhi* into the shared concept, and kept the shared unique identifier.

Three other rickettsial species—*Rickettsia japonica*, *Rickettsia honei*, and *Rickettsia africae*—were present in the modified shared vocabulary, but not in the modified local vocabulary. These concepts could not be added automatically. The system gave the user the opportunity to search for equivalent concepts, but there were none.

A case that fit the second situation for automatic concept addition and merging was the addition of *Mediterranean spotted fever* from the shared vocabulary to the local vocabulary. There was no concept named *Mediterranean spotted fever* in the local vocabulary, but there was a local concept that included a synonym *Mediterranean spotted fever*. The concept name of this local concept was *boutonneuse fever*, and a synonym of the shared concept was *boutonneuse fever*. Because the concept name of one was the synonym of the other, and vice versa, the system concluded that the two concepts were the same. The system automatically merged the local concept into the shared concept. Screen shots in step mode are shown in Figures 6.5 and 6.6.

Change records for the operations *add parent* and *add child* are processed automatically in a limited number of situations. For example, if a parent was added to a concept in the shared vocabulary, and it was also added to the concept in the local vocabulary, clearly, it does not need to be added to the local vocabulary again. In facilitate mode, the system does not display the change record at all. There were no examples in the test set.

Figure 6.5. System recommends merging *Mediterranean spotted fever* and *boutonneuse fever*.

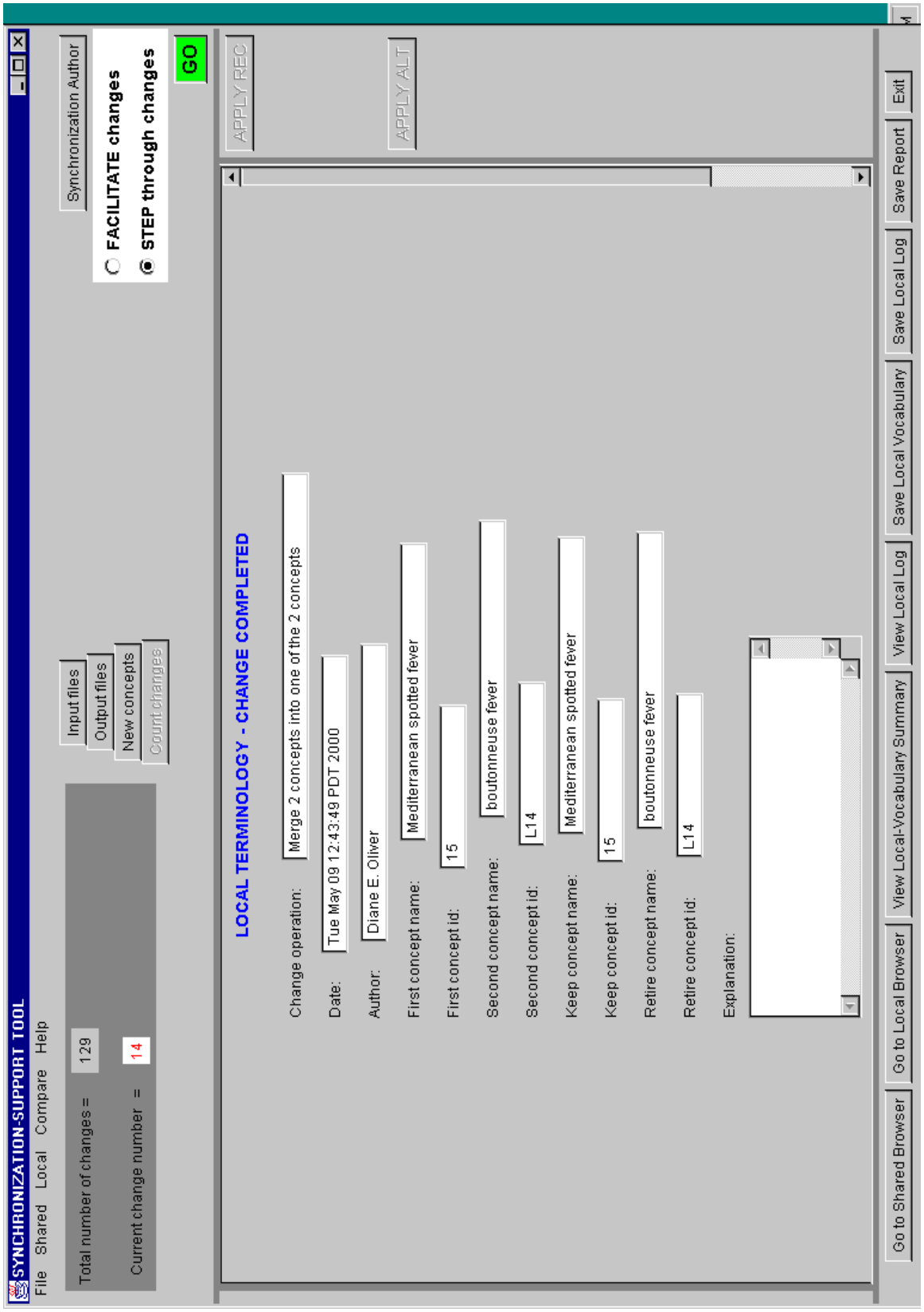

Figure 6.6. System shows that the merge of *Mediterranean spotted fever* and *boutonneuse fever* has been completed.

If the parent was not added already, the system adds the parent if there is no conflict. The same is true for *add child*. If there is a conflict, such as the creation of a cycle, or violation of inheritance rules for attribute–value pairs, the system requires the user to solve the conflict first, and gives the user choices for doing so. The test set did not contain examples of such conflicts.

Remove parent and *remove child* can potentially leave a concept with zero parents. The CONCORDIA model requires at least one parent for each concept. If removing a parent or child does not leave any concept without parents, the operation can be performed automatically in facilitate mode. In this test set, the only examples of such removals were the removals of the parent *tropical disease of disputed nature or minor importance* from its children. However, the same changes had occurred in the local vocabulary already, and did not have to be repeated when the shared log was processed.

The operations *add attribute–value pair* and *replace attribute value* can be performed automatically if there are no conflicts with inherited attribute–value pairs in ancestors or in descendants. There were 17 examples of *add attribute–value pair* in this test set. All were done automatically. However, the cautious local-vocabulary maintainer might prefer to run such changes in step mode to guarantee that no problems with meaning occur. The operation *delete attribute–value pair* cannot cause conflicts and can be performed automatically. There was no example of *delete attribute–value pair* in the test set.

6.4.1.2 Supported Changes

The most frequently supported change was *add concept*. There were 44 cases of *add concept* that could not be automated, but that were supported. There also were 3 cases of *add attribute* that were supported. Support for additions takes two forms: (1) identification of another concept or attribute that has the same name, or (2) display of other concepts and attributes added during the change interval that may have the same meaning.

Whenever a concept is identified that has the same name as another concept, but does not fit a requirement for automatic merging, the system alerts the user to the possibility that the concept being added from the shared vocabulary might be the same concept as a concept in the local vocabulary. *Ehrlichiosis*, *Queensland tick typhus*, *rickettsialpox*, and *murine typhus* were such concepts. Figure 6.7 asks the user if he would like to merge the local concept into the shared concept for *Queensland tick typhus*.

Every time the synchronizer encounters *add concept* or *add attribute* in the shared log, the local maintainer has the opportunity to review the concepts or attributes that were added to the local vocabulary during the previous change interval. When the system cannot identify a concept or attribute in the local vocabulary that is equivalent to the added concept or attribute from the shared vocabulary, it is up to the maintainer to do so. For example, the attribute *vector* was equivalent to *transmitted-by*, the attribute *etiology* was equivalent to *has-etiology*, and the concept *Orientia tsutsugamushi* was equivalent to *Rickettsia tsutsugamushi*. Figure 6.8 shows how the user selects a concept to merge with

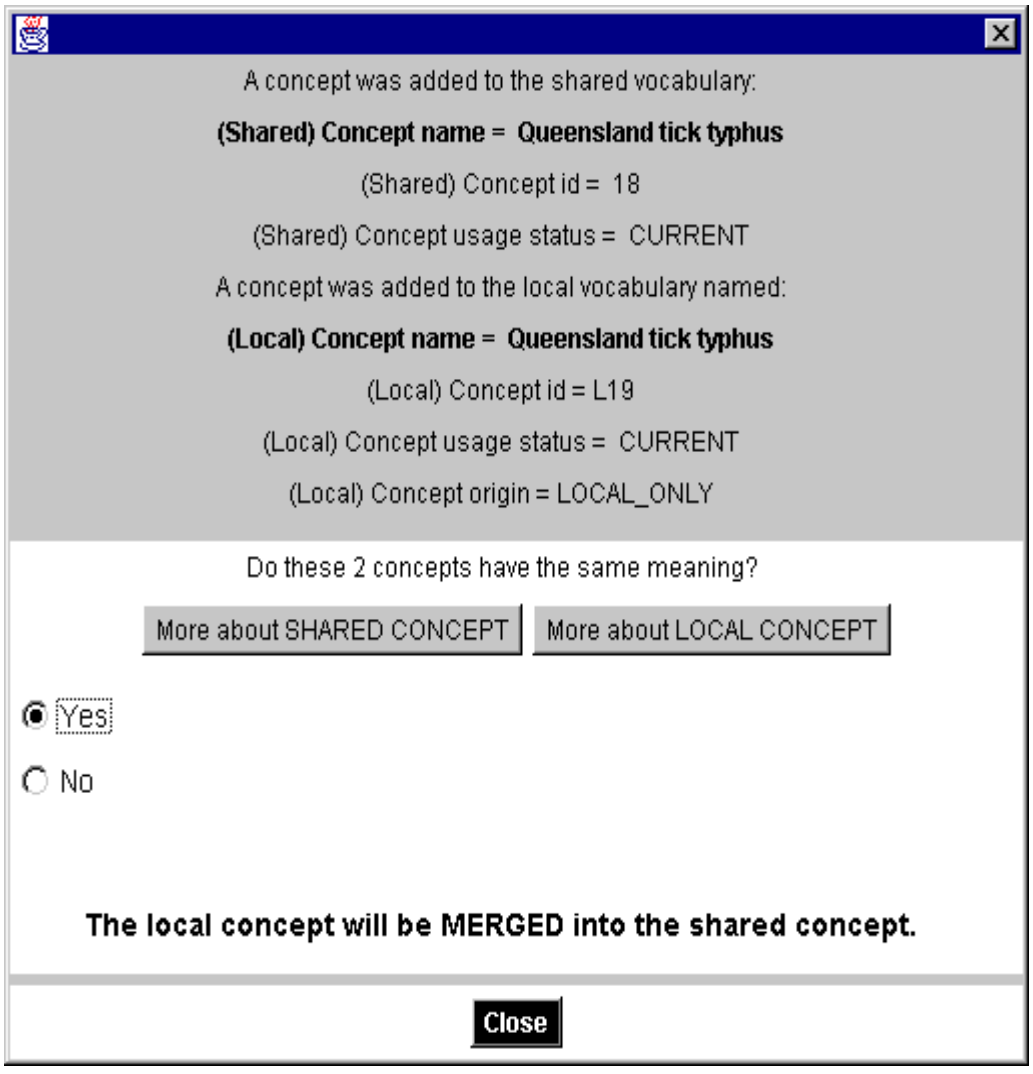

Figure 6.7. Panel that alerts the user to the fact that two concepts with the same name exist. The user may click on the buttons for more information about the shared and local concepts to see if they have the same meaning. If the user confirms that the two concepts have the same meaning, the local concept will be merged into the shared concept.

Figure 6.8. Panel that displays added local concepts, among which the user searches for a local concept that is equivalent to the shared concept *Orientia tsutsugamushi*.

Orientia tsutsugamushi. The user first clicks a button labeled "Look for EQUIVALENT local concept." Then a dialog box appears with a list of local concepts for the maintainer to review. Algorithms that look for partial string matches or for exact matches of tokens within phrases could be implemented to help the user to identify concepts that might be the same.

Two types of conflicts can occur with *add parent* and *add child*. First, if a cycle will exist after the addition, there is a conflict. Second, if the rules about a concept's attribute–value pairs and inherited attribute−value pairs will be violated, there is a conflict. If the system detects that there will be conflict after the change, the system asks the user to resolve the conflict before continuing. There were no examples of such situations in the test set. Similarly, *add attribute–value pair* and *replace attribute value* may result in a conflict between a concept's attribute–value pairs and its inherited attribute−value pairs. If so, the system asks the user to resolve the conflict before continuing. Again, there were no examples in this test set.

6.4.2 Synchronization Report

The system produces a synchronization report so that the local maintainer may review a readable version of the changes that took place during the synchronization session. Figure 6.9 shows a portion of the synchronization report produced in this experiment. Appendix K shows the entire report. The report can help the maintainer to select portions of the vocabulary to review. The maintainer uses the local-vocabulary browser to display and review selected portions of the modified hierarchy, and verifies that the changes made are appropriate for the local site.

6.5 Analysis of Results

In this section, I consider the questions that I raised at the beginning of this chapter: (1) Were the synchronization criteria fulfilled? (2) Was the model effective for this test set? (3) Did automation of certain tasks and support of other tasks facilitate synchronization?

6.5.1 Were the Synchronization Criteria Fulfilled?

A comparison of the final version of the shared vocabulary, SV-1, and the final version of the local vocabulary, LV-1-synch, demonstrates that the three criteria for synchronization were met. Those criteria are (1) preservation of concept existence and

```
REPORT OF CHANGES PERFORMED DURING SYNCHRONIZATION

DATE OF REPORT: Mon Dec 13 16:37:04 PST 1999
AUTHOR OF SYNCHRONIZATION: Diane E. Oliver

FILE NAMES:

Local vocabulary (input): E:\Development\Demo\Save Items
Local\LocalVocabTimeI
Shared vocabulary (input): E:\Development\Demo\Save
Items\SharedVocabTimeF
Local log (input): E:\Development\Demo\Save Items Local\LocalLogAToI
Shared log (input): E:\Development\Demo\Save Items\SharedLogAToF
Synchronized local vocabulary (output):
E:\Development\Demo\LocalVocabTime1Synch
Synchronization local log (output): E:\Development\Demo\LocalLogSynch
This report: E:\Development\Demo\AToFSynchReport.txt

LIST OF CHANGES:

Add concept: "rickettsial disease" with parent = "disease"
Merge 2 concepts into one of the 2 concepts: "rickettsial disease" into
"rickettsial disease"
Add concept: "mite-borne spotted fever" with parent = "rickettsial
disease"
Add concept: "tick-borne spotted fever" with parent = "rickettsial
disease"
Add concept: "flea-borne rickettsial disease" with parent =
"rickettsial disease"
Add concept: "louse-borne rickettsial disease" with parent =
"rickettsial disease"
Add concept: "ehrlichiosis" with parent = "rickettsial disease"
Merge 2 concepts into one of the 2 concepts: "ehrlichiosis" into
"ehrlichiosis"
Add concept: "Q fever" with parent = "rickettsial disease"
Merge 2 concepts into one of the 2 concepts: "Q fever" into "Q fever"
(Concept: scrub typhus) Add parent: "rickettsial disease"
(Concept: scrub typhus) Remove parent: "tropical disease of disputed
nature or minor importance"
(Concept: Rocky Mountain spotted fever) Add parent: "tick-borne spotted
fever"
Add concept: "Mediterranean spotted fever" with parent = "tick-borne
spotted fever"
Merge 2 concepts into one of the 2 concepts: "boutonneuse fever" into
"Mediterranean spotted fever"
Add concept: "African tick-bite fever" with parent = "tick-borne
spotted fever"
Add concept: "Japanese spotted fever" with parent = "tick-borne spotted
fever"
Add concept: "Queensland tick typhus" with parent = "tick-borne spotted
fever"
Merge 2 concepts into one of the 2 concepts: "Queensland tick typhus"
into "Queensland tick typhus"
```

Figure 6.9. Portion of the synchronization report.

identity; (2) preservation of subsumption relationships; and (3) preservation of attribute–value pairs. The first criterion requires that all concepts from the shared vocabulary are present in the local vocabulary with the same unique identifier after synchronization. The synchronization methods guarantee that this criterion will be met, if the software is implemented correctly. Built into the synchronization-support tool are checks that compare the two versions of the vocabularies during the synchronization session. At the conclusion of the session, the software displays comparison data for the final versions of the vocabularies. Figure 6.10 shows the comparison of concepts before and after synchronization (top half of each panel).

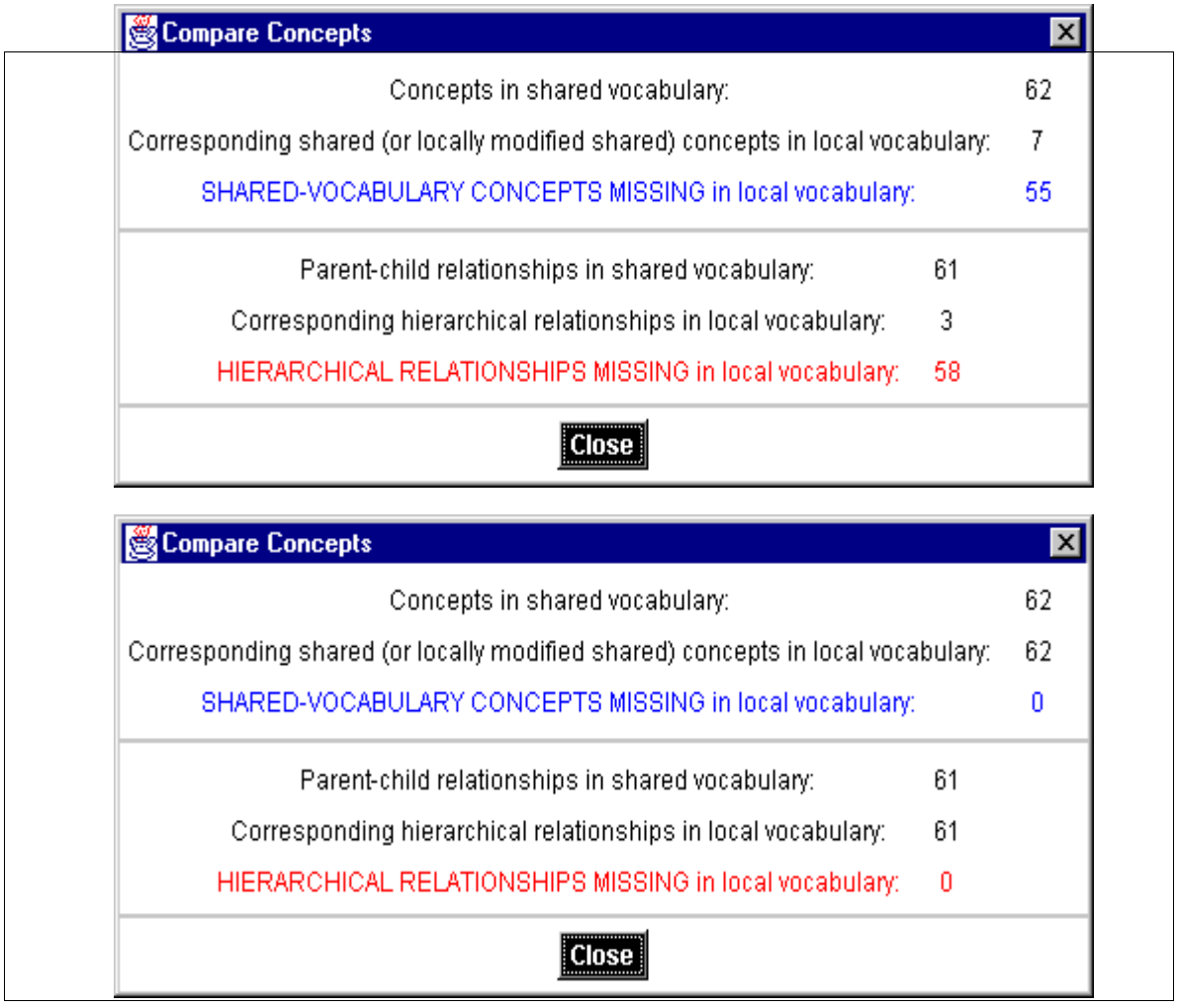

Figure 6.10. Comparison of shared and local vocabularies. Upper panel shows the comparison before synchronization begins. Lower panel shows the comparison after synchronization is completed.

The method for this comparison is straightforward. For every concept in the shared vocabulary, the system looks up the unique identifier of the shared concept in the

local vocabulary, and verifies that a concept with that identifier exists in the local vocabulary.

The second criterion requires that all hierarchical relationships in the shared vocabulary are present in the local vocabulary. Again, the synchronization methods guarantee that this criterion will be met, and a simple, straightforward algorithm performs a comparison between the shared and local vocabularies. For each parent–child pair in the shared vocabulary where parent concept *P* has unique identifier *P.id*, and child concept *Ch* has unique identifier *Ch.id*, the system locates a concept in the local vocabulary whose identifier is *P.id*, and verifies that this concept has a descendant whose identifier *Ch.id*. Figure 6.10 shows the comparison of hierarchical relationships before and after synchronization (bottom half of each panel).

The third criterion requires that all attribute–value pairs of a concept in the shared vocabulary are represented for the corresponding concept in the local vocabulary—either as attribute–value pairs or as inherited attribute–value pairs. The system currently does not track and display the fulfillment of this criterion.

6.5.2 Was the Model Effective for This Test Set?

The structural, change, and log models for this test set were generally adequate. The goal was not to represent all content in the chapters of the textbooks, but only selected content. Change operations for the test set were satisfactory. The log model was designed to support synchronization and served the purpose well.

6.5.2.1 Effectiveness of Structural Model

Features of the structural model that were clearly essential were concept unique identifier, unique concept name, parents, and synonyms. Names did change, and constant unique identifiers were necessary to maintain identity of concepts. Few medical-vocabulary experts would argue against the usefulness of these data elements. There were a few examples in the test set of multiple parents for a single concept, and experts generally agree that for anticipated uses of controlled medical vocabularies, the ability to classify a concept in more than one way is important [Chute 1999, Cimino 1998].

Including an explicit listing of children in the representation of a concept duplicates information stored in the vocabulary about the concept's parents. As a user, it is helpful to be able to view both parents and children, and from an implementation

perspective, it is easier to store children than to compute them each time the system displays them or uses them to compute descendants.

Abbreviations were useful in this test set, but it is uncertain whether abbreviations should be lumped with synonyms, or should be distinct. The answer to this question depends on the applications that will use the knowledge stored in the vocabulary. For browsing and editing, it appears to be convenient to have them displayed separately.

I did not study text definitions and UMLS codes in this example. However, for users who wish to use the vocabulary resource as a dictionary and for maintainers who need to know intended meaning of concepts to classify concepts correctly, text definitions would be useful. If the vocabulary system does not provide translation to multiple coding systems (or natural languages), a translation code to another system, such as the UMLS, that provides such translations would be useful.

Finally, the use of attribute–value pairs as a means of linking related concepts to one another was explored in this study. In CONCORDIA, attributes and their values are meant to provide non-controversial, defining information for a concept. A set of necessary and sufficient conditions is not required in the CONCORDIA model. It would have been difficult, if not impossible, to create necessary and sufficient conditions for all the concepts in this test set. However, it was possible to link diseases with etiologic organisms and with vectors of transmission, and attribute–value pairs were useful to store such knowledge.

The CONCORDIA model does not prevent a developer from entering non-definitional knowledge in attribute–value pairs. For example, the textbooks contained information about the distribution, incidence, pathology, clinical manifestations, diagnosis, prognosis, and treatment of rickettsial diseases. A controlled vocabulary is not meant to be a repository of all knowledge known about a topic, and I did not include this information in the vocabulary. However, it may be difficult for a developer to decide whether knowledge is definitional or not.

There is currently no way in CONCORDIA for values of attributes to be numeric values or strings. This restriction posed no problem for the test set, but could for other test sets.

6.5.2.2 Effectiveness of Change Model

Change operations that I used during editing of the shared and local vocabularies were *add concept, add attribute, replace concept name, add synonym, add abbreviation,*

add parent, *remove parent*, *retire concept*, *add attribute–value pair*, and *merge two concepts into one of the two concepts*. I did not use *correct concept name*, *delete synonym*, *delete abbreviation*, *replace UMLS code*, *delete attribute–value pair*, *replace attribute value*, *replace attribute name*, *replace attribute definition*, *merge two concepts into new concept*, *split concept*, *merge two attributes into one of the two attributes*, or *merge two attributes into new attribute*. In editing the local version, I did not use *preserve concept* and *preserve attribute* in the local vocabulary because no shared concepts had yet been retired. I could have used *hide concept* to make *verruga peruviana* and *heat stroke* invisible in the modified local vocabulary. These concepts are not rickettsial diseases, and, therefore, were not in the rickettsial disease chapter of *Cecil*.

6.5.2.3 Effectiveness of Log Model

The log model was designed with the goals of the synchronization-support tool in mind, and therefore, it served the requirements of synchronization well. However, if other applications have different requirements for a change log, a different log structure might be preferred. For example, if an application needs all the information that is current about a concept at the time of the change, then the contents of the entire concept object would have to be stored. However, the log design for CONCORDIA follows the principle of logs used in databases for recovery from failures [Korth 1991], in which saved log data can be used to bring the database back to a desired state. Data stored in the log makes it possible to undo or redo the stored operations. Although I have not implemented software to undo changes, a previous state of the vocabulary could be generated by application of changes in reverse. Alternatively, the current state of the vocabulary could be generated from a previous state by redoing changes specified in the log.

6.5.3 Did Automation of Certain Tasks and Support of Other Tasks Facilitate Synchronization?

When I designed the synchronization-support tool, I assumed that synchronization would be only a partially automated process rather than a fully automated process. That is, certain tasks could be automated, but other tasks would have to be supported, but not fully automated. My experience with this test set suggests this assumption is reasonable.

The easiest changes for the system to perform automatically during synchronization were name changes, including synonym and abbreviation changes. In this test set, name changes were relatively common. Although accepting a change to a concept name, synonym, or abbreviation does not take much of the user's time when the

change is not performed automatically, if there are many such changes, time saved for the user could be significant. Also, the user may be spared confusion if multiple names for a concept are similar but not exactly the same. For example, the concept named *typhus fever* in the initial shared vocabulary became *louse-borne epidemic typhus* in the modified local vocabulary, and *epidemic typhus* in the shared vocabulary. Other terms in this portion of the vocabulary were *scrub typhus*, *endemic typhus* and *murine typhus*, but those terms did not refer to the concept formerly known as *typhus fever*. A maintainer needs to keep the names straight and needs to decide which name should be current in the local vocabulary for a particular concept. A vocabulary maintainer—especially a maintainer who is not an expert in typhus fever, its variants, and the different names by which the condition is recognized—may find that the synchronization-support tool is valuable because automation of name changes reduces confusion.

The most frequent change to the shared vocabulary was *add concept*. Since the same concept could have been added to the local vocabulary that was added to the shared vocabulary and a concept must have only one unique identifier, identification of equivalent concepts is important during synchronization. This task is not easy to automate, but probably can be automated successfully in certain circumstances. In other circumstances, the system can help the user to identify matching concepts. I implemented three types of assistance with identification of equivalent concepts in the synchronization-support tool, and in this test set, use of each type was demonstrated.

In the first type of equivalent-concept identification that I implemented, the system identifies a concept in the shared vocabulary and a concept in the local vocabulary, recognizes them as the same concept, and merges them automatically. The user is spared the risk of making an error in performing the merge, or of accidentally retaining the local concept instead of the shared concept. The system always retains the shared concept with the shared unique identifier, and retires the local concept. An example in the test set of was the merge of *boutonneuse fever* and *Mediterranean spotted fever*. The shared concept, *Mediterranean spotted fever*, was retained.

In the second type of equivalent-concept identification, the system identifies a potential match, but asks for verification from the user. *Queensland tick typhus* and *ehrlichiosis* fell into this category. For *Queensland tick typhus*, two concepts by the same name existed in the shared and local vocabulary, but criteria for automatic merging were not met. The user had to verify that these two concepts were equivalent.

Third, the system does not recognize a potential match, and the user must search for a match. For example, the system did not recognize *Rickettsia tsutsugamushi* and *Orientia tsutsugamushi* as equivalent or potentially equivalent. The user had to review a list of new concepts to determine that these two concepts were the same. Similarly, the user had to recognize that the attributes *etiology* and *has-etiology* were the same, and that *vector* and *transmitted-by* were the same. Simple algorithms could suggest potential matches between concepts that share the same word within a multi-word phrase or that share the same substring. More sophisticated algorithms could consider additional clues and rank the possibilities. Algorithms of this kind were not implemented, but would be worth studying.

If the shared-vocabulary developers and the local-vocabulary developers had assigned UMLS codes to added concepts, then the synchronization-support tool could use this additional information to match concepts. This feature was not implemented.

6.6 Conclusion

This case study provided a good example of divergence of a local version of a shared vocabulary from the evolving shared vocabulary. The synchronized local vocabulary reached a state in which the synchronization criteria were fulfilled. The CONCORDIA structural model was appropriate for the representation of selected content; the CONCORDIA change model was sufficient for the performance of desired changes; and the CONCORDIA log model served the synchronization process well. The shared-vocabulary browser, shared-vocabulary editor, local-vocabulary browser, local-vocabulary editor, and synchronization-support tool provided appropriate services for supporting the process. The synchronization-support tool automated certain synchronization tasks, and supported other tasks that required input from the user. Usability of the software tools would be enhanced by improvements in user-interface design, and additional algorithms that match potentially equivalent concepts would be useful in the synchronization-support tool.

The chapters studied in *Harrison's* and *Cecil* both contained content on rickettsial diseases, but there were differences in classification and naming of concepts. These differences suggest that different domain experts who work independently will generate different representations of the same subdomain when they develop vocabularies, as they do when they author textbooks. It is not obvious why the authors of the chapter in *Harrison's* made certain choices and the author of the chapter in *Cecil* made other choices, but differences were present.

A vocabulary developer could be influenced by three factors: (1) representation of content in resources such as textbooks and dictionaries written by other authors, (2) preferences of the developer for what content to include and exclude, and (3) choices made regarding which concept-representation constructs to employ. In this experiment, the final differences in the overall structure were the result of a series of small choices made at multiple decision points.

The synchronization process offered choices explicitly. When the synchronization-support tool ran in facilitate mode, it rapidly cycled through changes such as *add synonym* and *add abbreviation*, but stopped on many cases of *add concept*. Further work could investigate additional techniques for guiding the synchronization process and for increasing automation.

6.6.1 Strengths and Limitations of Study

In this experiment, authors of textbooks served as surrogates for vocabulary maintainers. Textbook authors are experts in their field, and the same individuals who write textbooks could be domain experts for vocabulary development. Like vocabulary developers, textbook authors generally organize topics by grouping concepts that share common features. In the textbooks used, different authors organized topics differently and used different approaches to naming the same concepts. Therefore, the use of textbooks provided a ready source of data that reflected differences in how domain experts model a domain.

To minimize the effect of individual interpretation when a domain modeler transfers content from a textbook to a vocabulary, I created domain-modeling rules (Box 6.1). These rules helped to promote consistency in the transfer process, and they provide an explanation of the process that was followed.

The subdomain of rickettsial diseases was a good choice for testing, because it contained examples of many of the constructs in the CONCORDIA structural and change models. In particular, infectious-disease concepts lend themselves well to being defined by attributes and attribute values. Semantic definitions in the Read codes are analogous to defining attribute sets in CONCORDIA. When Brown and colleagues studied the frequency with which disorders in the Read codes could be represented by semantic definitions, and grouped those disorders by specialty, they found that 80% of infectious diseases could be described with semantic definitions [Brown 1998]. Rickettsial diseases fall into the category of infectious diseases. For comparison, the percent of diseases that could be

described completely with semantic definitions ranged from 11% for psychiatric diseases to 87% for neoplastic diseases.

The long period of time between the publication date of Source 1 and the publication dates of Sources 2 and 3 (1917 for the former source, and 1996 and 1998 for the latter sources) made it likely that significant changes would have occurred during the change interval. Indeed, there were many new concepts, there were changes in concept names, there were changes in classification, and one concept became obsolete.

Despite these advantages, there were a number of limitations.

The goals of an author of a textbook differ from those of a vocabulary maintainer, and conventions for organizational structure and for terms used are not the same. The author of a textbook may classify concepts, but is not required to organize topics by subsumption relationships. Names do not need to be unique, if meaning is clear from the context.

Although domain-modeling rules encouraged consistency, they could not guarantee that domain modeling would be reproducible from one subject to another. Different individuals who generate a vocabulary from text-based content will make different choices. To study the actual behavior of domain modelers who develop and synchronize vocabularies, a user study is necessary.

Because the study was limited to a single subdomain, the study did not address problems of scalability. A comprehensive medical terminology used in an electronic medical record or a thesaurus used to support medical-information retrieval would be much larger than this test set, by several orders of magnitude. The methods presented here may need to be modified to accommodate large vocabularies.

The wide gap in time between 1917 and the 1990s made it likely that changes would occur over time due to changes in medical knowledge. However, in practice, synchronization would take place at shorter intervals. The wide gap in time might overemphasize certain change types and underestimate others. Other studies would be required to investigate the impact of the time interval.

Finally, the synchronization task would be performed repeatedly throughout the life cycle of a local vocabulary. This study considered only the first cycle of change.

Despite limitations, this work offers a new perspective on management of change and local divergence of controlled medical vocabularies. It emphasizes the need for explicit representation of structure and change so that separate applications can share

change data. This study explores ways for medical content to be mapped to the CONCORDIA structural and change models, and looks at the kinds of decisions that arise during modeling, editing, and synchronization. In Chapter 7, I summarize this work and discuss how it might lead to future investigation.

7 Discussion and Future Work

The first task in this research was to review controlled medical vocabularies that are in use; because they are in use, these vocabularies force developers to deal with maintenance. Often, developers release a new version of a vocabulary, without reporting changes; if they do report changes, they rarely present the details of those changes in a consistently structured, machine-readable format.

The second task was to review frame-based knowledge-representation systems, including description logics. These systems have formal syntax and semantics, and well-defined specifications for changes. However, work on knowledge-representation systems does not emphasize representation of changes made, and systems often lack special constructs for managing constant unique meaningless identifiers, changeable unique names, synonyms, and abbreviations. Certain features that frame systems do offer probably are not needed in medical vocabularies, such as the ability to represent individuals, and the option to specify any integer value for cardinality of an attribute.

From this review of existing systems emerged CONCORDIA, a model whose goals are consistent with the requirements of rapidly changing controlled medical vocabularies, but whose features also include characteristics of frame-based knowledge-representation systems. My third task was to produce a specification of the CONCORDIA model, taking into consideration lessons learned from existing systems.

The fourth task was to demonstrate the utility of such a model by implementing a system that is based on the shared-vocabulary and local-vocabulary models of CONCORDIA. That implementation is Concept Manager—a set of browsers, editors, and a synchronization-support tool.

The fifth task was to create a vocabulary test set based on content from textbooks of medicine. The test set mimics the divergence that might occur if different developers were to modify a shared version of a vocabulary in different ways. Using the software tools of Concept Manager, I demonstrated synchronization of the local version of the test-set vocabulary with the shared version.

The CONCORDIA model provides a framework within which developers and users can manage, understand, and communicate about change. The synchronization methods that I developed are based on the model and offer one solution to the problem of local divergence.

The relevance of this work extends beyond controlled medical vocabularies. Problems of change management and local divergence occur in other types of knowledge-based systems. For example, representation of and management of change in computer-based clinical guidelines pose challenges that are similar to and perhaps more complex than those presented here.

In this chapter, first, I compare CONCORDIA with existing models. Next, I analyze the design of the CONCORDIA model and synchronization services that I implemented. I summarize the evaluation and describe further evaluation to be done. I discuss how problems and solutions that pertain to maintenance of controlled vocabularies also apply to the maintenance of computer-based clinical guidelines. I describe the contributions of this work to the scientific community, discuss unsolved problems, and conclude with a look ahead.

7.1 Comparison of CONCORDIA with Existing Models

In Chapter 2, I reviewed existing systems. Here, I compare CONCORDIA with those systems in terms of structural models, change models, and log models.

7.1.1 CONCORDIA and Existing Structural Models

A structural model can be as simple as a list of terms, or as complex as a set of knowledge-representation constructs in the most expressive description logic. If the community needs to agree on only a small set of terms, a list might suffice. Larger sets, however, require a meaningful organization structure so that a user can find terms of interest without viewing every term in the set. Handling synonyms, abbreviations, and lexical variants becomes important in a large vocabulary because these alternate names help a user to search for concepts. To ensure that a concept's meaning remains constant, developers attach a meaningless unique identifier to a set of terms that share equivalent meaning; the meaning of the concept identifier never changes, despite changes in the associated terms. The result is a concept-based vocabulary. CONCORDIA is such a vocabulary.

If concepts are organized in a directed acyclic graph, users can use the graph not only to navigate and to search for a desired concept, but also to retrieve concepts more specific or more general than a particular concept, and to determine whether one concept is a subtype or supertype of another. For these reasons, CONCORDIA has a hierarchical structure.

Although MeSH is a hierarchy, the types of relationships between parent concepts and child concepts are not specified. Although the UMLS contains hierarchies of its component vocabularies, the union of the hierarchies does not form a consistent hierarchical structure, and types of parent–child relationships are not labeled. Therefore, in MeSH and in the UMLS, it is possible for a user to navigate the hierarchy and to retrieve concepts that are more general or more specific than a particular concept, but it is not possible to determine whether one concept is truly a subtype or supertype of another. In contrast, CONCORDIA has a subsumption hierarchy, which guarantees *is-a* relationships between parent and child concepts. In this respect, CONCORDIA is like frame-based knowledge-representation languages.

If a vocabulary has a subsumption hierarchy, then it is possible to express how a child concept is similar to its parent concept by stating inherited properties that the child shares with its parent, and to differentiate the child from its parent by stating additional properties that the child has that the parent does not have. Designation of a concept's parents and properties gives the concept a structured definition. Vocabulary maintainers can use structured definitions to promote consistency of manual classification, as in the Read Codes [Schulz 1997], or to permit automatic classification, as in GALEN and SNOMED RT [Rector 1997, Spackman 1997]. A structured definition in the Read codes (called a semantic definition) is set of object–attribute–value triples [Schulz 1997]; a structured definition in SNOMED RT (called a definition) is a superconcept with a set of roles and their role values [Spackman 1997]; and a structured definition in GALEN (called a GRAIL canonical form) is a base concept and a set of criteria specified by attributes and attribute values [Rector 1997]. In GALEN, the value of an attribute may itself be a complex concept specified by a base concept and a set of criteria; therefore, the user can nest concepts to create complex, highly expressive concepts. A CONCORDIA concept has one or more parents, inherits attribute–value pairs from parents and ancestors, and may have additional attribute–value pairs of its own.

SNOMED RT and GALEN distinguish *defined* from *primitive* concepts [Mays 1996, Rector 1997]. In a structured definition, attributes or roles and their associated values

specify conditions that are true about a defined or primitive concept. As described in Chapter 2, a defined concept contains conditions that are necessary and sufficient, whereas a primitive concept does not. Concepts defined by necessary and sufficient conditions can be classified automatically. Developers of the Read Codes found that many concepts in medicine are not definable by necessary and sufficient conditions [Brown 1998]. There is no distinction in CONCORDIA between primitive and defined concepts. There is no expectation that conditions are necessary and sufficient, and automatic classification is not a feature of the implemented system. However, in CONCORDIA there are restrictions on values of attributes that encourage correct placement of concepts, and the implementation alerts the user to any violations of those restrictions.

7.1.2 CONCORDIA and Existing Change Models

Explicit descriptions of change models that include, for all change operations, operation names, inputs, constraints, and effects have not been widely distributed for existing health-care vocabulary systems. Therefore, it is difficult to compare existing systems in detail. However, we can gain insight from change files that are distributed for MeSH, SNOMED III, and the UMLS, as well as from published articles about changes to the Read Codes [Robinson 1997], the MED [Cimino 1996a], and the UMLS [Olson 1996, Suarez-Munist 1996, Tuttle 1995]. SNOMED RT change files are not yet available. I highlight similarities and differences among systems, and explain how the CONCORDIA change model compares.

Important points are whether the unique identifier for a concept can be changed, and whether the identifier of an obsolete concept can be reused. The current trend is clearly against either of these two types of changes [Chute 1998, Cimino 1998]. CONCORDIA forbids a user to change the unique identifier of a concept or attribute, or to reuse a code from an obsolete concept.

Most systems associate with each concept a unique name; in many such systems, that name can change. In GALEN, since the GRAIL canonical form serves as the unique identifier, the GRAIL canonical form cannot change without a subsequent change in the identity of the concept. There is, however, a name label in GRAIL that may be associated with the concept; this label is often more readable than the unique-identifier name, and is also unique [Rector 1997]. The label can be changed, without the identity of the concept being changed. In MeSH, when the name of a MeSH heading is replaced, a record of previous names is kept, and the old name frequently becomes an alternate name for

search purposes. However, there is no constant unique code identifier accessible to the user in MeSH; hence, a user must know the history of name changes to know which current headings relate to previous headings. In systems such as the Read codes, the MED, the UMLS, and SNOMED RT, the coded unique identifier is constant; thus, the concept name can change, without the identity of the concept being affected. CONCORDIA is like the Read codes, the MED, the UMLS, and SNOMED RT: The unique identifier cannot change, but the unique name can.

Changes in hierarchical structure can be represented by changes in codes that indicate location in a hierarchy, changes to a concept's set of parents or children, or changes in classification based on automatic inference. In MeSH, a term has tree numbers to specify where the term lies in the hierarchy. Maintainers add or delete tree numbers to change a MeSH term's location. In SNOMED III, maintainers assign a new code to a term to alter the term's location in the hierarchy. In the successor of SNOMED III, SNOMED RT, a concept's location in the hierarchy is no longer determined by that concept's code. In a manually classified system (e.g., the Read codes), changes to parent–child relationships are specified directly. In an automatically classified system (e.g., SNOMED RT and GALEN), changes to parent–child relationships can occur as a result of changes made to roles and role values or to attributes and attribute values. CONCORDIA has operations that add and remove parents and children and operations that modify attribute–value pairs. Therefore, CONCORDIA is a manually classified system like the Read codes, rather than an automatically classified system like SNOMED RT or GALEN. However, as in SNOMED RT and GALEN, a concept in CONCORDIA specifies who its parents and children are by pointers to those concepts; there is no code or tree number to indicate the concept's location.

The UMLS has operations for deletion of concept unique identifiers, term unique identifiers (which identify lexically related strings), and string unique identifiers [Olson 1996]. In contrast, concepts in the MED are *retired*, instead of *deleted* [Cimino 1998]. Concepts in the Read codes, also are not deleted: They are labeled *optional* if they are no longer useful, or *redundant* if they duplicate the meaning of other concepts [Robinson 1997]. CONCORDIA uses the operation *retire concept*. A concept that is retired has a usage status that is changed from *current* to *retired*.

In Cimino's taxonomy of change operations, *disambiguation* is a type of change; such a change results in the creation of new terms from an ambiguous term. In the Read codes, if a concept is ambiguous or obsolete, it is flagged *extinct* and is no longer

recommended. Concepts may be ambiguous because they have synonyms that were inappropriately assigned, or because they have hierarchical relationships that imply more than one meaning for the concept [Robinson 1997]. CONCORDIA's split operation is similar to disambiguation or to flagging of a concept as *extinct*. However, after a CONCORDIA concept is split, it is labeled *retired*.

The UMLS has a merge operation that causes two concepts to be combined into one. The effects of a merge can occur in MeSH, but there is no high-level merge operation recorded in the change files. For example, analysis of MeSH change tables reveals that, through a series of steps, *diabetic acidosis* and *diabetic ketosis* were removed from MeSH and *diabetic ketoacidosis* was added. In the UMLS, this change is represented as a merge. In Cimino's taxonomy, *redundancy* is an operation that corresponds in CONCORDIA to *merge two concepts into new concept* or *merge two concepts into one of the two concepts*. For consistency of naming in the CONCORDIA change model, change-operation names are verbs in the imperative form.

Because KRSS lacks constructs that represent and distinguish synonyms, abbreviations, and constant coded unique identifiers, the model underlying KRSS does not suffice for a controlled–medical-vocabulary structural model. Statements available in KRSS are largely constructors, and there are no statements for concept retirement, merges, splits, name changes, or addition or removal of synonyms or abbreviations; therefore, additional features beyond those in KRSS are needed for a medical-vocabulary change model. The goals of CONCORDIA are to fulfill the requirements of controlled medical vocabularies, and to offer the types of change operations that developers need to maintain those vocabularies. CONCORDIA lacks representation of individuals, and does not support inferencing about individuals in the way that a KRSS-compliant system does. In general, description-logic inferencing in KRSS systems is more elegant than is inferencing offered by a CONCORDIA implementation, but I designed CONCORDIA intentionally to avoid individuals, to be more flexible about change, and to support management of names.

OKBC is a formally defined protocol whose underlying model is frame based. OKBC defines methods that query for knowledge (read-only methods) and that update the knowledge (write methods). The CONCORDIA change model is analogous to OKBC methods that update knowledge. Unlike OKBC, the CONCORDIA specification does not define queries for retrieval of information. Because the underlying model of OKBC is different from the CONCORDIA model, change operations defined for the two specifications are not the same. However, the clarity and completeness with which the

OKBC developers define operations should be goals for designers of controlled medical vocabularies, and for standards developers in the health-care–terminology community.

7.1.3 CONCORDIA and Existing Log Models

Developers of ICD-9-CM, MeSH, SNOMED III, and DSM produce change records, but they follow a variety of approaches for presenting changes made. They depict changes in tables and lists, in unstructured text written in paragraph form, or in mixtures of structured and unstructured data. In the knowledge-representation community, researchers unified conceptual structures and operations in different frame-based systems when they created OKBC and KRSS, but did not create a data model for documenting completed changes.

The CONCORDIA log model is an object-oriented data model for representation of completed changes. Each type of change operation has a slightly different set of data elements that would be recorded in a log. Each change record shares with all change records certain data elements—such as change operation name, author, and date of change—but also has other data elements that are specific to the type of change. An object-oriented model is well suited to the organization of such data, and an XML Document Type Definition serves as a specification for the documentation of these data in text files.

7.1.4 CONCORDIA and Local Extensions in Existing Systems

Tuttle and colleagues recognized the problem of local modification with the earliest releases of the UMLS Metathesaurus. They knew that sites would want to add their own concepts and terms, and they realized that it would be difficult for local sites to incorporate a new release of the UMLS into their version. There are few cases of this problem reported in the literature, but in the case of the VA Lexicon (Section 1.3.1), which evolved from the UMLS, new updates to the UMLS were not incorporated.

Campbell and colleagues studied problems of local variation, but their goals were different from mine. They aimed to create a *convergent* medical terminology. In their approach, several different local terminology developers work independently and then strive to reconcile conflicts and to reach consensus. In contrast, I postulate that there will be so many local sites that the central authority will be unable to serve requests from all the sites in a timely fashion, and may choose not to honor all requests. Local groups will have needs that they do not share with the remainder of the community, and the time

delay for distribution of shared updates may be a problem for local sites. My goal, therefore, is to support carefully controlled *divergence*. The CONCORDIA model and the methods for synchronization offer a solution.

The National Health Service in the United Kingdom and users of the Read codes have dealt with the problem of local modification. The National Health Service mandates use of the Read codes, and accepts change requests from users. Clinical information systems that support the Read codes make it possible for local sites to add concepts and synonyms. Concepts and synonyms added locally initially are labeled *temporary*. When a local group submits a change request to the NHS, the NHS determines if the change will be made to the official Read codes, and sends a response to the local group. If a change request is accepted, the local site makes updates to incorporate the new Read code; if the change request is not accepted, the local site labels the concept or synonym *local*. CONCORDIA permits not only addition of concepts and synonyms, but also modification of parent–child relationships, modification of attributes and values, and hiding and preservation of concepts and attributes.

7.2 Analysis of Model and Methods

In the evaluation, I demonstrated the utility of the CONCORDIA model by implementing software that is faithful to the model and by conducting a case study of synchronization. Because the exercise is a case study, it suggests—but does not prove in the general case—the value of design and implementation choices that I made. I analyze those choices here.

7.2.1 Analysis of the CONCORDIA Design

The CONCORDIA structural model was well suited to the subdomain of the test set. For diseases and organisms in the chapters on rickettsial diseases, it was not difficult to identify a preferred name. Many synonyms were given. A few concepts had abbreviations. Since etiologic organisms that cause rickettsial diseases and vectors that transmit those diseases are generally known, I was able to structure such knowledge using attributes and attribute values for most of the disease concepts. Several diseases belonged to more than one grouping, and the option of assigning multiple parents to a CONCORDIA concept was useful. Although I did not use UMLS codes or text definitions in the evaluation, they are theoretically useful.

The change operations available in CONCORDIA were sufficient for creating and modifying the test vocabulary. I did not use all the change operations, but the changes that I needed to make were supported adequately by the model. Many of the change operations require little comment because they are unlikely to be controversial, given acceptance of the structural model. For example, *add concept, replace concept definition, add synonym, delete synonym, add abbreviation, delete abbreviation, replace UMLS code, add parent, remove parent, add child,* and *remove child* are straightforward. However, different observers might have different opinions about how to perform *retire concept, replace concept name, correct concept name, add attribute–value pair, delete attribute–value pair, replace attribute value,* and the merge and split operations. Also, the hide and preserve operations deserve brief comment.

Retire concept results in the labeling of a concept as retired, and the relinking of parents of the retired concept to children of the retired concept. An alternative approach would be to retire the entire subtree of concepts beneath the retired concept. If the concept being retired is a leaf node in the hierarchy, then the effects of the two approaches are the same. In the experiment, when *tropical disease of unknown etiology or minor importance* was retired, it was more appropriate to relink its parents and children than to retire the entire subtree. The children of the retired concept included *typhus fever* and *tsutsugamushi*, and these concepts were still valid. This example supports the choice I made for how to perform *retire concept*.

I created *replace concept name* as a compound change operation that results in the new name replacing the concept's old name, and the old name being added to the concept's synonym list. The same effect would be achieved if the user performed the two steps separately as distinct operations. It is useful to include the compound operation if it saves the user time. In the experiment, the concepts *typhus fever* and *tsutsugamushi* underwent name changes through the use of *replace concept name*. In each of these two cases, it was appropriate to include the old name in the synonym list, and it saved the maintainer an extra step. Therefore, the compound operation appears to be useful. I did not use *correct concept name* in the modification of the shared or local vocabulary, but if the user did not want to keep the old name as a synonym, it would be important to have this operation available.

Because the CONCORDIA structural model specifies a set of attribute–value pairs associated with a concept, the change operations available in the CONCORDIA change model include the addition and deletion of attribute–value pairs and the replacement of an attribute value. An alternative approach would be to permit the user to assign an attribute

to a concept without specifying a value, and later to assign a value to that attribute. Although this approach makes sense in a frame-based system, and *add slot* is a valid operation applied to a class in OKBC, I did not take this approach in CONCORDIA.

The hide and preserve operations make it possible for the local-vocabulary developers to make choices different from those made by the shared-vocabulary developers about the inclusion or exclusion of particular concepts or attributes. Hide and preserve operations simply label concepts; software developers have the responsibility to decide how to make use of these labels. For example, a browser can make hidden concepts invisible to the user by not displaying those concepts. The local-vocabulary browser in Concept Manager uses the hidden-label information in this way.

I did not use the hide operation when I modified the local vocabulary in the test set, but experimentation with the local-vocabulary editor made it clear that two additional operations would be useful: *unhide concept* and *unhide attribute*. Developers at the local site may choose to hide a concept at one point in time, but later find that users at the local site need it. The unhide operations should be added to the local extension of the CONCORDIA model.

If a local site wants to use only a small portion of the shared vocabulary, the majority of the concepts in the local vocabulary might be hidden. For example, if the shared vocabulary is a comprehensive health-care vocabulary that provides broad coverage of many specialties such as cardiology, gastroenterology, nephrology, neurology, critical care, orthopedics, ophthalmology, neurosurgery, cardiothoracic surgery, and dermatology, but the local site is a podiatry clinic, there may be many concepts that the local site does not need. The sheer size of the shared vocabulary may make it undesirable for the local site to maintain the entire system, if only a small minority of concepts is used. Labeling concepts as hidden, then, may not be the optimal solution. If the concepts of interest were all located in a subtree that had only a few links to the rest of the vocabulary, then the subtree could be maintained and distributed as a separate entity. However, if the links to the rest of the vocabulary are complex, managing the subset as a separate entity may be problematic. More research is needed in this area.

Although I did not use the split operation in this test set, a split operation appears to be conceptually useful. In my initial design of the CONCORDIA change model, I included two split operations: *split concept off* and *split concept into two new concepts*. The former operations resulted in a portion of the original concept being split off to form another concept. The original concept was retained with the original unique identifier.

However, I concluded that if, after a split, a concept loses a chunk of its meaning, the concept should no longer have the same unique identifier.

Splitting a concept off from its predecessor concept occurs in MeSH. Because MeSH is a thesaurus used for indexing the medical literature, a MeSH header forms a category for a set of related citations. If a group of citations under a single MeSH header becomes large, it makes sense to divide the group. If there is a meaningful subset of citations within the large group of citations, that subset could be grouped under a new MeSH header. The effect of this kind of change is similar to the effect of splitting a portion of a concept off, and retaining the original concept. If the MeSH header served as the unique identifier, this kind of change would violate the goal that unique identifiers have constant meaning.

The operation *split concept into two new concepts* differs from two *add child* operations, because in a split, the original concept is retired, and in two additions, the original concept is retained. A true split may be uncommon. Further study of medical subdomains and applications that depend on controlled medical vocabularies could provide evidence for or against the use of split operations.

The merge operations in CONCORDIA are likely to be useful, given that duplicates can occur. Any concept that is duplicated in the vocabulary by another concept with the same meaning, but different identifier, must be merged into its duplicate concept. The two types of merges in CONCORDIA are *merge two concepts into one of the two concepts* and *merge two concepts into new concept*. I used only the former in the test set. Further study will determine which of the two merge operations maintainers find most useful.

The operation *merge local concept into shared concept* is a synchronization operation, and currently is not part of the CONCORDIA change model, or part of the local extension of the CONCORDIA change model. In the shared vocabulary, the operation *merge two concepts into one of the two concepts* permits the merge of two concepts, which is, of course, a merge of two shared concepts. In the local vocabulary, the operation *merge two concepts into one of the two concepts* permits the merge of two concepts, but only if the concept being retired is not a shared concept, since it is illegal to retire a shared concept in the local vocabulary (from the local-vocabulary editor). Therefore, it is possible to merge a local concept into a shared concept in the local vocabulary, but the operation used is *merge two concepts into one of the two concepts*. The synchronization-support tool has an operation *merge local concept into shared*

concept, but this operation always occurs in conjunction with the processing of an *add concept* change record from the shared log.

An alternative design of the local extension of the CONCORDIA change model would restrict the operation *merge two concepts into one of the two concepts* to local concepts only, and would include the additional change operation *merge local concept into shared concept*. Since there is one more type of merge—*merge two concepts into a new concept*—there would be three merge operations in the local extension of CONCORDIA. Such design issues are subtle, but clarification and agreement on the exact names and meaning of change operations is critical for communication about change.

A problem will arise if changes in domain content imply that a concept should be split into three or more concepts, or that three or more concepts should be merged into one concept. Examples of changes that would be difficult, or impossible, to model in CONCORDIA are the changes that would have taken place in a structured vocabulary to represent the transition from the Rappaport classification of non-Hodgkin's lymphoma to the Lukes–Collins classification [Lukes 1974], or from the Rappaport classification to the Kiel classification [Lennert 1975]. These transitions involved mappings between lymphomas that were not one to one. For example, three types of lymphomas in the Rappaport classification mapped to follicular-center-cell, large, cleaved-cell lymphoma in the Lukes–Collins classification, and lymphocytic, poorly differentiated lymphoma in the Rappaport classification mapped to four types of lymphomas in the Lukes–Collins classification [Lukes 1974]. In the absence of specialized operations to support such changes, the developer of a CONCORDIA-based vocabulary probably would add the new concepts identified by Lukes and Collins, and would retire the old concepts popularized by Rappaport. The developer would have to describe the connections between the new and old concepts in text in the explanation field of the change record for *add concept*, and in the text-definition field of each new concept. Structured documentation of the changes would not be possible unless a more flexible merge operation or additional types of merge operations were available.

7.2.2 Analysis of Synchronization-Support Services

The goal of the synchronization-support tool was to automate as many services as possible, but to direct the user's attention to relevant data or decision options when human input was necessary. My work suggests that useful services offered by a synchronization-support tool are those that do the following:

1. *Assist the user with decisions that are difficult*. Finding equivalent concepts or attributes is a task that the system can achieve occasionally, but that frequently requires feedback from the user. Any time that a concept is added, the system or user must determine whether there is a concept that is equivalent to the added concept.

   In the design of the synchronizer, I allowed only two cases in which the system concludes automatically that a shared and local concept have the same meaning. One criterion was that a concept name of the first concept matches a synonym of the second concept, and vice versa. An alternative criterion was that the concept names of the two concepts are the same, and that the parents, children, and attribute–value pairs are the same. Other design choices are possible. I made a restrictive choice to minimize false positives (that is, errors that occur when the system concludes wrongly that two concepts are the same).

   If matching criteria were less restrictive, the false-negative rate (the rate at which the system concludes wrongly that two concepts are not the same) would decrease, but the false-positive rate might increase. For example, a less restrictive criterion for concluding that two concepts have the same meaning would be that the two concepts share the same concept name or a same synonym. Further research could evaluate different methods for matching concepts and for rank ordering possible matches.

2. *Perform complex tasks*. When the system processes a change that requires multiple steps and multiple links, the system reduces complexity for the user by performing steps and creating links automatically. For example, in a merge, the system assigns to the merged concept the union of parents, children, and attribute–value pairs that belonged to the two original concepts. If there are no conflicts, the system performs the tasks seamlessly. If there are conflicts, the system presents the choices available to the user to resolve those conflicts.

3. *Do not display changes that do not require a decision*. Changes that do not require a decision include *replace concept name* in the absence of unique-name conflicts, *add synonym*, *delete synonym*, *add abbreviation*, *delete abbreviation*, *replace UMLS code*, and *replace concept definition*. The system can easily make these changes automatically. If the local site is willing to

accept the changes imposed by the shared-vocabulary maintainers, then the local-vocabulary maintainer can be spared from viewing such changes.

4. *Do not display changes that have already been completed*. A concept that has already been retired, a concept whose name has already been replaced with the new name, a synonym or abbreviation that has already been added or deleted, a UMLS code that has already been replaced with the new code, or a text definition that has already been replaced with the new definition in the local vocabulary does not have to be presented the user.

5. *Reduce number of steps*. For concept changes, the system retrieves the concept in the local vocabulary that corresponds to the concept that was changed in the shared vocabulary. The concept is retrieved by unique code, instead of by unique name, because the concept's name in the local vocabulary may be different from its name in the shared vocabulary. The user does not have to take steps to locate the concept to make the change.

   In certain changes, the system can perform more than one step based on a single command from the user. For example, if a concept is retired in the shared vocabulary, the user may choose to retire it and preserve it at the same time. If a concept is added to the vocabulary, the user may add it and hide it at the same time. Otherwise, the user would have to apply the change from the shared vocabulary and go back to the local vocabulary later to preserve the retired concept or to hide the added concept.

6. *Identify cycles*. The system must identify potential cycles that will occur following *add parent*, *add child*, or a merge and must present to the user choices that will prevent the cycles. The system permits the addition or merge to take place only after the user makes an intervention to prevent the cycle.

7. *Identify attribute–value–pair conflicts*. The system must identify potential attribute–value–pair conflicts that will occur following *add parent*, *add child*, *add attribute–value pair*, or *replace attribute value*, and presents the user with choices that will prevent the conflicts. Again, the system permits the change to take place only after the user makes an intervention to prevent the conflict.

8. *Give the user a choice for more or less control over the synchronization process*. The current system addresses user preference for control of decision making by offering a choice of facilitate mode or step mode. In step mode, the user has to make a choice for every change in the shared log.

However, in facilitate mode, the system may make choices that do not reflect the user's preference. For example, in facilitate mode, the shared concept name is chosen over the local concept name if *replace concept name* is encountered in the shared log. However, the user may prefer to keep the local concept name in every case of *replace concept name* that is processed from the shared log. The system could allow the user to specify alternative preferences in advance, and carry them out in facilitate mode.

7.2.3 Further Evaluation

The study based on vocabulary content from textbooks was a first step. Additional resources of time and money would permit further evaluation of the CONCORDIA model, the synchronization process, and the software tools. User studies that involve knowledgeable subjects as vocabulary authors and that cover a wide range of clinical subdomains would improve our understanding of the generalizability of this work. Studies could be done in a laboratory setting, and then, when feasible, in a real-world setting. I describe here studies that could be done in a laboratory setting.

The CONCORDIA structural and change models would be evaluated first. Knowledgeable experts would be asked to model a variety of subdomains. The set of subdomains would cover many specialties and multiple practice settings. The experts would use medical-information resources—such as medical textbooks, medical records, journal articles, and controlled medical vocabularies—to provide assistance with content development. Theoretically, subjects could develop content without software tools, but it is generally more difficult to develop content if there is no support for search and display. The goal, however, would be to evaluate the CONCORDIA model independent of the quality of the tools. Questionnaires and interviews would be used to collect data on the subjects' views on the suitability of the model.

Following analysis of data from the modeling experiment, the CONCORDIA model would be refined and software tools would be modified accordingly. In preparation for a software usability study, the software (browsers, editors, and synchronization-support tool) would be made as robust as possible, the user interface would be enhanced, and search and display functions would be improved. Again, knowledgeable individuals would be recruited to be subjects. The usability study would be designed to determine what functions best serve the needs of users, and to determine what user-interface–design techniques are most successful. Questionnaires, interviews, and audiotapes and videotapes of the experimental sessions would be used to collect data.

The results of the usability study would be used to refine and improve the software, and finally, a study of the synchronization process could be performed. Another set of knowledgeable subjects would be recruited. One subject would play the role of the shared-vocabulary developer, and the remaining subjects would act as local-vocabulary developers.

The subject acting as the shared-vocabulary developer would produce a shared vocabulary. Each subject acting as a local-vocabulary developer would modify the shared vocabulary to produce a local vocabulary. The shared-vocabulary developer would also modify the shared vocabulary. Then, each local-vocabulary developer would synchronize his local vocabulary with the modified shared vocabulary. Given additional time and resources, a series of modifications and synchronizations would be studied. As in the usability study, investigators would use questionnaires, interviews, audiotapes, and videotapes to collect data. Goals of data analysis would be to determine what level of support users want from the synchronization-support tool, how well they understand and agree with the process, and how comfortable they are with the time required to perform synchronization.

7.3 Beyond Vocabularies: Application of Change-Management Principles to Clinical Guidelines

While interest in sharing controlled medical vocabularies has grown during the past decade, interest has also grown in sharing clinical guidelines. There are parallels between the problems of change that face the vocabulary community and similar problems that face the guideline community. In both communities, there are problems related to knowledge representation, change management, sharing, and local modification. This situation exists not only for guideline systems, but also for knowledge bases in general. Because of the relevance to clinical medicine, I discuss the relevant issues specifically with regard to clinical guidelines.

The work of this dissertation demonstrated the need for expression of and agreement on formal structural models, change models, and log models for vocabulary change management. In particular, such models are important to support local variation. Although the analogous premise for computer-based guidelines has not been studied, it is a reasonable hypothesis that these three basic types of models are essential for sharing guidelines, and in particular, for supporting local variation.

Lack of shared models results in heterogeneity in software that makes integration of information difficult. Shared models are crucial, but developing shared models is not enough; the next step is to agree on shared data formats and application programming interfaces. Data-format specifications facilitate data interchange by dictating explicitly how data are documented in files. Interfaces specify how communication takes place between client and server software entities. I concentrate on the underlying models, but clarify that the benefit of these models will be realized only when standards for interchange formats and programming interfaces follow.

Structural models of guidelines for computer-based representation have been proposed. Basic elements include eligibility criteria, actions, and decisions. PROforma [Fox 1998], GLIF [Ohno-Machado 1998], the Arden Syntax [Hripcsak 1994], and Asbru [Shahar 1998] have constructs that permit representation of such elements. The Arden syntax is a current standard that is used primarily for alerts and reminders. A universally accepted standard for complex guidelines has not yet been established. Hence, there is no universally accepted standard for a change model either.

The types of change operations that we need to support evolution of medical knowledge in guidelines will depend on how guidelines are represented, and must reflect the kinds of changes that guideline authors actually make to guidelines. In a review of an antiemetic guideline at Mayo Clinic Rochester, Loprinzi and colleagues give the history of the development of this guideline [Loprinzi 2000]. Their review provides an example of how a guideline can change. The serotonin-receptor antagonists ondansetron and granisetron were relatively new treatments for chemotherapy-induced nausea and vomiting when the guideline was developed. These agents were expensive, and were used in different ways by different oncologists at the Mayo Clinic. The diversity in usage was due both to the complexity of the literature, and to the lack of definitive answers to clinically important questions. A committee of physicians, nurses, and pharmacists convened to develop a guideline for the institution, which they released initially in September 1995.

Loprinzi and colleagues describe the initial guideline and changes made in three revisions [Loprinzi 2000]. For example, in February 1997, intravenous doses of dexamethasone were decreased from 20 mg to 10 mg because there was no evidence that 20 mg was substantially more effective than 10 mg. In July 1998, metoclopramide 40 mg orally twice a day was substituted for ondansetron 8 mg orally every 8 hours for 7 doses on the 4 days after chemotherapy. Also, the 20 mg dose of dexamethasone was resumed,

but the route was change from intravenous to oral. After the July 1998 guidelines were instituted with the switch to metoclopramide, the oncology nurses noticed a higher frequency of restlessness, agitation, drowsiness, and sleeplessness in patients. In December 1998, the use of metoclopramide in the regimen was discontinued.

These changes appear complex on the surface, but careful analysis of patterns could lead to the identification of commonly used change operations. Change operations that would support the changes described for this guideline might include *replace medication x with medication y*, *assign dosing regimen r to medication x*, *change route for medication x from intravenous to oral*, and *discontinue medication x*. In the documentation of each change, an explanation for why each change was made should be included, if available. For a formal description of the change model, the change operations would be named, their input parameters specified, their constraints carefully documented, and their effects on constructs in the guideline knowledge model expressed clearly. Shahar and colleagues took a similar approach and identified a set of formal change operators for clinical guidelines [Shahar 1995].

For the example of the Mayo Clinic antiemetic guideline, I suggest a possible change record for the discontinuation of metoclopramide. The record might include the date (*December 1998*), the name of the guideline (*use of ondansetron and granisetron to prevent chemotherapy-induced nausea and vomiting*), a reference to the particular guideline step affected (e.g., a code identifier for the particular action step in the encoded guideline), the name of the change operation (*discontinue medication x*), the name of the medication discontinued (*metoclopramide*), an explanation for why the change was made (*higher incidence of restlessness, agitation, drowsiness and sleeplessness in patients*), and the names of the authors who made the change. What the required data elements are and how they are organized would determine the log model.

For both vocabulary and guidelines, there is local pressure to modify content because of differences in opinions and priorities. In certain cases, guidelines may be accepted only if modifications can be made locally.

Text-based guidelines often have overly general or excessively vague recommendations so that the recommendations will be valid in all situations. However, encoding guidelines for computer-based guidelines frequently demands greater detail and specificity than text-based guidelines provide. One problem with encoding guidelines in greater detail is that recommendations may depend on how the organization carries out its

responsibilities. Organizations vary in terms of departmental structure, workflow processes, and roles of actors. Fridsma and colleagues have developed techniques for making guidelines site specific [Fridsma 1996]. They address organizational issues that affect implementation of a guideline.

Shahar and colleagues proposed an intention-based guideline-representation language to support detection of or explanation of significant changes to guidelines [Shahar 1995]. Their approach takes into consideration the intended goal of a guideline and recognizes that different paths may accomplish the same goal. In their work, they consider differences between the shared guideline and its execution by a particular local provider at a local institution. Thus, they recognize and address the problem of local modification of guidelines.

A formal approach to the development of structural models, change models, and log models will encourage progress in the development and distribution of computer-based guidelines just as a formal approach will benefit development and distribution of vocabularies. In both cases, changing medical knowledge forces knowledge to change over time, and local views and goals affect local preferences.

7.4 Contributions

Medical informatics is an interdisciplinary field that bridges gaps between computer science and the practice of medicine. The research conducted for this dissertation contributes both to the field of medical informatics and to the associated disciplines of computer science and clinical medicine.

7.4.1 Contributions to Medical Informatics

In describing work conducted by medical-informatics researchers, Friedman and Wyatt stated, “We study the collection, processing, and dissemination of health-care information; and we build ‘information resources’—usually consisting of computer hardware or software—to facilitate these activities” [Friedman 1997]. Browsers, editors, and synchronization-support tools are software tools that developers use to build vocabulary information resources; they facilitate directly or indirectly the collection, processing, and dissemination of health-care information. Such activities rely on the transmission of medical concepts from one processing system to another; controlled medical vocabularies are essential in this process. Health-care workers are responsible for

transmitting medical concepts from human to human, from human to computer, from computer to computer, and from computer to human. For communication to be effective, system developers need effective methods for representing concepts and for managing change in controlled medical vocabularies, since neither the language of medicine nor the practice of medicine is static.

Developers of controlled medical vocabularies in the medical-informatics community and ontology researchers in the computer-science community historically have worked independently from each other; my work brings together their work with the goal of improving vocabularies for computer-based patient-record systems. Although I have not proved the value of my methods in real-world settings in which many different users have diverse requirements, I defend my model on the basis of lessons learned from studying what other people have done.

The problem of local variation of vocabularies is recognized by researchers and developers who struggle to maintain vocabularies that serve the needs of many users. Seldom, however, has local variation been studied in a formal way. I have introduced the idea of *synchronization*, which has not been defined previously in the literature. I have shown that the problem of synchronization is a difficult problem worthy of careful thought and sophisticated methods. Organizations currently face the problem of vocabulary divergence, and the problem may become more severe as health-care institutions store increasing volumes of patient data on-line. By designing and testing one solution, I identify and address key aspects of the problem.

Friedman and Wyatt cite several reasons for performing evaluation of information systems in medicine, including (1) to encourage the use of particular systems; (2) to uncover principles of medical informatics with regard to structure, function, and impact of medical information resources; and (3) to help developers understand why certain approaches succeed or fail [Friedman 1997]. My evaluation addresses the second reason. The medical-informatics resource comprises the controlled vocabulary and the software tools that support its use. I studied structure and function by developing and analyzing use of the CONCORDIA model, and by performing a pilot test of the synchronization process. Additional studies are needed to evaluate their impact.

In the struggle to develop standards, medical-informatics researchers and developers have paid much attention to the standardization of content. In fact, most evaluations of controlled medical vocabularies assess completeness of content [Chute

1996, Henry 1997, Humphreys 1996b, Humphreys 1997]. A smaller number of analyses and evaluations concentrate on representation methods and comparisons of representation methods [Rogers 1998, Rogers 1997, Spackman 1998, Spackman 1997]. Even less attention has been paid to change. Sharability is impeded by lack of standards for vocabulary structure, inconsistent structure, and inconsistent data formats. This research brings to the forefront issues concerning change and local variation that must be solved before the user community can truly share medical-vocabulary content.

7.4.2 Contributions to Computer Science

My work makes a contribution to the computer-science subdisciplines of knowledge representation and ontological engineering. I have selectively chosen design features from KL-ONE, CLASSIC, GRAIL, OKBC, and KRSS that are relevant for health-care purposes. These systems emphasize the importance of formal approaches to managing concept hierarchies and relationships among concepts, but do not emphasize the use of synonyms and abbreviations to support search in large ontologies, do not recognize the need for the association of unique constant meaningless identifiers with meaningful unique names that may change, and do not acknowledge the importance of translation among different coding systems or natural languages. Frame-based knowledge-representation systems can handle these naming problems through the general notion of a slot, but if constructs are limited to concepts, slots, slot values, facets, and facet values, it is difficult to enforce constraints on naming conventions, and there will be no change operations that reflect the user's view of the world for manipulating unique meaningless code identifiers, unique meaningful names, synonyms, abbreviations, and translation codes.

My work also contributes to computer science because it takes a practical approach to knowledge representation and ontologies. Medical informatics is an applied discipline, and the advancement of theory and identification of new problems requires application of ideas to practical domains. By building a system that contains medical concepts from two different time periods, I have emphasized problems of maintenance that historically inspired little interest in the knowledge-representation community.

7.4.3 Contributions to the Practice of Medicine

Clinicians and patients have higher expectations for computer systems that provide clinical services and access to health-care information than they did just a few

years ago. There is increased interest on the part of hospital and clinic administrators to improve the computer infrastructure of their health-care institutions to track services rendered and resources utilized. In addition, pressure to reduce the cost of paperwork associated with billing encourages electronic transmission of claims, which may require inclusion of certain clinical data. Clinical data-entry systems, clinical databases, clinical decision-support systems, information-retrieval systems, and claims transactions all make use of controlled medical vocabularies. Goals of sharing data and knowledge among systems cannot be realized without controlled medical vocabularies.

Controlled medical vocabularies are expensive to build and maintain. We need to find ways to reuse systems built by different vocabulary developers, but it is probably not possible for one vocabulary to serve everybody's purposes. Therefore, it is important to find ways to share vocabularies, but it is important to have flexibility that permits modification for local purposes. The biggest challenge in sharing controlled vocabularies for use in patient care is their integration into the local electronic medical record. Many sites that deliver health care have legacy systems. It may be difficult to integrate legacy systems with new software, and a painful mapping process must be accomplished at each health-care site. In an ideal setting, standards for interfaces and content would exist, all systems would be compliant with standards, and the vision of plug-and-play components would be realized. Because the ideal may be difficult to achieve, the focus should be on incremental advancement. Although my work does not solve the problem of legacy systems, it offers an option for maintaining local differences if the local site is willing to devote resources initially to map the legacy vocabulary to the standard vocabulary.

A perceived benefit of electronic medical records is the ability to provide decision support and to catch errors before patients are adversely affected. Computer-based clinical guidelines offer hope that computers can deliver these promises. There is a tight coupling between guidelines and vocabulary. Communication about eligibility criteria, conditions for decisions and actions, patient data that clinicians need to obtain, and actions that clinicians need to perform requires conformance to a standard vocabulary. It will be impossible to share the logic and knowledge of clinical guidelines unless it is possible to share the names and identity of clinical concepts.

7.5 Unsolved Problems Related to this Research

Many problems arise when people build, maintain, and use vocabularies. I concentrate on the representation of change and on the management of divergence of a

local version of a vocabulary from the evolving shared vocabulary from which the local version was derived. However, for my ideas to be useful in a real-world setting, other problems must be solved.

The following topics are areas in which future work is needed to address pertinent questions.

1. *Choice of content*. What is the most useful sharable medical content to store in shared controlled medical vocabularies if the goal is to support large numbers of users for a wide variety of clinical applications? Who are the users, and what are the applications?

2. *Development by multiple authors*. What problems are encountered when multiple authors build and maintain a single vocabulary? Do differences in domain modeling occur due to differences in domain expertise or due to differences in how domain modelers prefer to use available structures in the modeling language? If multiple authors work on the same vocabulary at the same time, what is the best way to manage concurrency control? How is the vocabulary partitioned so that different authors can work on different content areas with the least impact on the work of one another?

3. *Persistent storage*. What are the best ways to maintain large numbers of concepts in persistent storage media, yet provide immediate access to all concepts and relationships when they are needed?

4. *Distributed storage*. If it is impractical to store and maintain a single vocabulary on one server, should different subdomains of the vocabulary (e.g., drugs, laboratory tests, and surgical equipment) be maintained by different expert developers on separate servers? If subsets are maintained on separate servers, how are the servers coordinated so that the vocabulary appears to be a single resource to a user?

5. *Iterative feedback about content and functionality from users*. What processes should support iterative feedback between end users of systems and vocabulary maintainers? How do users make requests and receive responses from the shared-vocabulary maintainers? How does the shared-vocabulary maintenance organization handle large volumes of requests from thousands of users?

6. *Compositional concept generation*. What techniques support compositional concept generation? If concepts are generated on the fly, how can their uniqueness be guaranteed, and how are unique identifiers assigned?

7. *User-interface design*. What principles of user-interface design for browsers, editors, and synchronization-support tools best support display, navigation, and manipulation of concepts and attributes?

8. *Scalability of concept search and display techniques*. What techniques scale well for search and display of concepts for a vocabulary of a million or more concepts?

9. *Linkage to local legacy vocabularies*. What methods would support linkages and maintenance of linkages between local legacy vocabularies and newly adopted shared vocabularies?

10. *Coordination of shared vocabulary with clinical data entry*. How are non-subsumption hierarchies of terms required for structured data entry of the clinical record coordinated with the subsumption hierarchy of concepts in the shared controlled medical vocabulary? (An example of a hierarchy of terms that would be reasonable in a physician's note—but that is not a subsumption hierarchy—is *physical examination, HEENT, fundi,* and *discs sharp*.)

11. *Updating of patient data*. How will patient data that are based on the shared vocabulary be updated to incorporate changes made to that vocabulary?

13. *Integration of vocabulary and guidelines*. If a computer-based guideline depends on a shared controlled vocabulary, how is guideline knowledge updated when the shared vocabulary changes?

14. *Boundaries of knowledge*. Where is the boundary between knowledge that belongs in a shared controlled vocabulary (maintained by a central shared-vocabulary maintenance organization), and knowledge that belongs in separate knowledge bases (maintained by other expert groups)?

15. *Recursive local modification*. How should recursive local modification be managed? That is, what problems arise if a shared vocabulary is adopted by a local site, where modifications are made and shared locally, and then a subgroup at the local site makes its own additional modifications?

7.6 A Look Ahead

My work with CONCORDIA confronts the needs for a clear vocabulary structural model, for formal change operations that make it possible to maintain these vocabularies, and for a log model. If the community moves toward a common representation of change, that move will encourage vocabulary developers to produce structured documentation of change, will help developers of merged vocabularies (such as the UMLS) to manage updates, and will facilitate the incorporation of updates from an evolving shared vocabulary into divergent local versions of that shared vocabulary.

Explicit and detailed descriptions of a change model and a log model give interested parties the opportunity to discuss the nuances of change. A shared understanding of vocabulary structure and change operations enables vocabulary developers, application developers, and end users to address the complexities of vocabulary evolution in the same way. A common approach will lead to the production of compatible software components that are developed independently. Lack of agreement on structural models, change models, and log models will make it difficult for local maintainers to update a clinical vocabulary to serve local needs, if the local site also depends on shared resources.

If local sites undertake local modification, they must control divergence carefully so that they do not lose the benefits of sharing. The challenges for developers of a shared—and possibly a standard or mandated—health-care vocabulary are to identify a vocabulary structural model that serves optimally the needs of its users, to create an explicit change model that does not conflict with the goals of local sites, and to clarify a log model. The challenge for local sites is to make choices that balance conformance and autonomy. The burden of keeping up to date a locally modified version of a shared health-care vocabulary will be theirs; techniques such as those offered by CONCORDIA will make synchronization possible.

Appendix A: Shared-Vocabulary Structural Model

The **shared-vocabulary structural model** is specified by definitions for a vocabulary concept, a shared-vocabulary root concept, a shared-vocabulary attribute, and a shared vocabulary, and by axioms that form constraints. This appendix gives those definitions and axioms, and also gives other relevant definitions.

Definition. A **shared-vocabulary concept** *C* is a structure that contains the following required elements:

1. Concept unique identifier
2. Concept name
3. Usage status
4. Set of parents that contains at least one element

Concept *C* also contains the following elements, but these elements may have values of null or the empty set:

1. Concept definition
2. Synonyms
3. Abbreviations
4. UMLS code
5. Children
6. Attribute–value pairs
7. Retired parents
8. Retired children

Concept *C* is represented by the set {*C.concept_name, C.concept_id, C.concept_definition, C.umls_code, C.synonyms, C.abbreviations, C.parents, C.children, C.avpairs, C.retired_parents, C.retired_children*}.

Definition. A **shared-vocabulary root concept** *R* is a shared-vocabulary concept whose set of parents is empty and whose set of retired parents is empty (*R.parents* = {}, *R.retired_parents* = {}).

Definition. A **shared-vocabulary attribute** *A* is a structure that contains the following required elements:

1. Attribute unique identifier
2. Attribute name
3. Attribute usage status

Attribute *A* also may contain the following element, although it is not required:

4. Attribute definition

Attribute *A* is represented by the set {*A.attribute_id*, *A.attribute_name*, *A.usage_status*, *A.attribute_definition*}.

Definition. A shared-vocabulary concept C_1 **subsumes** shared-vocabulary concept C_2 if C_2 *is a kind of* C_1, or C_2 is a C_1.

Definition. A shared-vocabulary concept P is a **parent** of shared-vocabulary concept *C* if *P* subsumes *C* and *P* belongs to *C.parents*.

Definition. A shared-vocabulary concept *Ch* is a **child** of shared-vocabulary concept *C* if *C* subsumes *Ch* and *Ch* belongs to *C.children*.

Definition. An **ancestor** of a concept is a parent of a concept or a parent of an ancestor of a concept (recursive definition). The set of **ancestors** of a concept is a collection that contains every ancestor of that concept, with duplicates removed. (Ancestors of concept *C* are represented by *C.ancestors*.)

Definition. A **descendant** of a concept is a child of a concept or a child of a descendant of a concept (recursive definition) The set of **descendants** of a concept is a collection that contains every descendant of that concept, with duplicates removed. (Descendants of concept *C* are represented by *C.descendants*.)

Definition. An **attribute–value pair** of shared-vocabulary concept *C* is a two-element set containing a shared-vocabulary attribute *A* and a shared-vocabulary concept *V* such that *C* is related to *V* by attribute *A*. An attribute–value pair is also called an **avpair** and may be represented by the set {*A*, *V*}. An attribute-value pair of *C* may be represented by AV_C. If AV_C is an attribute-value pair of *C*, then the attribute of AV_C (AV_C*.attribute*) is A_C, and the value of AV_C (AV_C*.value*) is V_C. The set of **attribute–value pairs** (or **avpairs**) of a concept

C is a collection that contains every attribute–value pair of *C*. (Attribute–value pairs of concept *C* are represented by *C.avpairs*.)

Definition. An **inherited attribute–value pair** of a concept is an attribute–value pair that belongs to the set of attribute–value pairs of an ancestor of that concept. For all C_i belonging to *C.ancestors* and for all attribute–value pairs AV_i belonging to C_i*.avpairs*, AV_i is an inherited attribute–value pair of *C*. The set of **inherited attribute–value pairs** of concept *C* is a collection that contains every attribute–value pair of every ancestor, with duplicates removed. (The inherited attribute–value pairs of concept *C* may be represented by *C.inherited_avpairs*.)

Definition. The set $\{R, C_1, C_2, C_3, \ldots C_n, A_1, A_2, \ldots A_m\}$ is a **shared vocabulary** *SV* if:

1. *R* is a shared-vocabulary root concept.
2. $C_1, C_2, C_3, \ldots C_n$ are shared-vocabulary concepts.
3. $A_1, A_2, \ldots A_m$ are shared-vocabulary attributes.
4. For all concepts C_i belonging to *SV* and for all concepts P_j belonging to C_i*.parents*, P_j belongs to *SV*.
5. For all concepts C_i belonging to *SV* and for all concepts Ch_j belonging to C_i*.children*, Ch_j belongs to *SV*.
6. For all concepts C_i belonging to *SV*, for all attribute–value pairs AV_j belonging to C_i*.avpairs*, AV_j*.attribute* is a shared-vocabulary attribute that belongs to *SV*, and AV_j*.value* is a shared-vocabulary concept that belongs to *SV*.
7. For all concepts C_i belonging to *SV*, *R* belongs to C_i*.ancestors*.
8. The axioms in Axiom Set A are true for *V* = *SV*.

Definition. A **current concept** is a concept *C* such that *C.usage_status* = "current."

Definition. A **retired concept** is a concept *C* such that *C.usage_status* = "retired."

Definition. A **cycle** is a path of *n* parent–child relationships between concepts $C_1, C_2, \ldots C_{n-1}, C_n$ in *V* where C_2 is a parent of C_1, C_3 is a parent of $C_2, \ldots, C_n$ is a parent of C_{n-1}, and C_n is a parent of C_1. (Axiom A15 forbids the presence of cycles.)

For each axiom listed below, the axiom is expressed first by a formal statement, and second, by a simpler, less formal explanation.

Axiom A1. For every concept C belonging to V, and for all concepts C_i belonging to V such that $C_i \neq C$, $C.concept\_id \neq C_i.concept\_id$.

A concept unique identifier is unique.

Axiom A2. For every concept C belonging to V, $C.concept\_id$ and for any two points in time t_1 and t_2 after C is created, $C.concept\_id$ at $t_1 = C.concept\_id$ at t_2.

A concept unique identifier never changes.

Axiom A3. For every concept C belonging to V, and for all concepts C_i belonging to V such that $C_i \neq C$, $C.usage\_status$ = "current," and $C_i.usage\_status$ = "current," $C.concept\_name \neq C_i.concept\_name$.

A concept name is unique (i.e., it does not duplicate the name of any other current concept).

Axiom A4. For every concept C belonging to V, and for all concepts C_i belonging to V such that $C_i \neq C$, $C.usage\_status$ = "current," and $C_i.usage\_status$ = "current," if $C.concept\_definition \neq \{\}$, then $C.concept\_definition \neq C_i.concept\_definition$.

A concept definition is unique (i.e., it does not duplicate the definition of any other current concept).

Axiom A5. For every concept C belonging to V, if $C.umls\_code \neq \{\}$, then $C.umls\_code$ belongs to the set of concept unique identifiers (CUIs) in the UMLS.

Every concept that has a UMLS code must have a valid UMLS code. There is no requirement that the code must be unique.

Axiom A6. Let C belong to V and $C.synonyms = \{s_1, s_2, \ldots s_n\}$. For i, j belonging to 1, 2, ... n, $s_i \neq s_j$.

A synonym is not listed more than once for a given concept.

Axiom A7. Let C belong to V and $C.synonyms = \{s_1, s_2, \ldots s_n\}$. For i belonging to 1, 2, ... n, $s_i \neq C.concept\_name$.

The concept name is not the same as any synonym name for the same concept.

Axiom A8. Let C belong to V and $C.abbreviations = \{abbrev_1, abbrev_2, ... abbrev_n\}$. For i, j belonging to 1, 2, ... n, $abbrev_i \neq abbrev_j$.

An abbreviation is not listed more than once for a given concept.

Axiom A9. For every attribute A belonging to V, and for all attributes A_i belonging to V such that $A_i \neq A$, $A.attribute\_id \neq A_i.attribute\_id$.

An attribute unique identifier is unique (i.e., it does not duplicate the unique identifier of any other attribute).

Axiom A10. For every attribute A belonging to V, $A.attribute\_id$ and for any two points in time t_1 and t_2 after A is created, $A.attribute\_id$ at $t_1 = A.attribute\_id$ at t_2.

An attribute unique identifier never changes.

Axiom A11. For every attribute A belonging to V, and for all attributes A_i belonging to V such that $A_i \neq A$, $A.usage\_status$ = "current," and $A_i.usage\_status$ = "current," $A.attribute\_name \neq A_i.attribute\_name$.

An attribute name is unique (i.e., it does not duplicate the name of any other current attribute).

Axiom A12. For every attribute A belonging to V, and for all attributes A_i belonging to V such that $A_i \neq A$ and $A_i.attribute\_usage\_status$ = "current," $A.attribute\_definition \neq A_i.attribute\_definition$.

An attribute definition is unique (i.e., it does not duplicate the definition of any other current attribute).

Axiom A13. If concept C belongs to shared vocabulary V, $C.usage\_status$ = "current" or $C.usage\_status$ = "retired."

A concept must be either current or retired.

Axiom A14. If attribute A belongs to shared vocabulary V, $A.usage\_status$ = "current" or $A.usage\_status$ = "retired."

An attribute must be either current or retired.

Axiom A15. For all concepts C_i (for i = 1, …, number of concepts in the vocabulary), there is no path of n parent–child relationships (for any integer $n > 0$) from C_i to C_{i+n} where C_{i+1} is a parent of C_i, C_{i+2} is a parent of C_{i+1}, C_{i+3} is a parent of C_{i+2}, ... C_{i+n} is a parent of C_{i+n-1} and C_{i+n} is a parent of C_i; if there were such a

pair, a cycle would exist, and cycles are not allowed in a directed acyclic graph.

No cycles are allowed.

Axiom A16. If *P* and *C* are current concepts, then *P* belongs to *C.parents* if and only if *C* belongs to *P.children*.

If P is a parent of C, then C is a child of P, and vice versa.

Axiom A17. If attribute–value pair *AV* belongs to *Ancestor.avpairs* where *Ancestor* is a concept that belongs to *C.ancestors*, then *AV* does not belong to *C.avpairs*.

An inherited avpair of concept C cannot also be an avpair of C itself.

Axiom A18. If attribute–value pair AV_i belongs to *C.avpairs*, AV_j belongs to *C.inherited_avpairs*, and $AV_i.attribute = AV_j.attribute$, then $AV_i.value$ is a descendant of $AV_j.value$.

If an avpair of a concept has the same attribute that an inherited avpair of that concept has, then that avpair has a value that is a descendant of the value of the inherited avpair.

Axiom A19. If attribute–value pair AV_i belongs to *C.avpairs*, AV_j belongs to *C.inherited_avpairs*, $AV_i.attribute = AV_j.attribute$, and $AV_i.attribute$ is an attribute that reflects anatomic location, (e.g., *has-location)*, then $AV_i.value$ may be related to $AV_j.value$ by a partitive relationship (e.g., *part-of*). (Axiom A19 modifies Axiom A18.)

If an avpair of a concept has the attribute has-location and also has an inherited avpair has-location, then the concept's avpair may have a value that is related to the value of the inherited avpair by a part-of relationship.

Appendix B: Local-Vocabulary Structural Model

The **local-vocabulary structural model** is specified by definitions for a local-vocabulary root concept, a local-vocabulary concept, a local-vocabulary attribute, and a local vocabulary, and by axioms that form constraints. This appendix gives those definitions and axioms, and also gives other relevant definitions.

Definition. A **local-vocabulary concept** *C* is a structure that contains the same required and optional elements that are required or optional in a shared-vocabulary concept. In addition, it contains a site of origin, a set of parents of the concept in the shared vocabulary (SV parents), and a set of children of the concept in the shared vocabulary (SV children).

Definition. A **local-vocabulary root concept** *R* is a structure that contains the same required and optional elements that are required and optional in a shared-vocabulary root concept. In addition, it must contain a site of origin. A local-vocabulary root concept is, by definition, also a local-vocabulary concept, but the set of parents is the empty set and the set of retired parents is the empty set.

Definition. A **local-vocabulary attribute** *A* is a structure that contains the same required and optional elements that are required or optional in a shared-vocabulary concept. In addition, it contains a site of origin.

Definition. The set $\{R, C_1, C_2, C_3, \ldots C_n, A_1, A_2, \ldots A_m\}$ is a **local vocabulary** *LV* associated with a shared vocabulary *SV* if:

1. *R* is a local-vocabulary root concept.
2. $C_1, C_2, C_3, \ldots C_n$ are local-vocabulary concepts.
3. $A_1, A_2, \ldots A_m$ are local-vocabulary attributes.
4. For all concepts C_i belonging to *LV* and for all concepts P_j belonging to $C_i.parents$, P_j belongs to *LV*.
5. For all concepts C_i belonging to *LV* and for all concepts Ch_j belonging to $C_i.children$, Ch_j belongs to *LV*.

6. For all concepts C_i belonging to *LV*, for all attribute–value pairs AV_j belonging to C_i.*avpairs*, AV_j.*attribute* is a local-vocabulary attribute that belongs to *LV*, and AV_j.*value* is a local-vocabulary concept that belongs to *LV*.

7. For all concepts C_i belonging to *LV*, *R* belongs to C_i.*ancestors*.

8. The axioms in Axiom Set A (Axioms A1 through A19) are true for *V* = *LV*, with modification of A13 and A14 by B3 and B4.

9. The axioms in Axiom Set B (Axioms B1 through B25) are true.

Definition. A **hidden concept** is a local-vocabulary concept *LC* such that *LC.usage_status* = "hidden."

Definition. A **preserved concept** is a local-vocabulary concept *LC* such that *LC.usage_status* = "preserved."

Definition. A **shared concept** is a local-vocabulary concept *LC* such that *LC.site_of_origin* = "shared."

Definition. A **locally modified shared concept** is a local-vocabulary concept *LC* such that *LC.site_of_origin* = "locally modified shared."

Definition. A **local-only concept** is a local-vocabulary concept *LC* such that *LC.site_of_origin* = "local only."

AXIOMS FOR CONCEPTS (Axioms B1 through B10)

Axiom B1. For all concepts *LC* belonging to local vocabulary *LV*, *LC.usage_status* = "current," "retired," "hidden," or "preserved."

A local-vocabulary concept is current, retired, hidden, or preserved.

Axiom B2. For all concepts *LC* belonging to local vocabulary *LV*, *LC.site_of_origin* = "shared," "locally_modified_shared," or "local_only."

A local-vocabulary concept is shared, locally modified shared, or local only.

Axiom B3. If concept *C* is a local-vocabulary concept, and if *C.site_of_origin* = "shared" or "locally_modified_shared", then *C.usage_status* = "current", "retired", "hidden", or "preserved."

A local-vocabulary concept that is shared or locally modified shared may be current, retired, hidden, or preserved.

Axiom B4. If concept *C* is a local-vocabulary concept, and if *C.site_of_origin* = "local_only", then *C.usage_status* = "current" or "retired."

A local-vocabulary concept that is local only may be current or retired.

Axiom B5. If concept *LC* is a local-vocabulary concept such that *LC.site_of_origin* = "shared" or "locally_modified_shared," and if *LC.usage_status* = "current", then there exists concept *SC* in the shared vocabulary such that *LC.concept_id = SC.concept_id* and *SC.usage_status* = "current."

A shared or locally modified shared concept that is current in the local vocabulary is also current in the shared vocabulary.

Axiom B6. If concept *LC* is a local-vocabulary concept such that *LC.site_of_origin* = "shared" or "locally_modified_shared," and if *LC.usage_status* = "retired", then there exists concept *SC* in the shared vocabulary such that *LC.concept_id = SC.concept_id* and *SC.usage_status* = "retired."

A shared or locally modified shared concept that is retired in the local vocabulary is also retired in the shared vocabulary.

Axiom B7. If concept *LC* is a local-vocabulary concept such that *LC.site_of_origin* = "shared" or "locally_modified_shared," and if *LC.usage_status* = "hidden", then there exists concept *SC* in the shared vocabulary such that *LC.concept_id = SC.concept_id* and *SC.usage_status* = "current."

A concept that is hidden in the local vocabulary is current in the shared vocabulary.

Axiom B8. If concept *LC* is a local-vocabulary concept such that *LC.site_of_origin* = "shared" or "locally_modified_shared," and if *LC.usage_status* = "preserved", then there exists concept *SC* in the shared vocabulary such that *LC.concept_id = SC.concept_id* and *SC.usage_status* = "retired."

A concept that is preserved in the local vocabulary is retired in the shared vocabulary.

Axiom B9. If concept *LC* belongs to local vocabulary *LV*, concept *SC* belongs to shared vocabulary *SV*, and *LC.concept_id = SC.concept_id*, then *LC.site_of_origin* = "shared" or "locally_modified_shared."

If a concept belongs to both the local vocabulary and the shared vocabulary, then that concept is shared or locally modified shared.

Axiom B10. If concept *LC* belongs to local vocabulary *LV* and there is no concept *SC* that belongs to shared vocabulary *SV* such that *LC.concept_id* = *SC.concept_id*, then *LC.site_of_origin* = "local_only."

If a concept belongs to the local vocabulary, but not to the shared vocabulary, then that concept is local only.

AXIOMS FOR ATTRIBUTES (Axioms B11 through B20)

Axiom B11. For all attributes *LA* belonging to local vocabulary *LV*, *LA.usage_status* = "current," "retired," "hidden," or "preserved."

A local-vocabulary attribute is current, retired, hidden, or preserved.

Axiom B12. For all attributes *LA* belonging to local vocabulary *LV*, *LA.site_of_origin* = "shared," "locally_modified_shared", or "local_only."

A local-vocabulary attribute is shared, locally modified shared, or local only.

Axiom B13. If attribute *A* is a local-vocabulary attribute, and *A.site_of_origin* = "shared" or "locally_modified"shared", then *A.usage_status* = "current", "retired", "hidden", or "preserved."

A local-vocabulary attribute that is shared or locally modified shared may be current, retired, hidden, or preserved.

Axiom B14. If attribute *LA* is a local-vocabulary attribute, and *LA.site_of_origin* = "local_only", then *LA.usage_status* = "current" or "retired."

A local-vocabulary attribute that is local only may be current or retired.

Axiom B15. If attribute *LA* is a local-vocabulary attribute such that *LA.site_of_origin* = "shared" or "locally_modified"shared," and if *LA.usage_status* = "current", then there exists attribute *SA* in the shared vocabulary such that *LA.attribute_id = SA.attribute_id* and *SA.usage_status* = "current."

A shared or locally modified shared attribute that is current in the local vocabulary is also current in the shared vocabulary.

Axiom B16. If attribute *LA* is a local-vocabulary attribute such that *LA.site_of_origin* = "shared" or "locally_modified"shared", and if *LA.usage_status* = "retired",

then there exists attribute *SA* in the shared vocabulary such that *LA.attribute_id* = *SA.attribute_id* and *SA.usage_status* = "retired."

A shared or locally modified shared attribute that is retired in the local vocabulary is also retired in the shared vocabulary.

Axiom B17. If attribute *LA* is a local-vocabulary attribute such that *LA.site_of_origin* = "shared" or "locally_modified"shared," and if *LA.usage_status* = "hidden", then there exists attribute *SA* in the shared vocabulary such that *LA.attribute_id* = *SA.attribute_id* and *SA.usage_status* = "current."

An attribute that is hidden in the local vocabulary is current in the shared vocabulary.

Axiom B18. If attribute *LA* is a local-vocabulary attribute such that *LA.site_of_origin* = "shared" or "locally_modified"shared," and if *LA.usage_status* = "preserved", then there exists attribute *SA* in the shared vocabulary such that *LA.attribute_id* = *SA.attribute_id* and *SA.usage_status* = "retired."

An attribute that is preserved in the local vocabulary is retired in the shared vocabulary.

Axiom B19. If attribute *LA* is a local-vocabulary attribute, attribute *SA* is a shared-vocabulary attribute, and *LA.attribute_id* = *SA.attribute_id*, then *LA.site_of_origin* = "shared" or "locally_modified_shared."

If an attribute belongs to both the local vocabulary and the shared vocabulary, then that attribute is shared or locally modified shared.

Axiom B20. If attribute *LA* is a local-vocabulary attribute and there is no attribute *SA* that belongs to shared vocabulary *SV* such that *LA.attribute_id* = *SA.attribute_id*, then *LA.site_of_origin* = "local_only."

If an attribute belongs to the local vocabulary, but not to the shared vocabulary, then that attribute is local only.

NAMESPACE AXIOMS (Axioms B21 through B25)

Axiom B21. There exist two distinct namespaces: a local namespace and a shared namespace. The intersection of the set of strings in the local namespace and the set of strings in the shared namespace is null.

There is no overlap between the local namespace and the shared namespace.

Axiom B22. For all concepts *SC* belonging to *SV*, *SC.concept_id* belongs to the shared namespace.

Concepts in the shared vocabulary have identifiers in the shared namespace.

Axiom B23. For all attributes *SA* belonging to *SV*, *SA.attribute_id* belongs to the shared namespace.

Attributes in the shared vocabulary have identifiers in the shared namespace.

Axiom B24. For all concepts *LC* belonging to *LV* where *LC.site_of_origin* = "shared" or "locally_modified_shared", *LC.attribute_id* belongs to the shared namespace; for all concepts *LC* belonging to *LV* where *LC.site_of_origin* = "local only," *LC.concept_id* belongs to the local namespace.

Concepts in the local vocabulary have identifiers in the shared namespace if they are shared or locally modified shared, and in the local namespace if they are local only.

Axiom B25. For all attributes *LA* belonging to *LV* where *LA.site_of_origin* = "shared" or "locally_modified_shared", *LA.attribute_id* belongs to the shared namespace; for all concepts *LA* belonging to *LV* where *LA.site_of_origin* = "local only," *LA,attribute_id* belongs to the local namespace.

Attributes in the local vocabulary have identifiers in the shared namespace if they are shared or locally modified shared, and in the local namespace if they are local only.

Appendix C: Shared-Vocabulary Change Model

The **shared-vocabulary change model** is specified by definitions for change operations and two requirements. The change operations may be performed on a shared vocabulary *SV*, one change at a time. Formal specifications for the change operations are given below.

In the descriptions that follow, SV_i represents the state of the shared vocabulary *SV* at time *i*, and SV_{i+1} represents the state of the shared vocabulary at time *i+1*. A data element followed by SV_i or SV_{i+1} in parentheses refers to that data element in the shared vocabulary at time *i*, or at time *i+1*, respectively.

Definition. A change operation is a **valid shared-vocabulary change operation** if it is one of the following:

1. Add concept
2. Retire concept
3. Merge two concepts into one of the two concepts
4. Merge two concepts into new concept
5. Split concept into two new concepts
6. Add attribute
7. Retire attribute
8. Merge two attributes into one of the two attributes
9. Merge two attributes into new attribute
10. Replace concept name
11. Correct concept name
12. Replace concept definition
13. Replace UMLS code
14. Add synonym
15. Delete synonym
16. Add abbreviation

17. Delete abbreviation
18. Add parent
19. Remove parent
20. Add child
21. Remove child
22. Add attribute-value pair
23. Delete attribute-value pair
24. Replace attribute value
25. Replace attribute name
26. Replace attribute definition

Definition. A **vocabulary change operation** is a change that affects the existence of one or more concepts or attributes in the vocabulary. By definition, such a change operation is one of change operations 1 through 9 above.

Definition. A **concept change operation** is a change that affects one or more elements in a shared-vocabulary concept. By definition, such a change operation is one of change operations 10 through 24 above.

Definition. An **attribute change operation** is a change that affects one or more elements in a shared-vocabulary attribute. By definition, such a change operation is change operation 25 or 26 above.

Requirement C1. If SV_i fulfills the requirements of a shared vocabulary, and *change_op* is a valid shared-vocabulary change operation, which causes the shared vocabulary to be transformed from an initial state, SV_i, to a subsequent state, SV_{i+1}, then SV_{i+1} also fulfills the requirements of a shared vocabulary.

$$SV_i \xrightarrow{change\_op} SV_{i+1}$$

Requirement C2. If SV_i fulfills the requirements of a shared vocabulary, $change\_op_1$, $change\_op_2$, ...$change\_op_n$ are valid shared vocabulary change operations, and SV_{i+1}, SV_{i+2}, ...SV_{i+n} are the states of the shared vocabulary after each change operation respectively, then SV_{i+n} also fulfills the requirements of a shared vocabulary.

$$SV_i \xrightarrow{change\_op_1} SV_{i+1} \xrightarrow{change\_op_2} SV_{i+2} \ \ldots \xrightarrow{change\_op_n} SV_{i+n}$$

Notation used in the following descriptions of change operations include: (1) INT: intersection, and (2) U: union.

Change Operation. **Add concept**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *P* is a shared-vocabulary concept

INPUT PARAMETERS:

1. String *new_concept_name*
2. Concept *P*

CONSTRAINTS:

1. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current," *C.concept_name* ≠ *new_concept_name*
2. *P* belongs to *SV*
3. *P.usage_status* ≠ "retired"

EFFECTS:

1. New concept *C* is created
2. *C.concept_id* = next integer concept identifier assigned by system
3. *C.concept_name* = *new_concept_name*
4. *C.usage_status* = "current"
5. *C.parents*(SV_{i+1}) = {P}

6. $SV_{i+1} = SV_i \cup \{C\}$

7. $P.children(SV_{i+1}) = P.children(SV_i) \cup \{C\}$

Change Operation. **Retire concept**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*

CONSTRAINTS:

1. *C* belongs to *V*

2. *C.usage_status* ≠ "retired"

3. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current" and for all attribute–value pairs *AV* belonging to *C.avpairs*, *AV.value* ≠ *C*

EFFECTS:

1. *C.usage_status* = "retired"

2. For all concepts *P* belonging to *C.parents*,

   $P.retired\_children(SV_{i+1}) = P.retired\_children(SV_i) \cup \{C\}$

3. For all concepts *Ch* belonging to *C.children*,

   $Ch.retired\_parents(SV_{i+1}) = Ch.retired\_parents(SV_i) \cup \{C\}$

4. For all concepts *P* belonging to *C.parents*,

   $P.children(SV_{i+1}) = P.children(SV_i)$

   $- \{C\}$

   $\cup$ *C.children*

5. For all concepts *Ch* belonging to *C.children*,

   $Ch.parents(SV_{i+1}) = Ch.parents(SV_i)$

   $- \{C\}$

U *C.parents*

Change Operation. **Merge two concepts into one of the two concepts**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C1* is a shared-vocabulary concept
3. *C2* is a shared-vocabulary concept
2. *C_to_keep* is a shared-vocabulary concept
3. *C_to_retire* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C1*
2. Concept *C2*
3. Concept *C_to_keep*
4. Concept *C_to_retire*

CONSTRAINTS:

1. *C1* belongs to *SV*
2. *C2* belongs to *SV*
3. *C1* is not the root of *SV*
4. *C2* is not the root of *SV*
5. *C1* ≠ *C2*
6. *C1.usage_status* ≠ "retired"
7. *C2.usage_status* ≠ "retired"
8. (*C1* = *C_to_keep*) <=> (*C2* = *C_to_retire*)
9. (*C2* = *C_to_keep*) <=> (*C1* = *C_to_retire*)
10. *C_to_keep* does not belong to (*C_to_retire.ancestors* – *C_to_retire.parents*)
11. *C_to_retire* does not belong to (*C_to_keep.ancestors* – *C_to_keep.parents*)

12. *Ancestors of C_to_keep must have avpairs that are the same as or more general than C_to_retire's avpairs.*

    For all concepts *A* belonging to *C_to_keep.ancestors*, for all avpairs AV_A belonging to *A.avpairs*, and for all avpairs AV_R belonging to *C_to_retire.avpairs*, if AV_A*.attribute* = AV_R*.attribute*, then AV_A*.value* = AV_R*.value* or AV_A*.value* is ancestor of AV_R*.value*.

13. *Ancestors of C_to_retire must have avpairs that are the same as or more general than C_to_keep's avpairs.*

    For all concepts A_i belonging to *C_to_retire.ancestors*, for all avpairs AV_A belonging to *A.avpairs*, and for all avpairs AV_K belonging to *C_to_keep.avpairs*, if AV_A*.attribute* = AV_K*.attribute*, then AV_A*.value* = AV_K*.value* or AV_A*.value* is ancestor of AV_K*.value*.

14. *Descendants of C_to_keep must have avpairs that are more specific than C_to_retire's avpairs.*

    For all concepts *D* belonging to *C_to_keep.descendants*, for all avpairs AV_D belonging to *D.avpairs*, and for all avpairs AV_R belonging to *C_to_retire.avpairs*, if AV_D*.attribute* = AV_R*.attribute*, then AV_D*.value* = AV_R*.value* or AV_D*.value* is descendant of AV_R*.value*.

15. *Descendants of C_to_retire must have avpairs that are more specific than C_to_keep's avpairs.*

    For all concepts *D* belonging to *C_to_retire.descendants*, for all avpairs AV_D belonging to *D.avpairs*, and for all avpairs AV_K belonging to *C_to_keep.avpairs*, if AV_D*.attribute* = AV_K*.attribute*, then AV_D*.value* = AV_K*.value* or AV_D*.value* is descendant of AV_K*.value*.

EFFECTS:

1. Parents

   a. If *C_to_keep* belongs to *C_to_retire.parents* (i.e., *C_to_keep is a parent of C_to_retire*)

   C_to_keep.parents = *C_to_keep.parents* U *C_to_retire.parents*

   - (*C_to_keep.ancestors* INT *C_to_retire.parents*)

   - *C_to_keep*}

b. Alternatively, if *C_to_retire* belongs to *C_to_keep.parents* (i.e., *C_to_retire is a parent of C_to_keep*)

C_to_keep.parents = *C_to_keep.paren*ts U *C_to_retire.parents*

- (*C_to_keep.ancestors* INT *C_to_retire.parents*)

- *C_to_retire*

c. Alternatively, if *C_to_keep* does not belong to *C_to_retire.parents* and *C_to_retire* does not belong to *C_to_keep.parents* (i.e., *there is no parent–child relationship between C_to_keep and C_to_retire*)

C_to_keep.parents = *C_to_keep.parents* U *C_to_retire.parents*

- (*C_to_keep.ancestors* INT *C_to_retire.parents*)

2. Children

a. If *C_to_keep* belongs to *C_to_retire.parents* (i.e., *C_to_keep is a parent of C_to_retire*),

C_to_keep.children = *C_to_keep.children* U *C_to_retire.children*

- (*C_to_keep.descendants* INT *C_to_retire.children*)

- *C_to_retire*

b. Alternatively, if *C_to_retire* belongs to *C_to_keep.parents* (i.e., *C_to_retire is a parent of C_to_keep)*

C_to_keep.children = *C_to_keep.children* U *C_to_retire.children*

- (*C_to_keep.descendants* INT *C_to_retire.children*)

- *C_to_keep*

c. Alternatively, if *C_to_keep* does not belong to *C_to_retire.parents* and *C_to_retire* does not belong to *C_to_keep.parents* (i.e., *there is no parent–child relationship between C_to_keep and C_to_retire*)

C_to_keep.children = *C_to_keep.children* U *C_to_retire.children*

- (*C_to_keep.children* INT *C_to_retire.children*)

3. Attribute–value pairs

a. If *C_to_keep* belongs to *C_to_retire.parents* (i.e., *C_to_keep is a parent of C_to_retire*),

C_to_keep.avpairs = *C_to_keep.avpairs* U *C_to_retire.avpairs*

- {AV_R: AV_R belongs to *C_to_retire.avpairs* and there exists AV_K belonging to *C_to_keep.avpairs* where AV_R.*attribute* = AV_K.*attribute* and AV_R.*value* is ancestor of AV_K.*value*}

b. Alternatively, if *C_to_retire* belongs to *C_to_keep.parents* (i.e., *C_to_retire is a parent of C_to_keep*)

C_to_keep.avpairs = *C_to_keep.avpairs* U *C_to_retire.avpairs*

- {AV_K: AV_K belongs to *C_to_keep.avpairs* and there exists AV_R belonging to *C_to_retire.avpairs* where AV_K.*attribute* = AV_R.*attribute* and AV_K.*value* is ancestor of AV_R.*value*}

c. Alternatively, if *C_to_keep* does not belong to *C_to_retire.parents* and *C_to_retire* does not belong to *C_to_keep.parents* (i.e., *there is no parent–child relationship between C_to_keep and C_to_retire*)

C_to_keep.avpairs = *C_to_keep.avpairs* U *C_to_retire.avpairs*

- (*C_to_keep.avpairs* INT *C_to_retire.avpairs*)
- {AV_K: AV_K belongs to *C_to_keep.avpairs* and there exists AV_R belonging to *C_to_retire.avpairs* where AV_K.*attribute* = AV_R.*attribute* and AV_K.*value* is ancestor of AV_R.*value*}
- {AV_R: AV_R belongs to *C_to_retire.avpairs* and there exists AV_K belonging to *C_to_keep.avpairs* where AV_R.*attribute* = AV_K.*attribute* and AV_R.*value* is ancestor of AV_K.*value*}

4. Synonyms

C_to_keep.synonyms = *C_to_keep.synonyms*

U *C_to_retire.synonyms*

- (*C_to_keep.synonyms* INT *C_to_retire.synonyms*)
- ({*C_to_keep.concept name*} INT *C_to_retire.synonyms*)

5. Abbreviations

C_to_keep.abbreviations = *C_to_keep.abbreviations*

U *C_to_retire.abbreviations*

- (*C_to_keep.abbreviations* INT *C_to_retire.abbreviations*)

6. Usage status

C_to_retire.usage_status = “retired”

7. Children of *C_to_retire*

For each *Ch* belonging to

($C\_to\_retire.children(SV_i)$ –

($C\_to\_retire.children(SV_i)$ INT $C\_to\_keep.children(SV_i)$)),

$Ch.parents(SV_{i+1}) = Ch.parents(SV_i)$ U {*C_to_keep*} - {*C_to_retire*}

For each *Ch* belonging to

($C\_to\_retire.children(SV_i)$ INT $C\_to\_keep.children(SV_i)$),

$Ch.parents(SV_{i+1}) = Ch.parents(SV_i)$ - {*C_to_retire*}

8. Parents of *C_to_retire*

For each *P* belonging to

($C\_to\_retire.parents(SV_i)$ -

($C\_to\_retire.parents(SV_i)$ INT $C\_to\_keep.parents(SV_i)$)),

$P.children(SV_{i+1}) = P.children(SV_i)$ U {*C_to_keep*} - {*C_to_retire*}

For each *P* belonging to

($C\_to\_retire.parents(SV_i)$ INT $C\_to\_keep.parents(SV_i)$),

$P.children(SV_{i+1}) = P.children(SV_i)$ - {*C_to_retire*}

9. Descendants of *C_to_keep*

For each *D* belonging to $C\_to\_keep.descendants(SV_i)$,

$D.avpairs(SV_{i+1}) = D.avpairs(SV_i)$

- (*D.avpairs* INT
- (*C_to_retire.avpairs* U *C_to_retire.inherited_avpairs*))

10. Descendants of *C_to_retire*

For each *D* belonging to $C\_to\_retire.descendants(SV_i)$,

$D.avpairs(SV_{i+1}) = D.avpairs(SV_i)$

- (*D.avpairs* INT

- (*C_to_keep.avpairs* U *C_to_keep.inherited_avpairs*))

Change Operation. **Merge two concepts into new concept**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C1* is a shared-vocabulary concept
3. *C2* is a shared-vocabulary concept
4. *P* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C1*
2. Concept *C2*
3. String *new_concept_name*
4. Concept *P*

CONSTRAINTS:

1. *C1* belongs to *SV*
2. *C2* belongs to *SV*
3. *C1* is not the root of *SV*
4. *C2* is not the root of *SV*
5. *C1* ≠ *C2*
6. *C1.usage_status* ≠ "retired"
7. *C2.usage_status* ≠ "retired"
8. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current," *C.concept_name* ≠ *new_concept_name*
9. *C1* does not belong to (*C2.ancestors – C2.parents*)
10. *C2* does not belong to (*C1.ancestors– C1.parents*)
11. *P* belongs to *C1.parents* or *P* belongs to *C2.parents*
12. *Ancestors of C1 must have avpairs that are more general than C2's avpairs*.

For all concepts *A* belonging to *C1.ancestors*, all avpairs AV_A belonging to *A.avpairs*, and all avpairs AV_2 belonging to *C2.avpairs*, if AV_A*.attribute* = AV_k*.attribute*, then AV_A*.value* = AV_2*.value* or AV_A*.value* is an ancestor of AV_2*.value*

13. *Ancestors of C2 must have avpairs that are more general than C1's avpairs.*

    For all concepts *A* belonging to *C2.ancestors*, all avpairs AV_A belonging to *A.avpairs*, and all avpairs AV_1 belonging to *C1.avpairs*, if AV_A*.attribute* = AV_1*.attribute*, then AV_A*.value* = AV_1*.value* or AV_A*.value* is an ancestor of AV_1*.value*.

14. *Descendants of C1 must have avpairs that are more specific than C2's avpairs.*

    For all concepts *D* belonging to *C1.descendants*, fall avpairs AV_D belonging to *D.avpairs*, and all avpairs AV_2 belonging to *C2.avpairs*, if AV_D*.attribute* = AV_2*.attribute*, then AV_D*.value* = AV_2*.value* or AV_D*.value* is a descendant of AV_2*.value*

15. *Descendants of C2 must have avpairs that are more specific than C1's avpairs.*

    For all concepts *D* belonging to *C2.descendants*, all avpairs AV_D belonging to *D.avpairs*, and all avpairs AV_1 belonging to *C1.avpairs*, if AV_D*.attribute* = AV_1*.attribute*, then AV_D*.value* = AV_1*.value* or AV_D*.value* is a descendant of AV_1*.value*

EFFECTS:

1. Create concept *C_new* and add *C_new* to vocabulary

    a. Concept *C_new* is created

    b. *C_new.concept_id* = next integer concept identifier assigned by system

    c. *C_new.concept_name* = *new_concept_name*

    d. *C_new.usage_status* = "current"

    e. *C_new.parents* = {*P*}

    f. $SV_{i+1} = SV_i \cup \{C\_new\}$

    g. $P.children(SV_{i+1}) = P.children(SV_i) \cup \{C\_new\}$

2. Parents

a. If *C1* belongs to *C2.parents* (i.e., *C1* is a parent of *C2*),

C_new.parents = *C1.parents* U *C2.parents*

- (*C1.parents* INT *C2.parents*)

- *C2*

b. Alternatively, if *C2* belongs to *C1.parents* (i.e., *C2* is a parent of *C1*),

C_new.parents = *C1.parents* U *C2.parents*

- (*C1.parents* INT *C2.parents*)

- *C1*

c. Alternatively, if *C1* does not belong to *C2.parents* and *C*2 does not belong to *C1.parents* (i.e., there is no parent–child relationship between *C_to_keep* and *C_to_retire*),

C_new.parents = *C1.parents* U *C2.parents*

- (*C1.parents* INT *C2.parents*)

3. Children

a. If *C1* belongs to *C2.parents* (i.e., *C1* is a parent of *C2*),

C_new.children = *C1.children* U *C2.children*

- (*C1.children* INT *C2.children*)

- C2

b. Alternatively, if *C2* belongs to *C1.parents* (i.e., *C2 is a parent of C1*),

C_new.children = *C1.children* U *C2.children*

- (*C1.children* INT *C2.children*)

- C1

c. Alternatively, if *C1* does not belong to *C2.parents* and *C2* does not belong to *C1.parents* (i.e., there is no parent–child relationship between *C_to_keep* and *C_to_retire*),

C_new.children = *C1.children* U *C2.children*

- (*C1.children* INT *C2.children*)

4. Attribute–value pairs

C_new.avpairs = *C1.avpairs* U *C2.avpairs*

- (*C1.avpairs* INT *C2.avpairs*)
- {AV_1: AV_1 belongs to *C1.avpairs* and there exists AV_2 belonging to *C2.avpairs* where AV_1.*attribute* = AV_2.*attribute* and AV_1.*value* is an ancestor of AV_2.*value*}
- {AV_2: AV_2 belongs to *C2.avpairs* and there exists AV_1 belonging to *C1.avpairs* where AV_2.*attribute* = AV_1.*attribute* and AV_2.*value* is an ancestor of AV_1.*value*}

5. Synonyms

C_new.synonyms = *C1.synonyms* U *C2.synonyms*

- (*C1.synonyms* INT *C2.synonyms*)
- (*C_new.concept_name* INT *C1.synonyms*)
- (*C_new.concept_name* INT *C2.synonyms*)

6. Abbreviations

C_new.abbreviations = *C1.abbreviations* U *C2.abbreviations*

- (*C1.abbreviations* INT *C2.abbreviations*)

7. Usage Status

C1.usage_status = “retired”

C2.usage_status = “retired”

8. Parents of C1

For each *P* belonging to *C1.parents* in SV_i,

P.children(SV_{i+1}) = *P.children*(SV_i) U {*C_new*} - {*C1*}

9. Parents of *C2*

For each *P* belonging to *C2.parents* in SV_i,

P.children(SV_{i+1}) = *P.children*(SV_i) U {*C_new*} - {*C2*}

10. Children of *C1*

For each *Ch* belonging to *C1.children* in SV_i,

$Ch.parents(SV_{i+1}) = Ch.parents(SV_i)$ U {*C_new*} - {*C1*}

11. Children of *C2*

    For each *Ch* belonging to *C2.children* in SV_i,

    $Ch.parents(SV_{i+1}) = Ch.parents(SV_i)$ U {*C_new*} - {*C2*}

12. Descendants of *C1*

    For each *D* belonging to *C1.descendants* in SV_i,

    $D.avpairs(SV_{i+1}) = D.avpairs(SV_i)$

    - (*D.avpairs* INT *C2.avpairs*}

    - (*D.avpairs* INT *C2.inherited_avpairs*)

13. Descendants of *C2*

    For each *D* belonging to *C2.descendants* in SV_i,

    $D.avpairs(SV_{i+1}) = D.avpairs(SV_i)$

    - (*D.avpairs* INT *C1.avpairs*)

    - (*D.avpairs* INT *C1.inherited_avpairs*)

Change Operation. **Split concept into two new concepts**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept
3. *parents_to_add_to_first_concept* is a set of concepts
4. *children_to_add_to_first_concept* is a set of concepts
5. *avpairs_to_add_to_first_concept* is a set of attribute–value pairs
6. *synonyms_to_add_to_first_concept* is a set of strings
7. *abbreviations_to_add_to_first_concept* is a set of strings
8. *parents_to_add_to_second_concept* is a set of concepts
9. *children_to_add_to_second_concept* is a set of concepts
10. *avpairs_to_add_to_second_concept* is a set of attribute–value pairs

11. *synonyms_to_add_to_second_concept* is a set of strings

12. *abbreviations_to_add_to_second_concept* is a set of strings

INPUT PARAMETERS:

1. Concept *C*

2. String *new_concept_name1*

3. String *new_concept_name2*

4. Set of concepts *parents_to_add_to_first_concept*

5. Set of concepts *children_to_add_to_first_concept*

6. Set of attribute–value pairs *avpairs_to_add_to_first_concept*

7. Set of strings *synonyms_to_add_to_first_concept*

8. Set of strings *abbreviations_to_add_to_first_concept*

9. Set of concepts *parents_to_add_to_second_concept*

10. Set of concepts *children_to_add_to_second_concept*

11. Set of attribute–value pairs *avpairs_to_add_to_second_concept*

12. Set of strings *synonyms_to_add_to_second_concept*

13. Set of strings *abbreviations_to_add_to_second_concept*

14. For all synonyms *Syn* belonging to (*C.synonyms* - *synonyms_to_delete*), *Syn* ≠ *new_concept_name*

CONSTRAINTS:

1. *C* belongs to *SV*

2. *C.usage_status* ≠ "retired"

3. *P1* belongs to *SV* and belongs to *parents_to_add_to_first_concept*

4. *P1.usage_status* ≠ "retired"

5. *P2* belongs to *SV* and belongs to *children_to_add_to_second_concept*

6. *P2.usage_status* ≠ "retired"

7. *new_concept_name1* ≠ *new_concept_name2*

8. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current," *C.concept_name* ≠ *new_concept_name1* and *C.concept_name* ≠ *new_concept_name2*
9. *parents_to_add_to_first_concept* is a subset of *C.parents*
10. *children_to_add_to_first_concept* is a subset of *C.children*
11. *avpairs_to_add_to_first_concept* is a subset of *C.avpairs*
12. *synonyms_to_add_to_first_concept* is a subset of *C.synonyms*
13. *abbreviations_to_add_to_first_concept* is a subset of *C.abbreviations*
14. *parents_to_add_to_second_concept* is a subset of *C.parents*
15. *children_to_add_to_second_concept* is a subset of *C.children*
16. *avpairs_to_add_to_second_concept* is a subset of *C.avpairs*
17. *synonyms_to_add_to_second_concept* is a subset of *C.synonyms*
18. *abbreviations_to_add_to_second_concept* is a subset of *C.abbreviations*
19. For all synonyms *Syn* belonging to (*C.synonyms* - *synonyms_to_delete*), *Syn* ≠ *new_concept_name1* and *Syn* ≠ *new_concept_name1*

EFFECTS:

1. Concept *C1_new* is created

   C1_new.concept_id = next integer concept identifier assigned by system

   C1_new.concept_name = *new_concept_name1*

   C1_new.usage_status = "current"

   C1_new.parents = {*P1*}

   $SV_{i+1} = SV_i \cup \{C\_new1\}$

   $P1.children(SV_{i+1}) = P1.children(SV_i) \cup \{C\_new1\}$

2. Concept *C2_new* is created

   C2_new.concept_id = next integer concept identifier assigned by system

   C2_new.concept_name = *new_concept_name2*

   C2_new.usage_status = "current"

   C2_new.parents = {*P2*}

$SV_{i+1} = SV_i \cup \{C\_new2\}$

$P2.children(SV_{i+1}) = P2.children(SV_i) \cup \{C\_new2\}$

3. *C_new1.parents = parents_to_add_to_first_concept*
4. C*_new1.children = children_to_add_to_first_concept*
5. *C_new1.avpairs = avpairs_to_add_to_first_concept*
6. *C_new1.synonyms = synonyms_to_add_to_first_concept*
7. *C_new1.abbreviations = abbreviations_to_add_to_first_concept*
8. *C_new2.parents = parents_to_add_to_second_concept*
9. *C_new2.children = children_to_add_to_second_concept*
10. *C_new2.avpairs = avpairs_to_add_to_second_concept*
11. *C_new2.synonyms = synonyms_to_add_to_second_concept*
12. *C_new2.abbreviations = abbreviations_to_add_to_second_concept*
13. *C.usage_status* = "retired"
14. For each P_{1st} belonging to *parents_to_add_to_first_concept*,

    $P_{1st}.children(SV_{i+1}) = P_{1st}.children(SV_i) \cup \{C\_new1\}$
15. For each Ch_{1st} belonging to *children_to_add_to_first_concept*,

    $Ch_{1st}.parents(SV_{i+1}) = Ch_{1st}.parents(SV_i) \cup \{C\_new1\}$
16. For each P_{2nd} belonging to *parents_to_add_to_second_concept*,

    $P_{2nd}.children(SV_{i+1}) = P_{2nd}.children(SV_i) \cup \{C\_new2\}$
17. For each Ch_{2nd} belonging to *children_to_add_to_second_concept*,

    $Ch_{2nd}.parents(SV_{i+1}) = Ch_{2nd}.parents(SV_i) \cup \{C\_new2\}$

Change Operation. **Add attribute**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *A* is a shared-vocabulary attribute

INPUT PARAMETER:

1. String *new_attribute_name*

CONSTRAINTS:

1. For all attributes *A* belonging to *SV* such that *A.usage_status* = "current," *A.attribute_name* ≠ *new_ attribute _name*

EFFECTS:

1. New attribute *A* is created
2. *A.attribute_id* = next integer concept identifier assigned by system
3. A.*attribute_name = new_ attribute _name*
4. *A.usage_status* = "current"
5. $SV_{i+1} = SV_i \cup \{A\}$

Change Operation. **Retire attribute**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *A* is a shared-vocabulary attribute

INPUT PARAMETER:

1. Attribute *A*

CONSTRAINTS:

1. *A.usage_status* ≠ "retired"
2. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current" and for all *AV* belonging to *C, AV.attribute* ≠ *A*

EFFECTS:

1. *A.usage_status* = "retired"

Change Operation. **Merge two attributes into one of the two attributes**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *A1* is a shared-vocabulary attribute

3. *A2* is a shared-vocabulary attribute

2. *A_to_keep* is a shared-vocabulary attribute

3. *A_to_retire* is a shared-vocabulary attribute

INPUT PARAMETERS:

1. Attribute *A1*

2. Attribute *A2*

3. Attribute *A_to_keep*

4. Attribute *A_to_retire*

CONSTRAINTS:

1. *A1* belongs to *SV*

2. *A2* belongs to *SV*

3. *A1* ≠ *A2*

4. *A1.usage_status* ≠ "retired"

5. *A2.usage_status* ≠ "retired"

6. (*A1* = *A_to_keep*) <=> (*A2* = *A_to_retire*)

7. (*A2* = *A_to_keep*) <=> (*A1* = *A_to_retire*)

EFFECTS:

1. *A_to_retire.usage_status* = "retired"

Change Operation. **Merge two attributes into new attribute**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *A1* is a shared-vocabulary attribute

3. *A2* is a shared-vocabulary attribute

INPUT PARAMETERS:

1. Attribute *A1*

2. Attribute *A2*

3. String *new_attribute_name*

CONSTRAINTS:

1. *A1* belongs to *SV*

2. *A2* belongs to *SV*

3. *A1* ≠ *A2*

4. *A1.usage_status* ≠ "retired"

5. *A2.usage_status* ≠ "retired"

6. For all attributes *A* belonging to *SV* such that *A.usage_status* = "current," *A.attribute_name* ≠ *new_attribute_name*

EFFECTS:

1. Create attribute *A_new* and add *A_new* to vocabulary

    a. Attribute *A_new* is created

    b. *A_new.attribute_id* = next integer attribute identifier assigned by system

    c. *A_new.attribute_name* = *new_attribute_name*

    d. *A_new.usage_status* = "current"

2. *A1.usage_status* = "retired"

3. *A2.usage_status* = "retired"

Change Operation. **Replace concept name**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

3. *old_name* = *C.concept_name*

INPUT PARAMETERS:

1. Concept *C*

2. String *new_name*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *C.usage_status* ≠ "retired"
3. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current," *C.concept_name* ≠ *new_name*
4. *new_name* ≠ *old_name*

EFFECTS:

1. *C.concept_name = new_name*
2. *C.synonyms*(SV_{i+1}) = *C.synonyms*(SV_i) U {*old_name*}
3. If *C.synonyms* contains *new_name*, then

   C.synonyms(SV_{i+1}) = *C.synonyms*(SV_i) - {new_name}

Change Operation. **Correct concept name**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept
3. *old_name = concept_name*

INPUT PARAMETERS:

1. Concept *C*
2. String *new_name*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *C.usage_status* ≠ "retired"
3. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current," *C.concept_name* ≠ *new_name*
4. *new_name* ≠ *old_name*

EFFECTS:

1. *C.concept_name = new_name*

2. If *C.synonyms* contains *new_name*, then

   $C.synonyms(SV_{i+1}) = C.synonyms(SV_i) - \{new\_name\}$

Change Operation. **Replace concept definition**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

3. *old_definition = C.concept_definition*

INPUT PARAMETERS:

1. Concept *C*

2. String *new_definition*

CONSTRAINTS:

1. *C* belongs to *SV*

2. *C.usage_status* = "retired"

3. For all concepts *C* belonging to *SV* such that *C.usage_status* = "current," *C.concept_definition* $\neq$ *new_definition*

4. *new_definition* $\neq$ *old_definition*

EFFECTS:

1. *C.concept_definition = new_definition*

Change Operation. **Add synonym**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*

2. String *new_synonym*

CONSTRAINTS:

1. *C* belongs to *SV*

2. *C.usage_status* ≠ "retired"

3. *C.concept_name* ≠ *new_synonym*

4. For all synonyms *Syn* belonging to *C.synonyms*, *Syn* ≠ *new_synonym*

EFFECTS:

1. $C.synonyms(SV_{i+1}) = C.synonyms(SV_i) \cup \{new\_synonym\}$

Change Operation. **Delete synonym**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*

2. String *synonym_to_delete*

CONSTRAINTS:

1. *C* belongs to *SV*

2. *C.usage_status* ≠ "retired"

3. *synonym_to_delete* belongs to *C.synonyms*

EFFECTS:

(1) *C.synonyms* = *C.synonyms* - {*synonym_to_delete*}

Change Operation. **Add abbreviation**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*
2. String *new_ abbreviation*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *C.usage_status* ≠ "retired"
3. C_i*.concept_name* ≠ *new_abbreviation*
4. For all abbreviations *Abbrev* belonging to *C.abbreviations*, *Abbrev* ≠ *new_abbreviation*

EFFECTS:

1. *C.abbreviations* = *C.abbreviations* U {*new_abbreviation*}

Change Operation. **Delete abbreviation**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*
2. String *abbreviation _to_delete*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *C.usage_status* ≠ "retired"
3. *abbreviation_to_delete* belongs to *C.abbreviations*

EFFECTS:

1. *C.abbreviations* = *C.abbreviations* - {*abbreviation_to_delete*}

Change Operation. **Replace UMLS code**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept
3. *old_umls_code = C.umls_code*

INPUT PARAMETERS:

1. Concept *C*
2. String *new_umls_code*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *C.usage_status* ≠ "retired"
3. *C.umls_code* ≠ *new_umls_code*
4. *new_umls_code* belongs to the set of concept unique identifiers in the UMLS

EFFECTS:

1. *C.umls_code = new_umls_code*

Change Operation. **Add parent**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept
3. *P* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*
2. Concept *P*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *C.usage_status* ≠ "retired"
3. *C* ≠ root of *SV*

4. *P* belongs to *SV*

5. *P.usage_status* ≠ "retired"

6. *P* ≠ *C*

7. *C* does not belong to *P.ancestors*

8. For every attribute–value pair or inherited attribute–value pair of *P*, AV_P (where $AV_P = \{A_P, V_P\}$), and for any attribute–value pair of *C*, AV_C, (where $AV_C = \{A_C, V_C\}$), if $A_P = A_C$ then $V_P = V_C$ or V_P is an ancestor of V_C.

EFFECTS:

1. $C.parents(SV_{i+1}) = C.parents(SV_i) \cup \{P\}$

2. $P.children(SV_{i+1}) = P.children(SV_i) \cup \{C\}$

3. $C.avpairs(SV_{i+1}) = C.avpairs(SV_i)$

   - ((*P.avpairs* U *P.inherited-avpairs*) INT *C.avpairs*)

4. For each *D* belonging to *C.descendants*,

   $D.avpairs(SV_{i+1}) = D.avpairs(SV_i)$

   - ((*P.avpairs* U *P.inherited-avpairs*) INT *D.avpairs*)

Change Operation. **Remove parent**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

3. *P* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*

2. Concept *P*

CONSTRAINTS:

1. *C* belongs to *SV*

2. *C* ≠ root of *SV*

3. *C.usage_status* ≠ "retired"

4. *P* belongs to *SV*

5. *P* belongs to *C.parents*

6. *C.parents* - {*P*} ≠ {} (*P* is not the only parent of *C*}

EFFECTS:

1. $C.parents(SV_{i+1}) = C.parents(SV_i)$ - {*P*}

2. $P.children(SV_{i+1}) = P.children(SV_i)$ - {*C*}

Change Operation. **Add child**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

3. *Ch* is a shared-vocabulary concept (not a root)

INPUT PARAMETERS:

1. Concept or *C*

2. Concept *Ch*

CONSTRAINTS:

1. *C* belongs to *SV*

2. *Ch* belongs to *SV*

3. *C.usage_status* ≠ "retired"

4. *Ch.usage_status* ≠ "retired"

5. *Ch* ≠ root of *SV*

6. *Ch* ≠ *C*

7. *Ch* does not belong to *C.ancestors*

8. For every attribute–value pair or inherited attribute–value pair of *C*, AV_C (where $AV_C = \{A_C, V_C\}$), and for any attribute–value pair of *Ch*, $AV_{Ch,}$ (where $AV_{Ch} = \{A_{Ch}, V_{Ch}\}$), if $A_C = A_{Ch}$ then $V_C = V_{Ch}$ or V_C is an ancestor of V_{Ch}.

EFFECTS:

1. $C.children(SV_{i+1}) = C.children(SV_i) \cup \{Ch\}$
2. $Ch.parents(SV_{i+1}) = Ch.parents(SV_i) \cup \{C\}$
3. $Ch.avpairs(SV_{i+1}) = Ch.avpairs(SV_i)$

   $- ((C.avpairs(SV_i) \cup C.inherited\text{-}avpairs(SV_i)) \cap Ch.avpairs(SV_i))$
4. For each *D* belonging to *Ch.descendants*,

   $D.avpairs(SV_{i+1}) = D.avpairs(SV_i)$

   $- ((C.avpairs(SV_i) \cup C.inherited\text{-}avpairs(SV_i)) \cap D.avpairs(SV_i))$

Change Operation. **Remove child**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept
3. *Ch* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*
2. Concept *Ch*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *Ch* belongs to *SV*
3. *C.usage_status* ≠ "retired"
4. *Ch* belongs to *C.children*
5. $Ch.parents - \{C\} \neq \{\}$ (*C* is not the only parent of *Ch*}

EFFECTS:

1. $C.children(SV_{i+1}) = C.children(SV_i) - \{Ch\}$
2. $Ch.parents(SV_{i+1}) = Ch.parents(SV_i) - \{C\}$

Change Operation. **Add attribute–value pair**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept
3. *A* is a shared-vocabulary attribute
4. *Val* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Attribute *A*
2. Concept *Val*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *A* belongs to *SV*
3. *Val* belongs to *SV*
4. *C.usage_status* ≠ "retired"
5. *A.usage_status* ≠ "retired"
6. *Val.usage_status* ≠ "retired"
7. For any attribute–value pair or inherited attribute–value pair of *C*, AV_C, (where $AV_C = \{A_C, V_C\}$), if $A_C = A$, then $V_C \neq Val$ (cannot add the attribute–value pair if it is already an attribute value pair or an inherited attribute value pair)
8. For any inherited attribute–value pair of *C*, AV_C, (where $AV_C = \{A_C, V_C\}$), if $A_C = A$, then V_C is an ancestor of *Val*
9. For any attribute–value pair of *D*, AV_D, where *D* belongs to *C.descendants*, (where $AV_D = \{A_D, V_D\}$), if $A_D = A$, then $Val = V_D$ or V_D is a descendant of Val

EFFECTS:

Let *AV* be the avpair that belongs to *C.avpairs* in SV_i such that *AV.attribute* = *A* and *AV.value* = *Val*.

1. $C.avpairs(SV_{i+1}) = C.avpairs(SV_i) \cup \{AV\}$

2. For all *D* belonging to *C.descendants*, if *D.avpairs* contains avpair *AV*, then

   $D.avpairs(SV_{i+1}) = D.avpairs(SV_i) - \{AV\}$

Change Operation. **Delete attribute–value pair**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *C* is a shared-vocabulary concept
3. *A* is a shared-vocabulary attribute
4. *Val* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Attribute *A*
2. Concept *Val*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *A* belongs to *SV*
3. *V* belongs to *SV*
4. *C.usage_status* ≠ "retired"
5. *A.usage_status* ≠ "retired"
6. *V.usage_status* ≠ "retired"
7. There exists attribute–value pair *AV* belonging to *C.avpairs* such that *AV.attribute* = *A* and *AV.value* = *Val*

EFFECTS:

1. $C.avpairs(SV_{i+1}) = C.avpairs(SV_i) - \{AV\}$

Change Operation. **Replace attribute value**

ASSUMPTIONS:

1. *SV* is a shared vocabulary

2. *C* is a shared-vocabulary concept

INPUT PARAMETERS

1. Attribute *A*
2. Concept *oldVal*
3. Concept *newVal*

CONSTRAINTS:

1. *C* belongs to *SV*
2. *A* belongs to *SV*
3. *oldVal* belongs to *SV*
4. *newVal* belongs to *SV*
5. *C.usage_status* ≠ "retired"
6. *A.usage_status* ≠ "retired"
7. *oldVal.usage_status* ≠ "retired"
8. *newVal.usage_status* ≠ "retired"
9. *newVal* ≠ *oldVal*
10. There exists attribute–value pair *AV* belonging to *C.avpairs* such that *AV.attribute* = *A* and *AV.value* = *oldVal*
11. There does not exist attribute–value pair *AV* belonging to *C.avpairs* such that *AV.attribute* = *A* and *AV.value* = *newVal*
12. For any inherited attribute–value pair of *C*, AV_C, (where $AV_C = \{A_C, V_C\}$), if $A_C = A$, then V_C is an ancestor of *Val*
13. For any attribute–value pair of *D*, where *D* belongs to *C.descendants*, ($AV_D = \{A_D, V_D\}$), if $A_D = A$, then V_D = *newVal* or V_D is a descendant of *Val*

EFFECTS:

Let *AV* be the avpair that belongs to *C.avpairs* in SV_i such that *AV.attribute* = *A* and *AV.value* = *oldVal*

1. *AV.value* = *newVal*
2. For all *D* belonging to *C.descendants*, if *D.avpairs* contains avpair *AV*, then

$D.avpairs(SV_{i+1}) = D.avpairs(SV_i) - \{AV\}$

Change Operation. **Replace attribute name**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *A* is a shared-vocabulary attribute
3. *old_attribute_name = A.attribute_name*

INPUT PARAMETERS:

(1) Attribute *A*

(2) String *new_attribute_name*

CONSTRAINTS:

(1) *A* belongs to *SV*

(2) *A.usage_status* ≠ "retired"

(3) *new_attribute_name* ≠ *old_attribute_name*

(4) For all attributes *A* belonging to *SV* such that *A.usage_status* = "current," *A.attribute_name* ≠ *new_attribute_name*

EFFECTS:

(1) *A.attribute_name = new_attribute_name*

Change Operation. **Replace attribute definition**

ASSUMPTIONS:

1. *SV* is a shared vocabulary
2. *A* is a shared-vocabulary attribute
3. *old_attribute_definition = A.attribute_definition*

INPUT PARAMETERS:

1. Attribute *A*
2. String *new_attribute_definition*

CONSTRAINTS:

1. *A* belongs to *SV*
2. *A.usage_status* ≠ “retired”
3. *new_attribute_definition* ≠ *old_attribute_definition*
4. For all attributes *A* belonging to *SV* such that *A.usage_status* = “current,” *A.attribute_definition* ≠ *new_attribute_definition*

EFFECTS:

1. *A.attribute_definition* = *new_attribute_definition*

Appendix D: Local-Vocabulary Change Model

The **local-vocabulary change model** comprises of a set of change operations and three requirements. The change operations may be performed on local vocabulary *LV*, one change at a time. Formal definitions for change operations that are defined in both the shared vocabulary and the local vocabulary were given in Appendix C; these definitions hold for the local-vocabulary change operations, but are modified by assumptions that override assumptions for shared-vocabulary change operations, and may have additional constraints and effects. There are four additional change operations that exist in the local vocabulary, but that do not exist in the shared vocabulary. This appendix gives specifications for the local modifications to shared-vocabulary–change operations, and for the four additional change operations.

Definition. A change operation is a **valid local-vocabulary change operation** if it is one of the following:

1. A valid shared-vocabulary change operation
2. Hide concept
3. Preserve concept
4. Hide attribute
5. Preserve attribute

Requirement D1. If LV_i fulfills the requirements of a local vocabulary, and *change_op* is a valid local-vocabulary change operation, which causes the local vocabulary to be transformed from an initial state, LV_i, to a subsequent state, LV_{i+1}, then LV_{i+1} also fulfills the requirements of a local vocabulary.

change_op

LV_i ----------> LV_{i+1}

Requirement D2. If LV_i fulfills the requirements of a local vocabulary, $change\_op_1$, $change\_op_2$, ...$change\_op_n$ are valid local-vocabulary change operations, and LV_{i+1}, LV_{i+2}, ...LV_{i+n} are the states of the local vocabulary after each change operation respectively, then LV_{i+n} also fulfills the structural requirements of a local vocabulary.

$$LV_i \xrightarrow{change\_op_1} LV_{i+1} \xrightarrow{change\_op_2} LV_{i+2} \ \ldots \xrightarrow{change\_op_n} LV_{i+n}$$

For each change operation that exists in both the shared vocabulary and the local vocabulary, the description of the operation given in Appendix C applies to the operation by the same name in the local vocabulary, but "*SV*" is replaced by "*LV*," and the following modifications hold.

Change Operation. **Add concept** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. *P* is a local-vocabulary concept

ADDITIONAL EFFECT:

1. *C.site_of_origin* = "local_only"

Change Operation. **Retire concept** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. *C* is a local-vocabulary concept
3. P_1, P_2, ... P_n are local-vocabulary concepts
4. Ch_1, Ch_2, ... Ch_m are local-vocabulary concepts

ADDITIONAL CONSTRAINTS:

1. *C.site_of_origin* = “local_only”

Change Operation. **Merge two concepts into one of the two concepts** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. C_1 is a local-vocabulary concept
3. C_2 is a local-vocabulary concept
4. *C_to_keep* is a local-vocabulary concept
5. *C_to_retire* is a local-vocabulary concept

ADDITIONAL CONSTRAINT:

1. *C_to_retire.site_of_origin* = “local_only”

Change Operation. **Merge two concepts into new concept** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. C_1 is a local-vocabulary concept
3. C_2 is a local-vocabulary concept
4. *P* is a local-vocabulary concept

ADDITIONAL CONSTRAINTS:

1. C_1*.site_of_origin* = “local_only”
2. C_2*.site_of_origin* = “local_only”

EFFECTS:

1. If *P.site_of_origin* = “shared,” then *P.site_of_origin* = “locally_modified_shared”

Change Operation. **Split concept into two new concepts** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. *C* is a local-vocabulary concept
3. P_1 is a local-vocabulary concept
4. P_2 is a local-vocabulary concept

ADDITIONAL CONSTRAINTS:

1. *C.site_of_origin* = "local_only"

EFFECTS:

1. If P_1*.site_of_origin* = "shared," then P_1*.site_of_origin* = "locally_modified_shared"
2. If P_2*.site_of_origin* = "shared," then P_2*.site_of_origin* = "locally_modified_shared"
3. For all concepts *P* belonging to *parents_to_add_to_first_concept*, if *P.site_of_origin* = "shared," then *P.site_of_origin* = "locally_modified_shared"
4. For all concepts *P* belonging to *parents_to_add_to_second_concept*, if *P.site_of_origin* = "shared," then *P.site_of_origin* = "locally_modified_shared"
5. For all concepts *Ch* belonging to *children_to_add_to_first_concept*, if *Ch.site_of_origin* = "shared," then *Ch.site_of_origin* = "locally_modified_shared"
6. For all concepts *Ch* belonging to *children_to_add_to_second_concept*, if *Ch.site_of_origin* = "shared," then *Ch.site_of_origin* = "locally_modified_shared"

Change Operation. **Add attribute** or **retire attribute** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary

2. *A* is a local-vocabulary attribute

ADDITIONAL EFFECTS:

1. *A.site_of_origin* = “local_only”

Change Operation. **Merge two attributes into one of the two attributes** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. A_1 is a local-vocabulary attribute
3. A_2 is a local-vocabulary attribute
4. *A_to_keep* is a local-vocabulary attribute
5. *A_to_retire* is a local-vocabulary attribute

ADDITIONAL CONSTRAINT:

1. *A_to_retire.site_of_origin* = “local_only”

Change Operation. **Merge two attributes into new attribute** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. A_1 is a local-vocabulary attribute
3. A_2 is a local-vocabulary attribute

ADDITIONAL CONSTRAINTS:

1. A_1*.site_of_origin* = “local_only”
2. A_2*.site_of_origin* = “local_only”

Change Operation. **Replace concept name, correct concept name, replace concept definition, add synonym, delete synonym, add abbreviation, delete abbreviation,** or **replace UMLS code** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary

2. *C* is a local-vocabulary concept

ADDITIONAL EFFECT:

1. If *C.site_of_origin* = "shared," then *C.site_of_origin* = "locally_modified_shared"

Change Operation. **Add child** and **remove child** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary

2. *C* is a local-vocabulary concept

3. *Ch* is a local-vocabulary concept

4. *Ch* is not a local-vocabulary root concept

ADDITIONAL EFFECT:

1. If *C.site_of_origin* = "shared," then *C.site_of_origin* = "locally_modified_shared"

Change Operation. **Add attribute−value pair, delete attribute–value pair,** or **replace attribute value** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary

2. *C* is a local-vocabulary concept

3. *A* is a local-vocabulary attribute

4. *Val* is a local-vocabulary concept

ADDITIONAL EFFECT:

1. If *C.site_of_origin* = "shared," then *C.site_of_origin* = "locally_modified_shared"

Change Operation. **Replace attribute name** or **replace attribute definition** (Local Vocabulary)

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. *A* is a local-vocabulary attribute

ADDITIONAL EFFECT:

1. If *A.site_of_origin* = "shared," then *A.site_of_origin* = "locally_modified_shared"

The following operation exists only in the local vocabulary. They do not exist in the shared vocabulary.

Change Operation. **Hide concept**

ASSUMPTIONS:

1. *LV* is a local vocabulary
2. *C* is a local-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*

CONSTRAINTS:

1. *C* belongs to *LV*
2. *C.usage_status* = "current"
3. *C.site_of_origin* = "shared" or "locally_modified_shared"

EFFECTS:

1. *C.usage_status* = "hidden"

Change Operation. **Preserve concept**

ASSUMPTIONS:

1. *LV* is a local vocabulary

2. *C* is a shared-vocabulary concept

INPUT PARAMETERS:

1. Concept *C*

CONSTRAINTS:

1. *C* belongs to *LV*

2. *C.usage_status* = “retired”

3. *C.site_of_origin* = “shared” or “locally_modified_shared”

EFFECTS:

1. *C.usage_status* = “preserved”

Change Operation. **Hide attribute**

ASSUMPTIONS:

1. *LV* is a local vocabulary

2. *A* is a local-vocabulary attribute

INPUT PARAMETER:

1. Attribute *A*

CONSTRAINTS:

1. *A* belongs to *LV*

2. *A.usage_status* = “current”

3. *A.site_of_origin* = “shared” or “locally_modified_shared”

EFFECT:

1. A.usage_status = “hidden”

Change Operation. **Preserve attribute**

ASSUMPTIONS:

1. *LV* is a local vocabulary

2. *C* is a shared-vocabulary concept

INPUT PARAMETER:

1. Attribute *A*

CONSTRAINTS:

1. *A* belongs to *LV*
2. *A.usage_status* = “retired”
3. *A.site_of_origin* = “shared” or “locally_modified_shared”

EFFECT:

1. *A.usage_status* = “preserved”

Appendix E: Shared-Vocabulary Log Model

The **shared-vocabulary log model** is specified by definitions and an axiom that relate to documentation of changes completed in a shared vocabulary.

Definition. A **change-operation record** is the documentation of data elements for a single change operation. The set of elements depends on the type of change operation.

1. If the change is a *vocabulary change operation*, the set of data elements in the change record includes universal data elements and change-specific data elements.
2. If the change is a *concept change operation*, the set of data elements in the change record includes universal data elements, concept–change-operation data elements, and change-specific data elements.
3. If the change is an *attribute change operation*, the set of data elements in the change record includes universal data elements, attribute–change-operation data elements, and change-specific data elements.

The definitions of vocabulary change operations, concept-change operation, and attribute-change operation were given in Section 3.2. The definitions of universal data elements, concept–change-operation data elements, attribute–change-operation data elements, and change-specific data elements are given in this appendix.

Definition. If CR_1, CR_2, ... CR_n are change records that describe valid shared-vocabulary change operations that have been applied to shared vocabulary *SV* in sequence, then the ordered set {CR_1, CR_2, ... CR_n} is a **shared-vocabulary log** for *SV*.

Definition. The **universal data elements** for a change operation are the following:

1. Name of change operation (denoted *change_name*)
2. Timestamp
3. Author
4. Explanation

5. Sequence number assigned this change (denoted *seq_num*)

Definition. The **concept–change-operation data elements** are the following:

1. Current concept name
2. Concept identifier

Definition. The **attribute–change-operation data elements** are the following:

1. Current attribute name
2. Attribute identifier

For change record *CR*, these data elements are referred to as *CR.attribute_name and CR.attribute_id*.

Definition. The **change-specific data elements for *add concept*** are the following:

1. Concept name of added concept
2. Concept identifier of added concept
3. Concept name of parent of added concept
4. Concept identifier of parent of added concept

Definition. The **change-specific data elements for *retire concept*** are the following:

1. Concept name of retired concept
2. Concept identifier of retired concept

Definition. The **change-specific data elements for *merge two concepts into one of the two concepts*** are the following:

1. Concept name of first concept
2. Concept identifier of first concept
3. Concept name of second concept

4. Concept identifier of second concept
5. Concept name of concept to keep
6. Concept identifier of concept to keep
7. Concept name of concept to retire
8. Concept identifier of concept to retire

Definition. The **change-specific data elements for *merge two concepts into new concept* are** the following:

1. Concept name of first concept
2. Concept identifier of first concept
3. Concept name of second concept
4. Concept identifier of second concept
5. Concept name of new concept
6. Concept identifier of new concept
7. Concept name of parent of new concept
8. Concept identifier of parent of new concept

Definition. The **change-specific data elements for *split concept into two new concepts*** are the following:

1. Concept name of split concept
2. Concept identifier of split concept
3. Concept name of first new concept
4. Concept identifier of first new concept
5. Concept name of second new concept
6. Concept identifier of second new concept
7. Concept name of parent of first new concept
8. Concept identifier of parent of first new concept

9. Concept name of parent of second new concept

10. Concept identifier of parent of second new concept

Definition. The **change-specific data elements for *add attribute*** are the following:

1. Attribute name of added attribute

2. Attribute identifier of added attribute

Definition. The **change-specific data elements for *retire attribute*** are the following:

1. Attribute name of retired attribute

2. Attribute identifier of retired attribute

Definition. The **change-specific data elements for *merge two attributes into one of the two attributes*** are the following:

1. Attribute name of first attribute

2. Attribute identifier of first attribute

3. Attribute name of second attribute

4. Attribute identifier of second attribute

5. Attribute name of attribute to keep

6. Attribute identifier of attribute to keep

7. Attribute name of attribute to retire

8. Attribute identifier of attribute to retire

Definition. The **change-specific data elements for *merge two attributes into new attribute* are** the following:

1. Attribute name of first attribute

2. Attribute identifier of first attribute

3. Attribute name of second attribute

4. Attribute identifier of second attribute
5. Attribute name of new attribute
6. Attribute identifier of new attribute

Definition. The **change-specific data elements for *replace concept name*** are the following:

1. New concept name
2. Old concept name

Definition. The **change-specific data elements for *correct concept name*** are the following:

1. New concept name
2. Old concept name

Definition. The **change-specific data elements for *replace concept definition*** are the following:

1. New concept definition
2. Old concept definition

Definition. The **change-specific data elements for *add synonym*** are the following:

1. Added synonym

Definition. The **change-specific data elements for *delete synonym*** are the following:

1. Deleted synonym

Definition. The **change-specific data elements for *add abbreviation*** are the following:

1. Added abbreviation

Definition. The **change-specific data elements for *delete abbreviation*** are the following:

1. Deleted abbreviation

Definition. The **change-specific data elements for *replace UMLS code*** are the following:

1. New UMLS code
2. Old UMLS code

Definition. The **change-specific data elements for *add parent*** are the following:

1. Added parent name
2. Added parent identifier

Definition. The **change-specific data elements for *remove parent*** are the following:

1. Removed parent name
2. Removed parent identifier

Definition. The **change-specific data elements for *add child*** are the following:

1. Added child name
2. Added child identifier

Definition. The **change-specific data elements for *remove child*** are the following:

1. Removed child name
2. Removed child identifier

Definition. The **change-specific data elements for *add attribute–value pair*** are the following:

1. Attribute name of attribute in added attribute–value pair
2. Attribute identifier of attribute in added attribute–value pair
3. Concept name of value in added attribute–value pair
4. Concept identifier of value in added attribute–value pair

Definition. The **change-specific data elements for *delete attribute–value pair*** are the following:

1. Attribute name of attribute in deleted attribute–value pair
2. Attribute identifier of attribute in deleted attribute–value pair
3. Concept name of value in deleted attribute–value pair
4. Concept identifier of value in deleted attribute–value pair

Definition. The **change-specific data elements for *replace attribute value*** are the following:

1. Attribute name
2. Attribute identifier
3. New value name
4. New value identifier
5. Old value name
6. Old value identifier

Definition. The **change-specific data elements for *replace attribute name*** are the following:

1. New attribute name
2. Old attribute name

Definition. The **change-specific data elements for *replace attribute definition*** are the following:

1. New attribute definition
2. Old attribute definition

Axiom E1

Suppose:

1. *SV* is a shared vocabulary
2. *SVLog* is a shared log
3. *n* is the number of shared-vocabulary change operations applied to *SV*
4. sv_op_i and sv_op_j are the *i*th and *j*th shared-vocabulary change operations, respectively, applied to *SV* ($i, j = 1$ to n)
5. $\{sv\_op_1, sv\_op_2, \ldots sv\_op_n\}$ is the ordered set of *n* shared-vocabulary change operations applied to *SV*
6. CR_i and CR_j are the *i*th and *j*th change records, respectively, in *SVLog* ($i, j = 1$ to n)
7. $SVLog = \{CR_1, CR_2, \ldots CR_n\}$

If sv_op_i was applied to *SV* before sv_op_j was applied to *SV*, then $CR_i.seq\_num < CR_j.seq\_num$ in *SVLog*. (The order of change records in the log corresponds to the order in which change records were applied to the shared vocabulary.)

Appendix F: Document Type Definition (DTD) for Shared Vocabulary

```
<!DOCTYPE simple [
<!ELEMENT VOCABULARY (
      ROOT,
      ALL_ATTRIBUTES,
      ALL_CONCEPTS,
      ALL_LINKS)>
<!ELEMENT ROOT (
      ROOT_NAME,
      ROOT_ID,
      ROOT_USAGE_STATUS)>
<!ELEMENT ROOT_NAME (#PCDATA)>
<!ELEMENT ROOT_ID (#PCDATA)>
<!ELEMENT ROOT_USAGE_STATUS (#PCDATA)>
<!ELEMENT ALL_CONCEPTS (
      CONCEPT*)>
<!ELEMENT CONCEPT (
      CONCEPT_NAME,
      CONCEPT_ID,
      CONCEPT_USAGE_STATUS,
      CONCEPT_DEFINITION?,
      UMLS_CODE?,
      SYNONYM_SET?,
      ABBREVIATION_SET?)>
<!ELEMENT CONCEPT_NAME (#PCDATA)>
<!ELEMENT CONCEPT_ID (#PCDATA)>
<!ELEMENT CONCEPT_USAGE_STATUS (#PCDATA)>
<!ELEMENT UMLS_CODE (#PCDATA)>
<!ELEMENT SYNONYM_SET (
      SYNONYM*)>
<!ELEMENT SYNONYM (#PCDATA)>
<!ELEMENT ABBREVIATION_SET (
      ABBREVIATION*)>
<!ELEMENT ABBREVIATION (#PCDATA)>
<!ELEMENT ALL_ATTRIBUTES (
      ATTRIBUTE*)>
<!ELEMENT ATTRIBUTE (
```

```
      ATTRIBUTE_NAME,
      ATTRIBUTE_ID,
      ATTRIBUTE_USAGE_STATUS,
      ATTRIBUTE_DEFINITION?)>
<!ELEMENT ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT ATTRIBUTE_ID (#PCDATA)>
<!ELEMENT ATTRIBUTE_USAGE_STATUS (#PCDATA)>
<!ELEMENT ATTRIBUTE_DEFINITION (#PCDATA)>
<!ELEMENT ALL_LINKS (
      LINKS_FOR_ROOT*,
      LINKS_FOR_CONCEPT*)>
<!ELEMENT LINKS_FOR_ROOT (
      ROOT_NAME,
      ROOT_ID,
      CHILD_SET)>
<!ELEMENT LINKS_FOR_CONCEPT (
      CONCEPT_NAME,
      CONCEPT_ID,
      PARENT_SET,
      CHILD_SET,
      AVPAIR_SET)>
<!ELEMENT PARENT_SET (
      PARENT+)>
<!ELEMENT PARENT (
      PARENT_NAME,
      PARENT_ID)>
<!ELEMENT PARENT_NAME (#PCDATA)>
<!ELEMENT PARENT_ID (#PCDATA)>
<!ELEMENT CHILD_SET (
      CHILD*)>
<!ELEMENT CHILD (
      CHILD_NAME,
      CHILD_ID)>
<!ELEMENT CHILD_NAME (#PCDATA)>
<!ELEMENT CHILD_ID (#PCDATA)>
<!ELEMENT AVPAIR_SET (
      AVPAIR*)>
<!ELEMENT AVPAIR (
      AVPAIR_ATTRIBUTE,
      AVPAIR_VALUE)>
```

```
<!ELEMENT AVPAIR_ATTRIBUTE (
      AVPAIR_ATTRIBUTE_NAME,
      AVPAIR_ATTRIBUTE_ID)>
<!ELEMENT AVPAIR_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT AVPAIR_ATTRIBUTE_ID (#PCDATA)>
<!ELEMENT AVPAIR_VALUE (
      AVPAIR_VALUE_NAME,
      AVPAIR_VALUE_ID)>
<!ELEMENT AVPAIR_VALUE_NAME (#PCDATA)>
<!ELEMENT AVPAIR_VALUE_ID (#PCDATA)>
<!ELEMENT RETIRED_PARENT_SET (
      RETIRED_PARENT*)>
<!ELEMENT RETIRED_PARENT (
      RETIRED_PARENT_NAME,
      RETIRED_PARENT_ID)>
<!ELEMENT RETIRED_PARENT_NAME (#PCDATA)>
<!ELEMENT RETIRED_PARENT_ID (#PCDATA)>
<!ELEMENT RETIRED_CHILDREN_SET (
      RETIRED_CHILD*)>
<!ELEMENT RETIRED_CHILD (
      RETIRED_CHILD_NAME,
      RETIRED_CHILD_ID)>
<!ELEMENT RETIRED_CHILD_NAME (#PCDATA)>
<!ELEMENT RETIRED_CHILD_ID (#PCDATA)>
]>
```

Key:

Separated by commas (,): must appear in order listed

Plus (+): must appear at least once and possibly more

Asterisk (*): may appear any number of times, including zero times

Question mark (?): optional element, but if it appears, it can appear only once

Vertical slash (|): means OR

Appendix G: Document Type Definition (DTD) for Log

```
<!DOCTYPE simple [
<!ELEMENT LOG (
      (VOCABULARY_CHANGE |
      CONCEPT_CHANGE |
      ATTRIBUTE_CHANGE)*)>
<!ELEMENT VOCABULARY_CHANGE
      (UNIVERSAL_CHANGE_FEATURES,
      (CONCEPT_ADDITION |
      CONCEPT_RETIREMENT |
      ATTRIBUTE_ADDITION |
      ATTRIBUTE_RETIREMENT |
      CONCEPT_MERGE_INTO_EXISTING |
      CONCEPT_MERGE_INTO_NEW |
      CONCEPT_SPLIT |
      ATTRIBUTE_MERGE_INTO_EXISTING |
      ATTRIBUTE_MERGE_INTO_NEW))>
<!ELEMENT CONCEPT_CHANGE (
      UNIVERSAL_CHANGE_FEATURES,
      CONCEPT_CHANGE_FEATURES,
      (CONCEPT_NAME_REPLACEMENT |
      CONCEPT_DEFINITION_REPLACEMENT |
      SYNONYM_ADDITION |
      SYNONYM_DELETION |
      ABBREVIATION_ADDITION |
      ABBREVIATION_DELETION |
      UMLS_CODE_REPLACEMENT |
      PARENT_ADDITION |
      PARENT_REMOVAL |
      CHILD_ADDITION |
      CHILD_REMOVAL |
      AVPAIR_ADDITION |
      AVPAIR_DELETION |
      ATTRIBUTE_VALUE_REPLACEMENT))>
<!ELEMENT ATTRIBUTE_CHANGE (
      UNIVERSAL_CHANGE_FEATURES,
      ATTRIBUTE_CHANGE_FEATURES,
```

```
      (ATTRIBUTE_NAME_REPLACEMENT |
      ATTRIBUTE_DEFINITION_REPLACEMENT))>
<!ELEMENT UNIVERSAL_CHANGE_FEATURES (
      SEQUENCE_NUMBER,
      TYPE_OF_CHANGE,
      TIME_STAMP,
      AUTHOR?,
      EXPLANATION?)>
<!ELEMENT SEQUENCE_NUMBER (#PCDATA)>
<!ELEMENT TYPE_OF_CHANGE (#PCDATA)>
<!ELEMENT TIME_STAMP (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT EXPLANATION (#PCDATA)>
<!ELEMENT CONCEPT_CHANGE_FEATURES (
      CURRENT_CONCEPT_NAME,
      CONCEPT_ID)>
<!ELEMENT CURRENT_CONCEPT_NAME (#PCDATA)>
<!ELEMENT CONCEPT_ID (#PCDATA)>
<!ELEMENT ATTRIBUTE_CHANGE_FEATURES (
      CURRENT_ATTRIBUTE_NAME,
      ATTRIBUTE_ID)>
<!ELEMENT CURRENT_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT ATTRIBUTE_ID (#PCDATA)>
<!ELEMENT CONCEPT_ADDITION (
      ADDED_CONCEPT_NAME,
      ADDED_CONCEPT_ID,
      ADDED_CONCEPT_PARENT_NAME,
      ADDED_CONCEPT_PARENT_ID)>
<!ELEMENT ADDED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT ADDED_CONCEPT_ID (#PCDATA)>
<!ELEMENT ADDED_CONCEPT_PARENT_NAME (#PCDATA)>
<!ELEMENT ADDED_CONCEPT_PARENT_ID (#PCDATA)>
<!ELEMENT CONCEPT_RETIREMENT (
      RETIRED_CONCEPT_NAME,
      RETIRED_CONCEPT_ID)>
<!ELEMENT RETIRED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT RETIRED_CONCEPT_ID (#PCDATA)>
<!ELEMENT ATTRIBUTE_ADDITION (
      ADDED_ATTRIBUTE_NAME,
      ADDED_ATTRIBUTE_ID)>
```

```
<!ELEMENT ADDED_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT ADDED_ATTRIBUTE_ID (#PCDATA)>
<!ELEMENT ATTRIBUTE_RETIREMENT (
      RETIRED_ATTRIBUTE_NAME,
      RETIRED_ATTRIBUTE_ID)>
<!ELEMENT RETIRED_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT RETIRED_ATTRIBUTE_ID (#PCDATA)>
<!ELEMENT CONCEPT_MERGE_INTO_EXISTING (
      FIRST_CONCEPT_NAME,
      FIRST_CONCEPT_ID,
      SECOND_CONCEPT_NAME,
      SECOND_CONCEPT_ID,
      CONCEPT_TO_KEEP_NAME,
      CONCEPT_TO_KEEP_ID,
      CONCEPT_TO_RETIRE_NAME,
      CONCEPT_TO_RETIRE_ID)>
<!ELEMENT CONCEPT_MERGE_INTO_NEW (
      FIRST_CONCEPT_NAME,
      FIRST_CONCEPT_ID,
      SECOND_CONCEPT_NAME,
      SECOND_CONCEPT_ID,
      CREATED_CONCEPT_NAME,
      CREATED_CONCEPT_ID,
      PARENT_OF_CREATED_CONCEPT_NAME,
      PARENT_OF_CREATED_CONCEPT_ID)>
<!ELEMENT FIRST_CONCEPT_NAME (#PCDATA)>
<!ELEMENT FIRST_CONCEPT_ID (#PCDATA)>
<!ELEMENT SECOND_CONCEPT_NAME (#PCDATA)>
<!ELEMENT SECOND_CONCEPT_ID (#PCDATA)>
<!ELEMENT CONCEPT_TO_KEEP_NAME (#PCDATA)>
<!ELEMENT CONCEPT_TO_KEEP_ID (#PCDATA)>
<!ELEMENT CONCEPT_TO_RETIRE_NAME (#PCDATA)>
<!ELEMENT CONCEPT_TO_RETIRE_ID (#PCDATA)>
<!ELEMENT CREATED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT CREATED_CONCEPT_ID (#PCDATA)>
<!ELEMENT PARENT_OF_CREATED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT PARENT_OF_CREATED_CONCEPT_ID (#PCDATA)>
<!ELEMENT CONCEPT_SPLIT (
      SPLIT_CONCEPT_NAME,
      SPLIT_CONCEPT_ID,
```

```
      FIRST_CREATED_CONCEPT_NAME,
      FIRST_CREATED_CONCEPT_ID,
      SECOND_CREATED_CONCEPT_NAME,
      SECOND_CREATED_CONCEPT_ID,
      PARENT_OF_FIRST_CREATED_CONCEPT_NAME,
      PARENT_OF_FIRST_CREATED_CONCEPT_ID,
      PARENT_OF_SECOND_CREATED_CONCEPT_NAME,
      PARENT_OF_SECOND_CREATED_CONCEPT_ID)>
<!ELEMENT SPLIT_CONCEPT_NAME (#PCDATA)>
<!ELEMENT SPLIT_CONCEPT_ID (#PCDATA)>
<!ELEMENT FIRST_CREATED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT FIRST_CREATED_CONCEPT_ID (#PCDATA)>
<!ELEMENT SECOND_CREATED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT SECOND_CREATED_CONCEPT_ID (#PCDATA)>
<!ELEMENT PARENT_OF_FIRST_CREATED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT PARENT_OF_FIRST_CREATED_CONCEPT_ID (#PCDATA)>
<!ELEMENT PARENT_OF_SECOND_CREATED_CONCEPT_NAME (#PCDATA)>
<!ELEMENT PARENT_OF_SECOND_CREATED_CONCEPT_ID (#PCDATA)>
<!ELEMENT ATTRIBUTE_MERGE_INTO_EXISTING (
      FIRST_ATTRIBUTE_NAME,
      FIRST_ATTRIBUTE_ID,
      SECOND_ATTRIBUTE_NAME,
      SECOND_ATTRIBUTE_ID,
      ATTRIBUTE_TO_KEEP_NAME,
      ATTRIBUTE_TO_KEEP_ID,
      ATTRIBUTE_TO_RETIRE_NAME,
      ATTRIBUTE_TO_RETIRE_ID)>
<!ELEMENT ATTRIBUTE_MERGE_INTO_NEW (
      FIRST_ATTRIBUTE_NAME,
      FIRST_ATTRIBUTE_ID,
      SECOND_ATTRIBUTE_NAME,
      SECOND_ATTRIBUTE_ID,
      CREATED_ATTRIBUTE_NAME,
      CREATED_ATTRIBUTE_ID)>
<!ELEMENT FIRST_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT FIRST_ATTRIBUTE_ID (#PCDATA)>
<!ELEMENT SECOND_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT SECOND_ATTRIBUTE_ID (#PCDATA)>
<!ELEMENT ATTRIBUTE_TO_KEEP_ID (#PCDATA)>
<!ELEMENT ATTRIBUTE_TO_RETIRE_ID (#PCDATA)>
```

```
<!ELEMENT CREATED_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT CREATED_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT CONCEPT_NAME_REPLACEMENT (
      NEW_CONCEPT_NAME,
      OLD_CONCEPT_NAME)>
<!ELEMENT CONCEPT_DEFINITION_REPLACEMENT (
      NEW_CONCEPT_DEFINITION,
      OLD_CONCEPT_DEFINITION)>
<!ELEMENT SYNONYM_ADDITION (
      ADDED_SYNONYM)>
<!ELEMENT SYNONYM_DELETION (
      DELETED_SYNONYM)>
<!ELEMENT ABBREVIATION_ADDITION (
      ADDED_ABBREVIATION)>
<!ELEMENT ABBREVIATION_DELETION (
      DELETED_ABBREVIATION)>
<!ELEMENT UMLS_CODE_REPLACEMENT (
      NEW_UMLS_CODE,
      OLD_UMLS_CODE)>
<!ELEMENT PARENT_ADDITION (
      ADDED_PARENT_NAME,
      ADDED_PARENT_ID)>
<!ELEMENT PARENT_REMOVAL (
      REMOVED_PARENT_NAME,
      REMOVED_PARENT_ID)>
<!ELEMENT CHILD_ADDITION (
      ADDED_CHILD_NAME,
      ADDED_CHILD_ID)>
<!ELEMENT CHILD_REMOVAL (
      REMOVED_CHILD_NAME,
      REMOVED_CHILD_ID)>
<!ELEMENT AVPAIR_ADDITION (
      ADDED_AVPAIR_ATTRIBUTE_NAME,
      ADDED_AVPAIR_ATTRIBUTE_ID,
      ADDED_AVPAIR_VALUE_NAME,
      ADDED_AVPAIR_VALUE_ID)>
<!ELEMENT AVPAIR_DELETED (
      DELETED_AVPAIR_ATTRIBUTE_NAME,
      DELETED_AVPAIR_ATTRIBUTE_ID,
      DELETED_AVPAIR_VALUE_NAME,
```

```
      DELETED_AVPAIR_VALUE_ID)>
<!ELEMENT ATTRIBUTE_VALUE_REPLACEMENT (
      ATTRIBUTE_FOR_VALUE_REPLACEMENT_NAME,
      ATTRIBUTE_FOR_VALUE_REPLACEMENT_ID,
      NEW_VALUE_NAME,
      NEW_VALUE_ID,
      OLD_VALUE_NAME,
      OLD_VALUE_ID)>
<!ELEMENT ATTRIBUTE_NAME_REPLACEMENT (
      NEW_ATTRIBUTE_NAME,
      OLD_ATTRIBUTE_NAME)>
<!ELEMENT ATTRIBUTE_DEFINITION_REPLACEMENT (
      NEW_ATTRIBUTE_DEFINITION,
      OLD_ATTRIBUTE_DEFINITION)>
<!ELEMENT NEW_CONCEPT_NAME (#PCDATA)>
<!ELEMENT OLD_CONCEPT_NAME (#PCDATA)>
<!ELEMENT NEW_CONCEPT_DEFINITION (#PCDATA)>
<!ELEMENT OLD_CONCEPT_DEFINITION (#PCDATA)>
<!ELEMENT ADDED_SYNONYM (#PCDATA)>
<!ELEMENT DELETED_SYNONYM (#PCDATA)>
<!ELEMENT ADDED_ABBREVIATION (#PCDATA)>
<!ELEMENT DELETED_ABBREVIATION (#PCDATA)>
<!ELEMENT NEW_UMLS_CODE (#PCDATA)>
<!ELEMENT OLD_UMLS_CODE (#PCDATA)>
<!ELEMENT ADDED_PARENT_NAME (#PCDATA)>
<!ELEMENT ADDED_PARENT_ID (#PCDATA)>
<!ELEMENT REMOVED_PARENT_NAME (#PCDATA)>
<!ELEMENT REMOVED_PARENT_ID (#PCDATA)>
<!ELEMENT ADDED_CHILD_NAME (#PCDATA)>
<!ELEMENT ADDED_CHILD_ID (#PCDATA)>
<!ELEMENT REMOVED_CHILD_NAME (#PCDATA)>
<!ELEMENT REMOVED_CHILD_ID (#PCDATA)>
<!ELEMENT ADDED_AVPAIR_NAME (#PCDATA)>
<!ELEMENT ADDED_AVPAIR_ID (#PCDATA)>
<!ELEMENT ADDED_AVPAIR_VALUE_NAME (#PCDATA)>
<!ELEMENT ADDED_AVPAIR_VALUE_ID (#PCDATA)>
<!ELEMENT DELETED_AVPAIR_NAME (#PCDATA)>
<!ELEMENT DELETED_AVPAIR_ID (#PCDATA)>
<!ELEMENT DELETED_AVPAIR_VALUE_NAME (#PCDATA)>
<!ELEMENT DELETED_AVPAIR_VALUE_ID (#PCDATA)>
```

```
<!ELEMENT ATTRIBUTE_FOR_VALUE_REPLACEMENT_NAME (#PCDATA)>
<!ELEMENT ATTRIBUTE_FOR_VALUE_REPLACEMENT_ID (#PCDATA)>
<!ELEMENT NEW_VALUE_NAME (#PCDATA)>
<!ELEMENT NEW_VALUE_ID (#PCDATA)>
<!ELEMENT OLD_VALUE_NAME (#PCDATA)>
<!ELEMENT OLD_VALUE_ID (#PCDATA)>
<!ELEMENT NEW_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT OLD_ATTRIBUTE_NAME (#PCDATA)>
<!ELEMENT NEW_ATTRIBUTE_DEFINITION (#PCDATA)>
<!ELEMENT OLD_ATTRIBUTE_DEFINITION (#PCDATA)>
]>
```

Key:

Separated by commas (,): must appear in order listed

Plus (+): must appear at least once and possibly more

Asterisk (*): may appear any number of times, including zero times

Question mark (?): optional element, but if it appears, it can appear only once

Vertical slash (|): means OR

Appendix H: Synchronization Actions

Table H 1. Action choices for *add concept.*

Action	Steps
Action 1	Step 1: Add concept
Action 2 (if local name conflicts)	Step 1: Rename local concept with conflicting name Step 2: Add concept
Action 3 (if local name conflicts)	Step 1: Rename local concept with conflicting name Step 2: Add concept Step 3: Rename new concept Step 4: Rename local concept back to original name
Action 4 (if local concept has same meaning)	Step 1: Add concept Step 2: Merge local concept into shared concept Step 3: Retain local name as synonym
Action 5 (if local concept has same meaning)	Step 1: Add concept Step 2: Merge local concept into shared concept Step 3: Rename concept with local name Step 4: Retain shared name as synonym
Action 6 (if local concept has same meaning)	Step 1: Add concept Step 2: Merge local concept into shared concept Step 3: Rename concept with local name
Action 7	Step 1: Add concept Step 2: Hide concept

Table H 2. Action choices for *retire concept.*

Action	Steps
Action 1	Step 1: Retire concept
Action 2	Step 1: Retire concept Step 2: Preserve concept

Table H 3. Action choices for *split concept into two new concepts.*

Action	Steps
Action 1	Step 1: Split concept into two new concepts
Action 2	Step 1: Split concept into two new concepts Step 2: Preserve retired concept

Table H 4. Action choices for *merge two concepts into one of the two concepts.*

Action	Steps
Action 1	Step 1: Merge two concepts into one of the two concepts
Action 2	Step 1: Merge two concepts into one of the two concepts Step 2: Preserve retired concept
Action 3 (if one concept is non-parent ancestor of other in LV)	Step 1: Retire local-only concepts in subsumption path between the two concepts Step 2: Merge two concepts into one of the two concepts
Action 4 (if one concept is non-parent ancestor of other in LV)	Step 1: Retire local-only concepts in subsumption path between the two concepts Step 2: Merge two concepts into one of the two concepts Step 3: Preserve retired concept

Table H 5. Action choices for *merge two concepts into new concept.*

Action	Steps
Action 1	Step 1: Merge two concepts into new concept
Action 2	Step 1: Merge two concepts into new concept Step 2: Preserve first retired concept Step 3: Preserve second retired concept
Action 3	Step 1: Merge two concepts into new concept Step 2: Preserve first retired concept
Action 4	Step 1: Merge two concepts into new concept Step 2: Preserve second retired concept
Action 5 (if one concept is non-parent ancestor of other in LV)	Step 1: Retire local concepts in subsumption path between the two concepts Step 2: Merge two concepts into new concept
Action 6 (if one concept is non-parent ancestor of other in LV)	Step 1: Retire local concepts in subsumption path between the two concepts Step 2: Merge two concepts into new concept Step 3: Preserve first retired concept Step 4: Preserve second retired concept
Action 7 (if one concept is non-parent ancestor of other in LV)	Step 1: Retire local concepts in subsumption path between the two concepts Step 2: Merge two concepts into new concept Step 3: Preserve first retired concept
Action 6 (if one concept is non-parent ancestor of other in LV)	Step 1: Retire local concepts in subsumption path between the two concepts Step 2: Merge two concepts into new concept Step 3: Preserve second retired concept

Table H 6. Action choices for *merge two attributes into one of the two attributes.*

Action	Steps
Action 1	Step 1: Merge two attributes into one of the two attributes
Action 2	Step 1: Merge two attributes into one of the two attributes Step 2: Preserve retired attribute

Table H 7. Action choices for *merge two attributes into new attribute.*

Action	Steps
Action 1	Step 1: Merge two attributes into new attribute
Action 2	Step 1: Merge two attributes into new attribute Step 2: Preserve first retired attribute Step 3: Preserve second retired attribute
Action 3	Step 1: Merge two attributes into new attribute Step 2: Preserve first retired attribute
Action 4	Step 1: Merge two concepts into new attribute Step 2: Preserve second retired attribute

Table H 8. Action choices for *add attribute.*

Action	Steps
Action 1	Step 1: Add attribute
Action 2 (if local name conflicts)	Step 1: Rename local attribute with conflicting name Step 2: Add attribute
Action 3 (if local name conflicts)	Step 1: Rename local attribute with conflicting name Step 2: Add attribute Step 3: Rename new attribute Step 4: Rename local attribute back to original name
Action 4 (if local attribute has same meaning)	Step 1: Add attribute Step 2: Merge local attribute into shared attribute
Action 5 (if local attribute has same meaning)	Step 1: Add attribute Step 2: Merge local attribute into shared attribute Step 3: Rename attribute with local name
Action 6	Step 1: Add attribute Step 2: Hide attribute

Table H 9. Action choices for *retire attribute.*

Action	Steps
Action 1	Step 1: Retire attribute
Action 2	Step 1: Retire attribute Step 2: Preserve attribute

Table H 10. Action choices for *replace concept name.*

Action	Steps
Action 1	Step 1: Change concept name to new name Step 2: Add old name to synonym list
Action 2	Step 1: Change concept name to new name (Step 2: Do not add old name to synonym list)
Action 3 (if new name conflicts with name of an existing local concept)	Step 1: Change name of conflicting local concept Step 2: Change concept name to new name Step 3: Add old name to synonym list
Action 4 (if new name conflicts with name of an existing local concept)	Step 1: Change name of conflicting local concept Step 2: Change concept name to new name (Step 3: Do not add old name to synonym list)
Action 5	(Step 1: Do nothing)

Table H 11. Action choices for *correct concept name.*

Action	Steps
Action 1	Step 1: Change concept name to new name
Action 2 (if new name conflicts with name of an existing local concept)	Step 1: Change name of conflicting local concept Step 2: Change concept name to new name
Action 3	(Step 1: Do nothing)

Table H 12. Action choices for *replace concept definition.*

Action	Steps
Action 1	Step 1: Replace concept definition
Action 2	(Step 1: Do nothing)

Table H 13. Action choices for *replace UMLS code.*

Action	Steps
Action 1	Step 1: Replace UMLS code
Action 2	(Step 1: Do nothing)

Table H 14. Action choices for *add synonym.*

Action	Steps
Action 1	Step 1: Add synonym
Action 2 (if synonym already exists or user preference)	(Step 1: Do nothing)

Table H 15. Action choices for *delete synonym.*

Action	Steps
Action 1	Step 1: Delete synonym
Action 2	(Step 1: Do nothing)

Table H 16. Action choices for *add abbreviation.*

Action	Steps
Action 1	Step 1: Add abbreviation
Action 2	(Step 1: Do nothing)

Table H 17. Action choices for *delete abbreviation*

Action	Steps
Action 1	Step 1: Delete abbreviation
Action 2	(Step 1: Do nothing)

Table H 18. Action choices for *add parent.*

Action	Steps
Action 1	Step 1: Add parent Step 2: Add parent to list of SV parents
Action 2 (if a cycle would result from adding the parent)	Step 1: Break cycle Step 2: Add parent Step 3: Add parent to list of SV parents
Action 3 (if concept gaining new parent is already subsumed by parent)	Step 1: Add parent to list of SV parents

Table H 19. Action choices for *add child.*

Action	Steps
Action 1	Step 1: Add child Step 2: Add child to list of SV children
Action 2 (if a cycle would result from adding the child)	Step 1: Break cycle Step 2: Add child Step 3: Add child to list of SV children
Action 3 (if concept gaining new child already subsumes child)	Step 1: Add child to list of SV children

Table H 20. Action choices for *remove parent.*

Action	Steps
Action 1	Step 1: Remove parent Step 2: Remove parent from list of SV parents
Action 2	Step 1: Remove parent from list of SV parents

Table H 21. Action choices for *remove child.*

Action	Steps
Action 1	Step 1: Remove child Step 2: Remove child from list of SV children
Action 2	Step 1: Remove child from list of SV children

Table H 22. Action choices for *add attribute–value pair.*

Action	Steps
Action 1 (if no conflict)	Step 1: Add attribute–value pair
Action 2 (if conflict exists in ancestor avpair, where value is more specific than new value)	Step 1: Remove conflicting attribute–value pair in ancestor Step 2: Add attribute–value pair
Action 3 (if conflict exists in descendant avpair, where value is more specific than new value)	Step 1: Remove conflicting attribute–value pair in descendant Step 2: Add attribute–value pair

Table H 23. Action choices for *delete attribute–value pair.*

Action	Steps
Action 1	Step 1: Delete attribute–value pair
Action 2	(Step 1: Do nothing)

Table H 24. Action choices for *replace attribute value.*

Action	Steps
Action 1 (if no conflict)	Step 1: Replace attribute value
Action 1 (if conflict exists in ancestor avpair, where value is more specific than new value)	Step 1: Remove conflicting attribute–value pair in ancestor Step 2: Replace attribute value
Action 1 (if conflict exists in descendant avpair, where value is more general than new value)	Step 1: Remove conflicting attribute–value pair in descendant Step 2: Replace attribute value

Appendix I: Changes Made to Shared Vocabulary

Changes made to the shared vocabulary, SV-0, to create the modified shared vocabulary, SV-1. See Figures 6.1 and 6.2 for SV-0 and SV-1, respectively.

1. Add concept: CONCEPT rickettsial disease, PARENT disease
2. Add concept: CONCEPT mite-borne spotted fever, PARENT rickettsial disease
3. Add concept: CONCEPT tick-borne spotted fever, PARENT rickettsial disease
4. Add concept: CONCEPT flea-borne rickettsial disease, PARENT rickettsial disease
5. Add concept: CONCEPT louse-borne rickettsial disease, PARENT rickettsial disease
6. Add concept: CONCEPT ehrlichiosis, PARENT rickettsial disease
7. Add concept: CONCEPT Q fever, PARENT rickettsial disease
8. Replace concept name: OLD tsutsugamushi, NEW scrub typhus
9. Add parent: CONCEPT scrub typhus, PARENT rickettsial disease
10. Remove parent: CONCEPT scrub typhus, PARENT tropical disease of disputed nature or minor importance
11. Replace concept name: OLD spotted fever of the Rocky Mountains, NEW Rocky Mountain spotted fever
12. Add parent: CONCEPT Rocky Mountain spotted fever, PARENT tick-borne spotted fever
13. Remove parent: CONCEPT Rocky Mountain spotted fever, PARENT tropical disease of disputed nature or importance
14. Add concept: CONCEPT Mediterranean spotted fever, PARENT tick-borne spotted fever
15. Add concept: CONCEPT African tick-bite fever, PARENT tick-borne spotted fever
16. Add concept: CONCEPT Japanese spotted fever, PARENT tick-borne spotted fever
17. Add concept: CONCEPT Queensland tick typhus, PARENT tick-borne spotted fever
18. Add concept: CONCEPT Flinders Island spotted fever, PARENT tick-borne spotted fever
19. Add concept: CONCEPT rickettsialpox, PARENT mite-borne spotted fever
20. Replace concept name: OLD typhus fever, NEW epidemic typhus
21. Add parent: CONCEPT epidemic typhus, PARENT louse-borne rickettsial disease
22. Add concept: CONCEPT Brill-Zinsser disease, PARENT epidemic typhus
23. Add concept: CONCEPT murine typhus, PARENT flea-borne rickettsial disease
24. Remove parent: CONCEPT epidemic typhus, PARENT tropical disease of disputed nature or minor importance
25. Add concept: CONCEPT human monocytic ehrlichiosis, PARENT ehrlichiosis
26. Add concept: CONCEPT human granulocytic ehrlichiosis, PARENT ehrlichiosis
27. Add parent: CONCEPT verruga peruviana, PARENT disease
28. Remove parent: CONCEPT verruga peruviana, PARENT tropical disease of disputed nature or minor importance

29. Add parent: CONCEPT heat stroke, PARENT disease
30. Remove parent: CONCEPT heat stroke, PARENT tropical disease of disputed nature or minor importance
31. Retire concept: CONCEPT tropical disease of disputed nature or minor importance
32. Add concept: CONCEPT living organism PARENT entity
33. Add concept: CONCEPT tick PARENT living organism
34. Add concept: CONCEPT mite PARENT living organism
35. Add concept: CONCEPT louse PARENT living organism
36. Add concept: CONCEPT chigger PARENT living organism
37. Add concept: CONCEPT Rickettsia, PARENT living organism
38. Add concept: CONCEPT Rickettsia rickettsii, PARENT Rickettsia
39. Add concept: CONCEPT Rickettsia conorii, PARENT Rickettsia
40. Add concept: CONCEPT Dermacentor variabilis, PARENT tick
41. Add concept: CONCEPT Dermacentor andersoni, PARENT tick
42. Add concept: CONCEPT Rhipicephalus sanguineus, PARENT tick
43. Add concept: CONCEPT Amblyomma cajennesne, PARENT tick
44. Add concept: CONCEPT Rickettsia japonica, PARENT Rickettsia
45. Add concept: CONCEPT Rickettsia australis, PARENT Rickettsia
46. Add concept: CONCEPT Rickettsia honei, PARENT Rickettsia
47. Add concept: CONCEPT Rickettsia akari, PARENT Rickettsia
48. Add concept: CONCEPT Rickettsia typhi, PARENT Rickettsia
49. Add concept: CONCEPT Rickettsia prowazekii, PARENT Rickettsia
50. Add concept: CONCEPT flea, PARENT living organism
51. Add concept: CONCEPT Xenopsylla cheopis, PARENT flea
52. Add concept: CONCEPT Pediculus humanus corporis, PARENT louse
53. Add concept: CONCEPT Orientia, PARENT living organism
54. Add concept: CONCEPT Orientia tsutsugamushi, PARENT Orientia
55. Add concept: CONCEPT Ctenocephalides felis, PARENT flea
56. Add concept: CONCEPT Ehrlichia, PARENT living organism
57. Add concept: CONCEPT Ehrlichia chaffeensis, PARENT Ehrlichia
58. Add concept: CONCEPT agent of human granulocytic ehrlichiosis, PARENT Ehrlichia
59. Add concept: CONCEPT Amblyomma americanum, PARENT tick
60. Add concept: CONCEPT Ixodes scapularis, PARENT tick
61. Add concept: CONCEPT Ixodes ricinus, PARENT tick
62. Add concept: CONCEPT Coxiella, PARENT living organism
63. Add concept: CONCEPT Coxiella burnettii, PARENT Coxiella

64. Add attribute: ATTRIBUTE has-etiology

65. Add attribute: ATTRIBUTE transmitted-by

66. Add attribute–value pair: CONCEPT tick-borne spotted fever, ATTRIBUTE transmitted-by VALUE tick

67. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever ATTRIBUTE transmitted-by VALUE Dermacentor variabilis

68. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever ATTRIBUTE has-etiology VALUE Rickettsia rickettsii

69. Add synonym: CONCEPT Dermacentor variabilis SYNONYM American dog tick

70. Add abbreviation: CONCEPT Rocky Mountain spotted fever ABBREVIATION RMSF

71. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever ATTRIBUTE transmitted-by VALUE Dermacentor andersoni

72. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever ATTRIBUTE transmitted-by VALUE Rhipicephalus sanguineus

73. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever ATTRIBUTE transmitted-by VALUE Amblyomma cajennesne

74. Add attribute–value pair: CONCEPT Mediterranean spotted fever ATTRIBUTE transmitted-by VALUE Rhipicephalus sanguineus

75. Add attribute–value pair: CONCEPT Mediterranean spotted fever ATTRIBUTE has-etiology VALUE Rickettsia conorii

76. Add synonym: CONCEPT Mediterranean spotted fever SYNONYM boutonneuse fever

77. Add synonym: CONCEPT Mediterranean spotted fever SYNONYM Kenya tick typhus

78. Add synonym: CONCEPT Mediterranean spotted fever SYNONYM Indian tick typhus

79. Add synonym: CONCEPT Mediterranean spotted fever SYNONYM Israeli spotted fever

80. Add synonym: CONCEPT Mediterranean spotted fever SYNONYM Astrakhan spotted fever

81. Add concept: CONCEPT Rickettsia africae PARENT Rickettsia

82. Add attribute–value pair: CONCEPT African tick-bite fever ATTRIBUTE has-etiology VALUE Rickettsia africae

83. Add attribute–value pair: CONCEPT Japanese spotted fever ATTRIBUTE has-etiology VALUE Rickettsia japonica

84. Add synonym: CONCEPT Japanese spotted fever SYNONYM Oriental spotted fever

85. Add attribute–value pair: CONCEPT Queensland tick typhus ATTRIBUTE has-etiology VALUE Rickettsia australis

86. Add concept: CONCEPT Ixodes holocyclus, PARENT tick

87. Add parent: CONCEPT Ixodes holocyclus PARENT tick

88. Remove parent: CONCEPT Ixodes holocyclus PARENT Dermacentor variabilis

89. Add attribute–value pair: CONCEPT Flinders Island spotted fever ATTRIBUTE has-etiology VALUE Rickettsia honei

90. Add attribute–value pair: CONCEPT Queensland tick typhus ATTRIBUTE transmitted-by VALUE Ixodes holocyclus

91. Add attribute–value pair: CONCEPT mite-borne spotted fever ATTRIBUTE transmitted-by VALUE mite

92. Add attribute–value pair: CONCEPT rickettsialpox ATTRIBUTE has-etiology VALUE Rickettsia akari

93. Add attribute–value pair: CONCEPT flea-borne rickettsial disease ATTRIBUTE transmitted-by VALUE flea

94. Add attribute–value pair: CONCEPT murine typhus ATTRIBUTE has-etiology VALUE Rickettsia typhi

95. Add attribute–value pair: CONCEPT murine typhus ATTRIBUTE transmitted-by VALUE Xenopsylla cheopis

96. Add attribute–value pair: CONCEPT murine typhus ATTRIBUTE transmitted-by VALUE Ctenocephalides felis

97. Add synonym: CONCEPT murine typhus SYNONYM endemic typhus

98. Add synonym: CONCEPT murine typhus SYNONYM human murine typhus

99. Add synonym: CONCEPT Xenopsylla cheopis SYNONYM Oriental rat flea

100. Add synonym: CONCEPT Ctenocephalides felis SYNONYM cat flea

101. Add attribute–value pair: CONCEPT louse-borne rickettsial disease ATTRIBUTE transmitted-by VALUE louse

102. Add attribute–value pair: CONCEPT epidemic typhus ATTRIBUTE has-etiology VALUE Rickettsia prowazekii

103. Add attribute–value pair: CONCEPT epidemic typhus ATTRIBUTE transmitted-by VALUE Pediculus humanus corporis

104. Add synonym: CONCEPT epidemic typhus SYNONYM epidemic louse-borne typhus

105. Add synonym: CONCEPT Pediculus humanus corporis SYNONYM human body louse

106. Add synonym: CONCEPT Brill-Zinsser disease SYNONYM recrudescent typhus

107. Add attribute–value pair: CONCEPT scrub typhus ATTRIBUTE has-etiology VALUE Orientia tsutsugamushi

108. Add attribute–value pair: CONCEPT scrub typhus ATTRIBUTE transmitted-by VALUE chigger

109. Add attribute–value pair: CONCEPT Q fever ATTRIBUTE has-etiology VALUE Coxiella burnettii

110. Add attribute–value pair: CONCEPT ehrlichiosis ATTRIBUTE has-etiology VALUE Ehrlichia

111. Add attribute–value pair: CONCEPT human monocytic ehrlichiosis ATTRIBUTE has-etiology VALUE Ehrlichia chaffeensis

112. Add attribute–value pair: CONCEPT human monocytic ehrlichiosis ATTRIBUTE transmitted-by VALUE Amblyomma americanum

113. Add attribute–value pair: CONCEPT human monocytic ehrlichiosis ATTRIBUTE transmitted-by VALUE Dermacentor variabilis

114. Add attribute: ATTRIBUTE major-target-cell

115. Add concept: CONCEPT anatomic part PARENT entity

116. Add concept: CONCEPT cell PARENT anatomic part

117. Add concept: CONCEPT monocyte PARENT cell

118. Add concept: CONCEPT granulocyte PARENT cell

119. Add abbreviation: CONCEPT human monocytic ehrlichiosis ABBREVIATION HME

120. Add attribute–value pair: CONCEPT human monocytic ehrlichiosis ATTRIBUTE has-etiology VALUE Ehrlichia chaffeensis

121. Add attribute–value pair: CONCEPT human monocytic ehrlichiosis ATTRIBUTE major-target-cell VALUE monocyte

122. Add attribute–value pair: CONCEPT human granulocytic ehrlichiosis ATTRIBUTE has-etiology VALUE agent of human granulocytic ehrlichiosis

123. Add abbreviation: CONCEPT human granulocytic ehrlichiosis ABBREVIATION HGE

124. Add attribute–value pair: CONCEPT human granulocytic ehrlichiosis ATTRIBUTE transmitted-by VALUE Ixodes scapularis

125. Add attribute–value pair: CONCEPT human granulocytic ehrlichiosis ATTRIBUTE transmitted-by VALUE Ixodes ricinus

126. Add attribute–value pair: CONCEPT human granulocytic ehrlichiosis ATTRIBUTE major-target-cell VALUE granulocyte

127. Add synonym: CONCEPT Ixodes scapularis SYNONYM deer tick

128. Add synonym: CONCEPT Dermacentor variabilis SYNONYM dog tick

129. Add synonym: CONCEPT Amblyomma americanum SYNONYM Lone Star tick

Appendix J: Changes Made to Local Vocabulary

Changes made to the local vocabulary, LV-0, to create the modified local vocabulary, LV-1. See Figures 6.1 and 6.3 for LV-0 and LV-1, respectively.

1. Add concept: CONCEPT rickettsial disease, PARENT disease
2. Add concept: CONCEPT typhus-like fever, PARENT rickettsial disease
3. Add concept: CONCEPT typhus-group disease, PARENT typhus-like fever
4. Add concept: CONCEPT murine typhus, PARENT typhus-group disease
5. Add concept: CONCEPT louse-borne epidemic typhus, PARENT typhus-group disease
6. Add concept: CONCEPT Brill-Zinsser disease, PARENT louse-borne epidemic typhus
7. Add concept: CONCEPT scrub typhus, PARENT typhus-like fever
8. Add concept: CONCEPT spotted-fever-group disease, PARENT typhus-like fever
9. Add concept: CONCEPT ehrlichiosis, PARENT spotted-fever-group disease
10. Replace concept name: OLD spotted fever of the Rocky Mountains, NEW Rocky Mountain spotted fever
11. Add parent: CONCEPT Rocky Mountain spotted fever, PARENT spotted-fever-group disease
12. Remove parent: CONCEPT Rocky Mountain spotted fever, PARENT tropical disease of disputed nature or minor importance
13. Add concept: CONCEPT boutonneuse fever, PARENT spotted-fever-group disease
14. Add concept: CONCEPT rickettsialpox, PARENT rickettsial disease
15. Add concept: CONCEPT Q fever, PARENT rickettsial disease
16. Add concept: CONCEPT tick-borne rickettsiosis, PARENT rickettsial disease
17. Add concept: CONCEPT North Asian tick-borne rickettsiosis, PARENT tick-borne rickettsiosis
18. Add concept: CONCEPT Queensland tick typhus, PARENT tick-borne rickettsiosis
19. Add parent: CONCEPT Rocky Mountain spotted fever, PARENT tick-borne rickettsiosis
20. Add parent: CONCEPT boutonneuse fever, PARENT tick-borne rickettsiosis
21. Add parent: CONCEPT ehrlichiosis, PARENT tick-borne rickettsiosis
22. Add concept: CONCEPT organism, PARENT entity
23. Add concept: CONCEPT Rickettsia, PARENT organism
24. Add concept: CONCEPT Rickettsia typhi, PARENT Rickettsia
25. Add concept: CONCEPT Rickettsia prowazekii, PARENT Rickettsia
26. Add concept: CONCEPT Ehrlichia, PARENT organism
27. Add concept: CONCEPT Ehrlichia chaffeensis, PARENT Ehrlichia

28. Add concept: CONCEPT Ehrlichia species of human granulocytic ehrlichiosis, PARENT Ehrlichia
29. Add concept: CONCEPT Rickettsia siberica, PARENT Rickettsia
30. Add concept: CONCEPT Rickettsia akari, PARENT Rickettsia
31. Add concept: CONCEPT Rickettsia tsutsugamushi, PARENT Rickettsia
32. Add synonym: CONCEPT Rickettsia tsutsugamushi, SYNONYM Rickettsia orientalis
33. Add concept: CONCEPT Coxiella, PARENT organism
34. Add concept: CONCEPT Coxiella burnettii, PARENT Coxiella
35. Add concept: CONCEPT Amblyomma, PARENT organism
36. Add concept: CONCEPT Xenopsylla, PARENT organism
37. Add concept: CONCEPT Xenopsylla cheopis, PARENT Xenopsylla
38. Add synonym: CONCEPT Xenopsylla cheopis, SYNONYM rat flea
39. Add concept: CONCEPT Pediculus, PARENT organism
40. Add concept: CONCEPT Pediculus humanus humanus, PARENT Pediculus
41. Add synonym: CONCEPT Pediculus humanus humanus, SYNONYM human body louse
42. Add synonym: CONCEPT Pediculus humanus humanus, SYNONYM body louse
43. Add attribute: ATTRIBUTE etiology
44. Add attribute–value pair: CONCEPT murine typhus, ATTRIBUTE etiology, VALUE Rickettsia typhi
45. Add attribute: ATTRIBUTE vector
46. Add attribute–value pair: CONCEPT murine typhus, ATTRIBUTE vector, VALUE Xenopsylla cheopis
47. Add concept: CONCEPT Rickettsia prowazekii, PARENT Rickettsia
48. Add attribute–value pair: CONCEPT louse-borne epidemic typhus, ATTRIBUTE etiology, VALUE Rickettsia prowazekii
49. Add attribute–value pair: CONCEPT louse-borne epidemic typhus, ATTRIBUTE vector, VALUE Pediculus humanus humanus
50. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM epidemic typhus
51. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM louse-borne typhus
52. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM classic typhus
53. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM classic typhus fever
54. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM European fever
55. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM jail fever
56. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM war fever
57. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM camp fever
58. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM ship fever
59. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM typhus exanthematique

60. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM tifus exantematico
61. Add synonym: CONCEPT louse-borne epidemic typhus, SYNONYM tifus tabardillo
62. Add concept: CONCEPT Dermacentor, PARENT organism
63. Add concept: CONCEPT Dermacentor variabilis, PARENT Dermacentor
64. Add synonym: CONCEPT Dermacentor variabilis, SYNONYM dog tick
65. Add concept: CONCEPT Dermacentor andersoni, PARENT Dermacentor
66. Add concept: CONCEPT Amblyomma americanum, PARENT Amblyomma
67. Add synonym: CONCEPT Dermacentor andersoni, SYNONYM wood tick
68. Add concept: CONCEPT Dermacentor sylvarum, PARENT Dermacentor
69. Add concept: CONCEPT Dermacentor nuttallii, PARENT Dermacentor
70. Add concept: CONCEPT Leptotrombidium, PARENT organism
71. Add concept: CONCEPT Leptotrombidium deliensis, PARENT Leptotrombidium
72. Add synonym: CONCEPT Leptotrombidium deliensis, SYNONYM chigger
73. Add concept: CONCEPT Ixodes, PARENT organism
74. Add concept: CONCEPT Ixodes holocyclus, PARENT Ixodes
75. Add concept: CONCEPT Allodermanyssus, PARENT organism
76. Add concept: CONCEPT Allodermanyssus sanguineus, PARENT Allodermanyssus
77. Add synonym: CONCEPT Allodermanyssus sanguineus, SYNONYM mite
78. Add concept: CONCEPT Rickettsia rickettsii, PARENT Rickettsia
79. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever, ATTRIBUTE etiology, VALUE Rickettsia rickettsii
80. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever, ATTRIBUTE vector, VALUE Ixodes
81. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever, ATTRIBUTE vector, VALUE Dermacentor andersoni
82. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever, ATTRIBUTE vector, VALUE Dermacentor variabilis
83. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever, ATTRIBUTE vector, VALUE Amblyomma americanum
84. Add concept: CONCEPT Rhipicephalus, PARENT organism
85. Add concept: CONCEPT Rhipicephalus sanguineus, PARENT Rhipicephalus
86. Add attribute–value pair: CONCEPT Rocky Mountain spotted fever, ATTRIBUTE vector, VALUE Rhipicephalus sanguineus
87. Add synonym: CONCEPT ehrlichiosis, SYNONYM spotless Rocky Mountain spotted fever
88. Add attribute–value pair: CONCEPT ehrlichiosis, ATTRIBUTE etiology, VALUE Ehrlichia
89. Add concept: CONCEPT ehrlichiosis caused by Ehrlichia chaffeensis, PARENT Ehrlichiosis

90. Add attribute–value pair: CONCEPT ehrlichiosis caused by Ehrlichia chaffeensis, ATTRIBUTE etiology, VALUE Ehrlichia chaffeensis

91. Add attribute–value pair: CONCEPT ehrlichiosis caused by Ehrlichia chaffeensis, ATTRIBUTE vector, VALUE Dermacentor variabilis

92. Add concept: CONCEPT human granulocytic ehrlichiosis, PARENT ehrlichiosis

93. Add attribute–value pair: CONCEPT human granulocytic ehrlichiosis, ATTRIBUTE etiology, VALUE Ehrlichia species of human granulocytic ehrlichiosis

94. Add abbreviation: CONCEPT human granulocytic ehrlichiosis, ABBREVIATION HGE

95. Add concept: CONCEPT Rickettsia conorii, PARENT Rickettsia

96. Add attribute–value pair: CONCEPT boutonneuse fever, ATTRIBUTE etiology, VALUE Rickettsia conorii

97. Add attribute–value pair: CONCEPT boutonneuse fever, ATTRIBUTE vector, VALUE Rhipicephalus sanguineus

98. Add synonym: CONCEPT boutonneuse fever, SYNONYM Mediterranean spotted fever

99. Add synonym: CONCEPT boutonneuse fever, SYNONYM North African tick typhus

100. Add synonym: CONCEPT boutonneuse fever, SYNONYM Kenya tick-bite fever

101. Add synonym: CONCEPT boutonneuse fever, SYNONYM Indian tick typhus

102. Add attribute–value pair: CONCEPT North Asian tick-borne rickettsiosis, ATTRIBUTE etiology, VALUE Rickettsia siberica

103. Add concept: CONCEPT Haemaphysalis, PARENT organism

104. Add concept: CONCEPT Haemaphysalis concinna, PARENT Haemaphysalis

105. Add attribute–value pair: CONCEPT North Asian tick-borne rickettsiosis, ATTRIBUTE vector, VALUE Haemaphysalis concinna

106. Add attribute–value pair: CONCEPT North Asian tick-borne rickettsiosis, ATTRIBUTE vector, VALUE Dermacentor sylvarum

107. Add attribute–value pair: CONCEPT North Asian tick-borne rickettsiosis, ATTRIBUTE vector, VALUE Dermacentor nuttallii

108. Add concept: CONCEPT Rickettsia australis, PARENT Rickettsia

109. Add attribute–value pair: CONCEPT Queensland tick typhus, ATTRIBUTE etiology, VALUE Rickettsia australis

110. Add attribute–value pair: CONCEPT Queensland tick typhus, ATTRIBUTE vector, VALUE Ixodes holocyclus

111. Add attribute–value pair: CONCEPT rickettsialpox, ATTRIBUTE etiology, VALUE Rickettsia akari

112. Add attribute–value pair: CONCEPT rickettsialpox, ATTRIBUTE vector, VALUE Allodermanyssus sanguineus

113. Add attribute–value pair: CONCEPT scrub typhus, ATTRIBUTE etiology, VALUE Rickettsia tsutsugamushi

114. Add attribute–value pair: CONCEPT scrub typhus, ATTRIBUTE vector, VALUE Leptotrombidium deliensis

115. Add attribute–value pair: CONCEPT Q fever, ATTRIBUTE etiology, VALUE Coxiella burnettii

116. Merge 2 concepts into one of the 2 concepts: CONCEPT1 tsutsugamushi, CONCEPT 2 scrub typhus, KEEP tsutsugamushi, RETIRE scrub typhus

117. Merge 2 concepts into one of the 2 concepts: CONCEPT1 typhus fever, CONCEPT 2 louse-borne epidemic typhus, KEEP typhus fever, RETIRE louse-borne epidemic typhus

118. Replace concept name: OLD typhus fever, NEW louse-borne epidemic typhus

119. Replace concept name: OLD tsutsugamushi, NEW scrub typhus

Appendix K: Synchronization Report

```
REPORT OF CHANGES PERFORMED DURING SYNCHRONIZATION

DATE OF REPORT: Mon Dec 13 16:37:04 PST 1999
AUTHOR OF SYNCHRONIZATION: Diane E. Oliver

FILE NAMES:

Local vocabulary (input): E:\Development\Demo\Save Items
Local\LocalVocabTimeI
Shared vocabulary (input): E:\Development\Demo\Save
Items\SharedVocabTimeF
Local log (input): E:\Development\Demo\Save Items Local\LocalLogAToI
Shared log (input): E:\Development\Demo\Save Items\SharedLogAToF
Synchronized local vocabulary (output):
E:\Development\Demo\LocalVocabTime1Synch
Synchronization local log (output): E:\Development\Demo\LocalLogSynch
This report: E:\Development\Demo\AToFSynchReport.txt

LIST OF CHANGES:

Add concept: "rickettsial disease" with parent = "disease"
Merge 2 concepts into one of the 2 concepts: "rickettsial disease" into
"rickettsial disease"
Add concept: "mite-borne spotted fever" with parent = "rickettsial
disease"
Add concept: "tick-borne spotted fever" with parent = "rickettsial
disease"
Add concept: "flea-borne rickettsial disease" with parent =
"rickettsial disease"
Add concept: "louse-borne rickettsial disease" with parent =
"rickettsial disease"
Add concept: "ehrlichiosis" with parent = "rickettsial disease"
Merge 2 concepts into one of the 2 concepts: "ehrlichiosis" into
"ehrlichiosis"
Add concept: "Q fever" with parent = "rickettsial disease"
Merge 2 concepts into one of the 2 concepts: "Q fever" into "Q fever"
(Concept: scrub typhus) Add parent: "rickettsial disease"
(Concept: scrub typhus) Remove parent: "tropical disease of disputed
nature or minor importance"
(Concept: Rocky Mountain spotted fever) Add parent: "tick-borne spotted
fever"
Add concept: "Mediterranean spotted fever" with parent = "tick-borne
spotted fever"
Merge 2 concepts into one of the 2 concepts: "boutonneuse fever" into
"Mediterranean spotted fever"
Add concept: "African tick-bite fever" with parent = "tick-borne
spotted fever"
Add concept: "Japanese spotted fever" with parent = "tick-borne spotted
fever"
Add concept: "Queensland tick typhus" with parent = "tick-borne spotted
fever"
Merge 2 concepts into one of the 2 concepts: "Queensland tick typhus"
into "Queensland tick typhus"
```

Add concept: "Flinder’s Island spotted fever" with parent = "tick-borne spotted fever"
Add concept: "rickettsialpox" with parent = "mite-borne spotted fever"
Merge 2 concepts into one of the 2 concepts: "rickettsialpox" into "rickettsialpox"
(Concept: louse-borne epidemic typhus) Correct concept name: "louse-borne epidemic typhus" with "epidemic typhus"
(Concept: epidemic typhus) Add parent: "louse-borne rickettsial disease"
Add concept: "Brill-Zinsser disease" with parent = "epidemic typhus"
Merge 2 concepts into one of the 2 concepts: "Brill-Zinsser disease" into "Brill-Zinsser disease"
Add concept: "murine typhus" with parent = "flea-borne rickettsial disease"
Merge 2 concepts into one of the 2 concepts: "murine typhus" into "murine typhus"
Add concept: "human monocytic ehrlichiosis" with parent = "ehrlichiosis"
Add concept: "human granulocytic ehrlichiosis" with parent = "ehrlichiosis"
Merge 2 concepts into one of the 2 concepts: "human granulocytic ehrlichiosis" into "human granulocytic ehrlichiosis"
(Concept: verruga peruviana) Add parent: "disease"
(Concept: verruga peruviana) Remove parent: "tropical disease of disputed nature or minor importance"
(Concept: heat stroke) Add parent: "disease"
(Concept: heat stroke) Remove parent: "tropical disease of disputed nature or minor importance"
Retire concept: "tropical disease of disputed nature or minor importance"
(Concept: epidemic typhus) Remove parent: "disease"
Add concept: "living organism" with parent = "entity"
Merge 2 concepts into one of the 2 concepts: "organism" into "living organism"
Add concept: "tick" with parent = "living organism"
Add concept: "mite" with parent = "living organism"
Add concept: "louse" with parent = "living organism"
Add concept: "chigger" with parent = "living organism"
Add concept: "Rickettsia" with parent = "living organism"
Merge 2 concepts into one of the 2 concepts: "Rickettsia" into "Rickettsia"
Add concept: "Rickettsia rickettsii" with parent = "Rickettsia"
Merge 2 concepts into one of the 2 concepts: "Rickettsia rickettsii" into "Rickettsia rickettsii"
Add concept: "Rickettsia conorii" with parent = "Rickettsia"
Merge 2 concepts into one of the 2 concepts: "Rickettsia conorii" into "Rickettsia conorii"
Add concept: "Dermacentor variabilis" with parent = "tick"
Merge 2 concepts into one of the 2 concepts: "Dermacentor variabilis" into "Dermacentor variabilis"
Add concept: "Dermacentor andersoni" with parent = "tick"
Merge 2 concepts into one of the 2 concepts: "Dermacentor andersoni" into "Dermacentor andersoni"
Add concept: "Rhipicephalus sanguineus" with parent = "tick"
Merge 2 concepts into one of the 2 concepts: "Rhipicephalus sanguineus" into "Rhipicephalus sanguineus"
Add concept: "Amblyomma cajennesne" with parent = "tick"

Add concept: "Rickettsia japonica" with parent = "Rickettsia"
Add concept: "Rickettsia australis" with parent = "Rickettsia"
Merge 2 concepts into one of the 2 concepts: "Rickettsia australis" into "Rickettsia australis"
Add concept: "Rickettsia honei" with parent = "Rickettsia"
Add concept: "Rickettsia akari" with parent = "Rickettsia"
Merge 2 concepts into one of the 2 concepts: "Rickettsia akari" into "Rickettsia akari"
Add concept: "Rickettsia typhi" with parent = "Rickettsia"
Merge 2 concepts into one of the 2 concepts: "Rickettsia typhi" into "Rickettsia typhi"
Add concept: "Rickettsia prowazekii" with parent = "Rickettsia"
Merge 2 concepts into one of the 2 concepts: "Rickettsia prowazekii" into "Rickettsia prowazekii"
Add concept: "flea" with parent = "living organism"
Add concept: "Xenopsylla cheopis" with parent = "flea"
Merge 2 concepts into one of the 2 concepts: "Xenopsylla cheopis" into "Xenopsylla cheopis"
Add concept: "Pediculus humanus corporis" with parent = "louse"
Merge 2 concepts into one of the 2 concepts: "Pediculus humanus humanus" into "Pediculus humanus corporis"
Add concept: "Orientia" with parent = "living organism"
Add concept: "Orientia tsutsugamushi" with parent = "Orientia"
Merge 2 concepts into one of the 2 concepts: "Rickettsia tsutsugamushi" into "Orientia tsutsugamushi"
Add concept: "Ctenocephalides felis" with parent = "flea"
Add concept: "Ehrlichia" with parent = "living organism"
Merge 2 concepts into one of the 2 concepts: "Ehrlichia" into "Ehrlichia"
Add concept: "Ehrlichia chaffeensis" with parent = "Ehrlichia"
Merge 2 concepts into one of the 2 concepts: "Ehrlichia chaffeensis" into "Ehrlichia chaffeensis"
Add concept: "agent of human granulocytic ehrlichiosis" with parent = "Ehrlichia"
Merge 2 concepts into one of the 2 concepts: "Ehrlichia species of human granulocytic ehrlichiosis" into "agent of human granulocytic ehrlichiosis"
Add concept: "Amblyomma americanum" with parent = "tick"
Merge 2 concepts into one of the 2 concepts: "Amblyomma americanum" into "Amblyomma americanum"
Add concept: "Ixodes scapularis" with parent = "tick"
Add concept: "Ixodes ricinus" with parent = "tick"
Add concept: "Coxiella" with parent = "living organism"
Merge 2 concepts into one of the 2 concepts: "Coxiella" into "Coxiella"
Add concept: "Coxiella burnetii" with parent = "Coxiella"
Merge 2 concepts into one of the 2 concepts: "Coxiella burnetii" into "Coxiella burnetii"
Add attribute: "has-etiology"
Merge 2 attributes into one of the 2 attributes: "etiology" into "has-etiology"
Add attribute: "transmitted-by"
Merge 2 attributes into one of the 2 attributes: "vector" into "transmitted-by"
(Concept: tick-borne spotted fever) Add attribute-value pair: "transmitted-by: tick"
(Concept: Dermacentor variabilis) Add synonym: "American dog tick"
(Concept: Rocky Mountain spotted fever) Add abbreviation: "RMSF"

(Concept: Rocky Mountain spotted fever) Add attribute-value pair: "transmitted-by: Amblyomma cajennesne"
(Concept: Mediterranean spotted fever) Add synonym: "Kenya tick typhus"
(Concept: Mediterranean spotted fever) Add synonym: "Israeli spotted fever"
(Concept: Mediterranean spotted fever) Add synonym: "Astrakhan spotted fever"
Add concept: "Rickettsia africae" with parent = "Rickettsia"
(Concept: African tick-bite fever) Add attribute-value pair: "has-etiology: Rickettsia africae"
(Concept: Japanese spotted fever) Add attribute-value pair: "has-etiology: Rickettsia japonica"
(Concept: Japanese spotted fever) Add synonym: "Oriental spotted fever"
Add concept: "Ixodes holocyclus" with parent = "tick"
Merge 2 concepts into one of the 2 concepts: "Ixodes holocyclus" into "Ixodes holocyclus"
(Concept: Ixodes holocyclus) Add parent: "tick"
(Concept: Flinder’s Island spotted fever) Add attribute-value pair: "has-etiology: Rickettsia honei"
(Concept: mite-borne spotted fever) Add attribute-value pair: "transmitted-by: mite"
(Concept: flea-borne rickettsial disease) Add attribute-value pair: "transmitted-by: flea"
(Concept: murine typhus) Add attribute-value pair: "transmitted-by: Ctenocephalides felis"
(Concept: murine typhus) Add synonym: "endemic typhus"
(Concept: murine typhus) Add synonym: "human murine typhus"
(Concept: Xenopsylla cheopis) Add synonym: "Oriental rat flea"
(Concept: Ctenocephalides felis) Add synonym: "cat flea"
(Concept: louse-borne rickettsial disease) Add attribute-value pair: "transmitted-by: louse"
(Concept: epidemic typhus) Add synonym: "epidemic louse-borne typhus"
(Concept: Brill-Zinsser disease) Add synonym: "recrudescent typhus"
(Concept: scrub typhus) Add attribute-value pair: "transmitted-by: chigger"
(Concept: human monocytic ehrlichiosis) Add attribute-value pair: "has-etiology: Ehrlichia chaffeensis"
(Concept: human monocytic ehrlichiosis) Add attribute-value pair: "transmitted-by: Amblyomma americanum"
(Concept: human monocytic ehrlichiosis) Add attribute-value pair: "transmitted-by: Dermacentor variabilis"
Add attribute: "major-target-cell"
Add concept: "anatomic part" with parent = "entity"
Add concept: "cell" with parent = "anatomic part"
Add concept: "monocyte" with parent = "cell"
Add concept: "granulocyte" with parent = "cell"
(Concept: human monocytic ehrlichiosis) Add abbreviation: "HME"
(Concept: human monocytic ehrlichiosis) Add attribute-value pair: "major-target-cell: monocyte"
(Concept: human granulocytic ehrlichiosis) Add attribute-value pair: "transmitted-by: Ixodes scapularis"
(Concept: human granulocytic ehrlichiosis) Add attribute-value pair: "transmitted-by: Ixodes ricinus"
(Concept: human granulocytic ehrlichiosis) Add attribute-value pair: "major-target-cell: granulocyte"
(Concept: Ixodes scapularis) Add synonym: "deer tick"(Concept: Amblyomma americanum)

```
Add synonym: "Lone Star tick"
```

References

1. AMIA (American Medical Informatics Association). Standards for medical identifiers, codes, and messages needed to create an efficient computer-stored medical record. *Journal of the American Medical Informatics Association* 1994;1(1):1–7.

2. Bailey PS, Read J. Software implementation of clinical terminologies: The use of component technology (tutorial), AMIA '99 Annual Symposium. Washington, DC; 1999.

3. Baorto DM, Cimino JJ, Parvin CA, Kahn MG. Using Logical Observation Identifier Names and Codes (LOINC) to exchange laboratory data among three academic hospitals. In: Masys DR, editor. *Proceedings of the 1997 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1997. p. 96–100.

4. Barnett G, Zielstorff R, Piggins J, McLatchey J, Morgan M, Barrett S, et al. COSTAR: A comprehensive medical information system for ambulatory care. In: Blum BI, editor. *Proceedings of the Sixth Annual Symposium on Computer Applications in Medical Care*; Washington, DC: The Institute of Electrical and Electronics Engineers; 1982. p. 8–18.

5. Bechhofer S. GALEN Documentation C1: GRAIL Frequently asked questions: University of Manchester, October 1994.

6. Bennett JC, Plum F, editors. *Cecil Textbook of Medicine*. 20th edition. Philadelphia: WB Saunders; 1996.

7. Bernauer J. Subsumption principles underlying medical concept systems and their formal reconstruction. In: Ozbolt JG, editor. *Proceedings of the 1994 Annual Symposium on Computer Applications in Medical Care*; Philadelphia: Hanley & Belfus; 1994. p. 140–144.

8. Bernauer J. Formal classification of medical concept descriptions: Graph-oriented operators. *Methods of Information in Medicine* 1998;37(4–5):510–517.

9. Brachman R, Schmolze J. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 1985;9:171–216.

10. Brachman RJ, Fikes R, Levesque H. Krypton: A functional approach to knowledge representation. *Computer* 1983;16(10):67–73.

11. Brachman RJ, McGuinness DL, Patel-Schneider PF, Resnick LA, Borgida A. Living with CLASSIC: When and how to use a KL-ONE-like language. In: Sowa JF, editor. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. San Mateo, CA: Morgan Kaufmann; 1991. p. 401–456.

12. Brown P, O'Neil M, Price C. Semantic definition of disorders in version 3 of the Read codes. *Methods of Information in Medicine* 1998;37(4–5):415–419.

13. Campbell K, Cohn S, Chute C, Rennels G, Shortliffe E. Galapagos: Computer-based support for evolution of a convergent medical terminology. In: Cimino JJ, editor. *Proceedings of the 1996 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1996. p. 269–273.

14. Campbell K, Cohn S, Chute C, Shortliffe E, Rennels G. Scalable methodologies for distributed development of logic-based convergent medical terminology. *Methods of Information in Medicine* 1998a;37(4–5):426–439.

15. Campbell KE, Oliver DE, Spackman KA, Shortliffe EH. Representing thoughts, words, and things in the UMLS. *Journal of the American Medical Informatics Association* 1998b;5(5):421–431.

16. Chaudhri V, Farquhar A, Fikes R, Karp P, Rice J. Open knowledge base connectivity 2.0. Stanford, CA: Knowledge Systems Laboratory, Stanford University, 1998a, Technical Report KSL-98-06. p. 1–104. http://www-ksl.stanford.edu/KSL_Abstracts/KSL-98-06.html

17. Chaudhri VK, Farquhar A, Fikes R, Karp PD, Rice JP. OKBC: A programmatic foundation for knowledge base interoperability. In: *AAAI-98. Proceedings of the Fifteenth National Conference on Artificial Intelligence*; Madison, WI; July 26–30, 1998b. p. 600–607.

18. Chute C, Elkin P, Sherertz D, Tuttle M. Desiderata of a clinical terminology server. In: Lorenzi NM, editor. *Proceedings of the AMIA '99 Annual Symposium*; Philadelphia: Hanley & Belfus; 1999. p. 42–46.

19. Chute CG, Cohn SP, Campbell JR. A framework for comprehensive health terminology systems in the United States: Development guidelines, criteria for selection, and public policy implications. *Journal of the American Medical Informatics Association* 1998;5(6):503–510.

20. Chute CG, Cohn SP, Campbell KE, Oliver DE, Campbell JR. The content coverage of clinical classifications. *Journal of the American Medical Informatics Association* 1996;3(3):224–233.

21. Cimino J. Desiderata for controlled medical vocabularies in the twenty-first century. *Methods of Information in Medicine* 1998;37(4–5):394–403.

22. Cimino J, Johnson S, Hripcsak G, Hill C, Clayton P. Managing vocabulary for a centralized clinical system. In: Greenes RA, Peterson HE, Protti DJ, editors. *MEDINFO '95. Proceedings of the Eighth World Congress on Medical Informatics*; Vancouver, BC: Healthcare Computing & Communications Canada; July 23–27, 1995. p. 117–120.

23. Cimino JJ. Formal descriptions and adaptive mechanisms for changes in controlled medical vocabularies. *Methods of Information in Medicine* 1996a;35(3):202–210.

24. Cimino JJ. An approach to coping with the annual changes in ICD-9-CM. *Methods of Information in Medicine* 1996b;35(3):220.

25. Cimino JJ. From data to knowledge through concept-oriented terminologies: Experience with the Medical Entities Dictionary. *Journal of the American Medical Informatics Association* 2000;7:288–297.

26. Cimino JJ, Clayton PD, Hripcsak G, Johnson SB. Knowledge-based approaches to the maintenance of a large controlled medical terminology. *Journal of the American Medical Informatics Association* 1994;1(1):35–50.

27. Connolly D. *Extensible Markup Language (XML)*. Last updated April 7, 1997. http://www.w3.org/XML

28. Divita G, Browne A, Rindflesch T. Evaluating lexical variant generation to improve information retrieval. In: Chute CG, editor. *Proceedings of the AMIA '98 Annual Symposium*; Philadelphia: Hanley & Belfus; 1998. p. 775–779.

29. Elhanan G, Cimino JJ. Controlled vocabulary and design of laboratory results displays. In: Masys DR, editor. *Proceedings of the 1997 AMIA Annual Fall Symposium*; Philadelphia: Hanley and Belfus; 1997. p. 348–352.

30. Elhanan G, Socratous SA, Cimino JJ. Integrating DXplain into a clinical information system using the World Wide Web. In: Cimino JJ, editor.

Proceedings of the 1996 AMIA Annual Fall Symposium; Philadelphia: Hanley and Belfus; 1996.

31. Farquhar A, Fikes R, Rice J. The Ontolingua server: A tool for collaborative ontology construction. *International Journal of Human-Computer Studies* 1997;46(6):707–727.

32. Fauci AS, Braunwald E, Isselbacher KJ, Wilson JD, Martin JB, Kasper DL, et al., editors. *Harrison's Principles of Internal Medicine*. New York: McGraw-Hill; 1998.

33. Fikes R, Farquhar A, Rice J. Tools for assembling modular ontologies in Ontolingua. In: *AAAI-97. Proceedings of the Fourteenth National Conference on Artificial Intelligence*; Providence, RI; July 27–31, 1997. p. 436–441.

34. Flanagan D. *Java in a Nutshell: A Desktop Quick Reference*. 2nd edition. Sebastopol, CA: O'Reilly; 1997.

35. Forrey AW, McDonald CJ, DeMoor G, Huff SM, Leavelle D, Leland D, et al. Logical observation identifier names and codes (LOINC) database: A public use set of codes and names of electronic reporting of clinical laboratory test results. *Clinical Chemistry* 1996;42(1):81–90.

36. Fox J, Johns N, Rahmanzadeh A. Disseminating medical knowledge: The PROforma approach. *Artificial Intelligence in Medicine* 1998;14:157–181.

37. Frances A, Pincus HA, First MB, editors. *Diagnostic and Statistical Manual for Mental Disorders, Fourth Edition: DSM-IV*. Washington, DC: American Psychiatric Association; 1994.

38. Fridsma D, Gennari J, Musen M. Making generic guidelines site-specific. In: Cimino J, editor. *American Medical Informatics Association Annual Fall Symposium (formerly SCAMC)*: Hanley & Belfus; 1996. p. 597–601.

39. Friedman CP, Wyatt JC. *Evaluation Methods in Medical Informatics*. New York: Springer-Verlag; 1997.

40. Genesereth M, Nilsson N. *Logical Foundations of Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann; 1987.

41. Giuse DA, Giuse NB, Miller RA. Evaluation of long-term maintenance of a large medical knowledge base. *Journal of the American Medical Informatics Association* 1995;2(5):297–306.

42. Gruber T. A translation approach to portable ontology specifications. *Knowledge Acquisition* 1993;5(2):199–220.

43. Hammond W. Call for a standard clinical vocabulary. *Journal of the American Medical Informatics Association* 1997;4(3):254–255.

44. Henry SB, Mead CN. Nursing classification systems: Necessary but not sufficient for representing "what nurses do" for inclusion in computer-based patient record systems. *Journal of the American Medical Informatics Association* 1997;4(3):222–232.

45. Hornick RB. Rickettsial diseases. In: Bennett JC, Plum F, editors. *Cecil Textbook of Medicine*. 20th edition. Philadelphia: WB Saunders; 1996a. p. 1726–1736.

46. Hornick RB. Table 324-1. Summary of some epidemiologic features of selected rickettsial diseases of humans. In: Bennett JC, Plum F, editors. *Cecil Textbook of Medicine*. 14th edition. Philadelphia: WB Saunders; 1996b. p. 1727.

47. Hornick RB. Table 324-2. Rickettsia target cell relationships, pathologic lesions, and clinical manifestations of human rickettsioses. In: Bennett JC, Plum F, editors. *Cecil Textbook of Medicine.* 14th edition. Philadelphia: WB Saunders; 1996c. p. 1727.

48. Horrocks I, Sattler U. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation* 1999;9(3):385–410.

49. Hripcsak G, Ludemann P, Pryor T, Wigertz O, Clayton P. Rationale for the Arden Syntax. *Computers in Biomedical Research* 1994;27:291–324.

50. Humphreys BL. *UMLS Knowledge Sources, 7th Experimental Edition Documentation*. Bethesda, MD: U.S. Department of Health and Human Services, National Institutes of Health, National Library of Medicine; 1996a.

51. Humphreys BL, Hole WT, McCray AT, Fitzmaurice JM. Planned NLM/AHCPR large-scale vocabulary test: Using UMLS technology to determine the extent to which controlled vocabularies cover terminology needed for health care and public health. *Journal of the American Medical Informatics Association* 1996b;3(4):281–7.

52. Humphreys BL, Lindberg DL. The UMLS project: Making the conceptual connection between users and the information they need. *Bulletin of the Medical Library Association* 1993;81(2):170–177.

53. Humphreys BL, McCray AT, Cheh M. Evaluating the coverage of controlled health data terminologies: Report on the results of the NLM/AHCPR large scale vocabulary test. *Journal of the American Medical Informatics Association* 1997;4(6):484–500.

54. ICD-9-CM. *International Classification of Diseases, 9th Revision, Clinical Modification, Fourth Edition*. Los Angeles: Practice Management Information Corporation; 1993.

55. Korth HF, Silberschatz A. *Database System Concepts*. 2nd edition. New York: McGraw-Hill; 1991.

56. Larmouth J. *ASN.1 Complete*. Accessed April 9, 2000. http://www.nokalva.com/asn1/larmouth.html

57. Lennert K, Mohri N, Stein H, E K. The histopathology of malignant lymphoma. *British Journal of Haematology* 1975;31(Suppl):193.

58. Lindberg DA, Humphreys BL, McCray AT. The Unified Medical Language System. *Methods of Information in Medicine* 1993;32(4):281–291.

59. Loprinzi C, Alberts S, Christensen B, Hanson L, Farley D, Broers J, et al. History of the development of antiemetic guidelines at Mayo Clinic Rochester. *Mayo Clinic Proceedings* 2000;75:303–309.

60. Lukes RJ, Collins RD. Immunologic characterization of human malignant lymphomas. *Cancer* 1974;34(4):1488–1503.

61. MacGregor RM. Inside the Loom description classifier. *SIGART Bulletin* 1991;2(3):88–92.

62. Mays E, Dionne R, Weida R. K-Rep system overview. *SIGART Bulletin* 1991;2(3):93–97.

63. Mays E, Weida R, Dionne R, Laker M, White B, Liang C, et al. Scalable and expressive medical terminologies. In: Cimino JJ, editor. *Proceedings of the 1996 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1996. p. 259–263.

64. McCray AT, Nelson SJ. The representation of meaning in the UMLS. *Methods of Information in Medicine* 1995;34(1–2):193–201.

65. Metrowerks. *CodeWarrior Integrated Development Environment, Professional Release 4* [computer program]. Version 3.3. Austin, TX: Metrowerks Inc.; 1998.

66. Mitra P, Wiederhold G, Kersten M. A graph-oriented model for articulation of ontology interdependencies. In: Zaniolo C, Lockemann PC, Scholl MH, Grust T, editors. *Advances in Database Technology - EDBT 2000, Sixth International Conference on Extending Database Technology*; Konstanz, Germany; March 27–31, 2000. p. 86–100.

67. Musen MA. Domain ontologies in software engineering: Use of Protégé with the EON architecture. *Methods of Information in Medicine* 1998;37(4–5):540–550.

68. Musen MA, Tu SW, Das AK, Shahar Y. EON: A component-based approach to automation of protocol-directed therapy. *Journal of the American Medical Informatics Association* 1996;3(6):367–88.

69. Musen MA, Wieckert KE, Miller ET, Campbell KE, Fagan LM. Development of controlled medical terminology: Knowledge acquisition and knowledge representation. *Methods of Information in Medicine* 1995;34(1–2):85–95.

70. Neches R, Fikes R, Finin T, Gruber T, Sanator T, Swartout W. Enabling technology for knowledge sharing. *AI Magazine* 1991;12:36–56.

71. NLM (National Library of Medicine). 1997 Medical Subject Headings: Annotated Alphabetic List. 1996.

72. NLM (National Library of Medicine). NLM. 1998 Medical Subject Headings: Annotated Alphabetic List. 1997.

73. NLM (National Library of Medicine). NLM. *MeSH vocabulary file: Data element descriptions*. Last updated October 20, 1998.
http://www.nlm.nih.gov/mesh/elmesh99.pdf

74. NLM (National Library of Medicine). NLM. *Medical Subject Headings files available to download*. Last updated October 21, 1999.
http://www.nlm.nih.gov/mesh/filelist.html

75. Nowlan WA, Rector A, Kay S, Horan B, Wilson A. A patient care workstation based on a user-centered design and a formal theory of medical terminology: PEN & PAD and the SMK formalism. In: Clayton P, editor. *Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care*; New York: McGraw-Hill; 1991. p. 855–857.

76. Noy NF, Musen MA. An algorithm for merging and aligning ontologies: Automation and tool support. In: *Proceedings of the Workshop on Ontology*

Management at the Sixteenth National Conference on Artificial Intelligence (AAAI-99); Orlando, FL: AAAI Press, 1999a.

77. Noy NF, Musen MA. SMART: Automated support for ontology merging and alignment. In: Gaines B, Kremer R, Musen M, editors. *KAW 99. Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management*; Banff, Alberta, Canada; October 16–21, 1999b. p. Track 4, No. 7, 1–20.

78. Ohno-Machado L, Gennari JH, Murphy SN, Jain NL, Tu SW, Oliver DE, et al. The GuideLine Interchange Format: A model for representing guidelines. *Journal of the American Medical Informatics Association* 1998;5:357–372.

79. Oliver DE, Shahar Y, Shortliffe EH, Musen MA. Representation of change in controlled medical terminologies. *Artificial Intelligence in Medicine* 1999;15(1):53–76.

80. Olson NE, Erlbaum MS, Tuttle MS, Sherertz DD, Suarez-Munist O, Lipow SS, et al. Exploiting the Metathesaurus update model. In: Cimino JJ, editor. *Proceedings of the 1996 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1996. p. 902.

81. O'Neil M, Payne C, Read J. Read codes Version 3: A user-led terminology. *Methods of Information in Medicine* 1995;34(1–2):187–192.

82. Osler W, McCrae T. *The Principles and Practice of Medicine*. 9th edition. New York: D. Appleton and Company; 1923.

83. Patel-Schneider PF, Swartout B. Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge-sharing effort. November 1, 1993. p. 1–19.
http://www.ida.liu.se/labs/iislab/people/patla/DL

84. Patil R, Fikes R, Patel-Schneider P, McKay D, Finin T, Gruber T, et al. The DARPA knowledge-sharing effort: Progress report. In: Rich C, Nebel B, Swartout W, editors. *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*; Cambridge, MA: Morgan Kaufmann; Oct. 25–29, 1992. p. 777–788.

85. Paty D, Studney D, Redekop K, Lublin F. MS COSTAR: A computerized patient record adapted for clinical research purposes. *Annals of Neurology* 1994;36:134–5.

86. Peltason C. The BACK system: An overview. *SIGART Bulletin* 1991;2(3):114–119.

87. Rassinoux A, Miller R, Baud R, Scherrer J. Modeling concepts in medicine for medical language understanding. *Methods of Information in Medicine* 1998;37(4–5):361–372.

88. Rector A, Bechhofer S, Goble C, Horrocks I, Nowlan W, Solomon W. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine* 1997;9(2):139–171.

89. Rector A, Solomon W, Nowlan W, Rush T, Zanstra P, Claassen W. A terminology server for medical language and medical information systems. *Methods of Information in Medicine* 1995;34(1–2):147–157.

90. Resnick L, Borgida A, Brachman R, McGuinness D, Patel-Schneider P, Zalondek K. *CLASSIC Description and Reference Manual for the COMMON LISP Implementation, Version 2.2*. New Jersey: AT&T Bell Lab; 1993.

91. Robinson D, Schulz E, Brown P, Price C. Updating the Read codes: User-interactive maintenance of a dynamic clinical vocabulary. *Journal of the American Medical Informatics Association* 1997;4(6):465–472.

92. Rocha RA, Huff SM, Haug PJ, Warner HR. Designing a controlled medical vocabulary server: The VOSER project. *Computers and Biomedical Research* 1994;27(6):472–507.

93. Rogers J, Price C, Rector A, Solomon W, Smejko N. Validating clinical terminology structures: Integration and cross-validation of Read Thesaurus and GALEN. In: Chute CG, editor. *Proceedings of the AMIA '98 Annual Symposium*; Philadelphia: Hanley & Belfus; 1998. p. 845–849.

94. Rogers JE, Rector AL. Terminological systems: Bridging the generation gap. In: Masys DR, editor. *Proceedings of the 1997 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1997. p. 610–614.

95. Rothwell DJ. SNOMED-based knowledge representation. *Methods of Information in Medicine* 1995;34(1–2):209–213.

96. Sargeaunt J, translator. *Terence II: Phormio, The Mother-in-Law, The Brothers*. In: Goold GP, editor. *Loeb Classical Library*, No. 23. Cambridge, MA: Harvard University Press; 1995, p. 50.

97. Schulz EB, Barrett JW, Price C. Semantic quality through semantic definition: Refining the Read codes through internal consistency. In: Masys DR, editor. *Proceedings of the 1997 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1997. p. 615–619.

98. Shahar Y, Miksch S, Johnson PD. The Asgaard project: A task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial Intelligence in Medicine* 1998;14(1–2):29–51.

99. Shahar Y, Musen MA. Plan recognition and revision in support of guideline-based care. In: *1995 Spring Symposium Series, Stanford University*: American Association for Artificial Intelligence; March 27–29, 1995.

100. Spackman KA, Campbell KE. Compositional concept representation using SNOMED: Towards further convergence of clinical terminologies. In: Chute CG, editor. *Proceedings of the AMIA '98 Annual Symposium*; Philadelphia: Hanley & Belfus; 1998. p. 740–744.

101. Spackman KA, Campbell KE, Cote RA. SNOMED RT: A reference terminology for health care. In: Masys DR, editor. *Proceedings of the 1997 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1997. p. 640–644.

102. Spitzer RL, Williams JBW, editors. *Diagnostic and Statistical Manual for Mental Disorders, Third Edition Revised: DSM-III-R*. Washington, DC: American Psychiatric Association; 1987.

103. Stitt E. *The Diagnostics and Treatment of Tropical Diseases*. 2nd edition. Philadelphia: P. Blakiston's Son and Company; 1917.

104. Suarez-Munist ON, Tuttle MS, Olson NE, Erlbaum MS, Sherertz DD, Lipow SS, et al. MEME-II supports the cooperative management of terminology. In: Cimino JJ, editor. *Proceedings of the 1996 AMIA Annual Fall Symposium*; Philadelphia: Hanley & Belfus; 1996. p. 84–88.

105. Swartout B, Patil R, Knight K, Russ T. Toward distributed use of large-scale ontologies. In: Gaines B, Musen M, editors. *KAW 96. Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*; Banff, Alberta, Canada; Nov. 9–14, 1996. p. 32.1–32.19.

106. Thurin A, Carlsson M, Gill H, Wigertz O. Arden Syntax and GALEN terminology support: A powerful combination to represent medical knowledge. In: Greenes RA, Peterson HE, Protti DJ, editors. *MEDINFO '95. Proceedings of*

the Eighth World Congress on Medical Informatics; Vancouver, BC: Healthcare Computing & Communications Canada; July 23–27, 1995. p. 110.

107. Topley K. *CORE Java Foundation Classes*. Upper Saddle River, NJ: Prentice Hall; 1998.

108. Tuttle M, Nelson S. A poor precedent. *Methods of Information in Medicine* 1996;35(3):211–217.

109. Tuttle M, Olson N, Campbell K, DD S, SJ N, WG C. Formal properties of the Metathesaurus. In: Safran C, editor. *Proceedings of the Eighteenth Annual Symposium on Computer Applications in Medical Care*; 1994. p. 145–149.

110. Tuttle M, Suarez-Munist O, Olson N, Sherertz D, Sperzel W, Erlbaum M, et al. Merging terminologies. In: Greenes RA, Peterson HE, Protti DJ, editors. *MEDINFO '95. Proceedings of the Eighth World Congress on Medical Informatics*; Vancouver, BC: Healthcare Computing & Communications Canada; July 23–27, 1995. p. 162–166.

111. Tuttle MS, Sherertz D, Erlbaum M, Sperzel W, Fuller L, Olson N. Adding your terms and relationships to the UMLS Metathesaurus. In: Clayton PD, editor. *Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care*; New York: McGraw-Hill; 1991. p. 219–223.

112. Uschold M, Clark P, Healy M, Williamson K, Woods S. An experiment in ontology reuse. In: Gaines B, Musen M, editors. *KAW 96. Proceedings of the Eleventh Knowledge Acquisition for Knowledge-Based Systems Workshop*; Banff, Alberta, Canada; April 18–23, 1998.

113. Valente A, Russ T, MacGregor R, Swartout W. Building and (re)using an ontology of air-campaign planning. *IEEE Intelligent Systems* 1999;14(1):27–36.

114. Walker D, Raoult D, Brouqui P, Marie T. Rickettsial diseases. In: Fauci AS, Braunwald E, Isselbacher KJ, Wilson JD, Martin JB, Kasper DL, et al., editors. *Harrison's Principles of Internal Medicine*. 14th edition. New York: McGraw-Hill; 1998. p. 1045–1052.

115. *Webster's New Collegiate Dictionary*. 9th edition. Springfield, MA: Merriam-Webster; 1991. Ontology; p. 825.

116. Weinstein PC, Birmingham WP. Comparing concepts in differentiated ontologies. In: Gaines B, Kremer R, Musen M, editors. *KAW 99. Proceedings of the Twelfth*

Workshop on Knowledge Acquisition, Modeling and Management; Banff, Alberta, Canada; October 16–21, 1999. p. Track 5, No. 6, 1–22.