Approaches for Guideline Versioning Using GLIF

Mor Peleg, Ph.D.[1], Rami Kantor, M.D.[2]

[1]Stanford Medical Informatics and [2]Division of Infectious Diseases and Center for AIDS Research, Stanford University School of Medicine, Stanford, CA

Computer-interpretable clinical guidelines (CIGs) aim to eliminate clinician errors, reduce practice variation, and promote best medical practices by delivering patient-specific advice during patient encounters. Clinical guidelines are being regularly updated and revised to handle expanding clinical knowledge. When revising CIGs, much effort can be saved by specifying changes among versions instead of encoding revised guidelines from scratch. A representation of differences between versions could focus the process of re-implementing CIGs in a clinical environment and help users understand and embrace changes. Guideline versioning has not been adequately dealt with by existing CIG formalisms. We present three approaches for CIG versioning. Focusing on one approach, we developed a versioning tool based on version 3 of the GuideLine Interchange Format (GLIF3), and used it to represent two guideline versions for management of community-acquired pneumonia (CAP) and the changes between them.

1 Introduction

Clinical guidelines aim to eliminate clinician errors, reduce practice variation, and promote best practices. CIGs are clinical guidelines encoded in a computer-interpretable way and integrated with clinical information systems. CIGs can deliver patient-specific advice during clinical encounters, which makes them more likely to affect clinician behavior compared with narrative guidelines[1]. Many groups are developing formalisms for representing CIGs[2]. One of these formalisms, on which we base this work, is GLIF3[3]. GLIF3 specifies guidelines as flowcharts of steps representing clinical actions, decisions, and patient states. The steps' details generate a computable specification enabling logical consistency and inference.

Guidelines are living documents that must be regularly updated and revised to handle expanding knowledge, including new risk factors, drugs, diagnostic tests, clinical studies, as well as pathogen incidence and drug resistance in the infectious diseases field. Corrective and perfection maintenance also change guideline knowledge. Revised clinical guidelines necessitate CIGs update, involving significant time and effort. This would make specifying changes among versions more valuable, as opposed to encoding revised guidelines from scratch. Moreover, representation of differences between a new guidelines version and one that has already been implemented in a clinical environment would greatly ease the implementation update process and would help users of the original version to understand and embrace changes and their justifications.

2 Related approaches for versioning

Despite the wealth of CIG formalisms, guideline versioning has not been adequately addressed by any of them[2]. CIG formalisms do not go beyond allocating a textual slot for indicating the version of the narrative guideline to which the CIG corresponds. Versioning of knowledge models is addressed in the related field of clinical vocabulary systems[4,5] and in, ontology-[6], database-[7,8], and workflow-evolution[9-11]. We summarize the approaches for versioning knowledge models, in which changes are expressed in terms of change operations. We looked at the way in which changes between versions of knowledge models are recorded, the way by which change operations are derived, and the tasks supported by versioning.

Two approaches are used to record change operations: creating a log file as changes are made and comparing two versions to produce a difference table.

Basic change operations are derived from the basic elements of knowledge models and enable adding, removing, and changing those elements. Thus, in vocabulary systems terms can be added or removed and the values of term attributes can be changed. In ontology evolution, basic operations include changes to classes, slots, slot restrictions, and instances. In relational databases the basic elements that are changed are relations and attributes, whereas in object-oriented databases they are classes, is-a relations, and attributes. Change operators in Workflows affect variables, task attributes, and ordering of tasks.

Additional change operations are defined to support versioning tasks. In vocabulary systems, the basic change operations are further classified to reflect reasons for change[4,5]. For example, term addition is classified to creating a new term, refining a previous

term, and replacing an ambiguous term. The semantic taxonomy affects the tasks of querying data, and interpreting previously encoded data. In ontology evolution, the basic operations are refined to enable tasks of preserving instance data, answers to queries, and knowledge that was inferred from the instance data[6]. As an example, moving a slot from class to class is refined into distinct change operations, reflecting the relationship between the two classes (e.g., subclass relation). In database schema versioning, basic operators can be combined to define complex operations such as class splitting and intersection[7,8]. These operations support tasks of data and queries migration. Complex and reusable operators can be defined based on basic operators[11]. These simplify the task of specifying changes between workflow versions.

3 Approaches for versioning of GLIF3 CIGs

The approaches we've considered for versioning of GLIF3 CIGs involve specification of a logical change model, as well as development of tools for representing and visualizing changes.

3.1 Logical models for versioning of GLIF3 CIGs

We considered three models for representing changes among CIG versions: log files, difference tables, and version annotations.

Log files list the history of change operations that were used to derive a new CIG version from an existing one. As Figure 1(a) shows, each entry in the log file is characterized by the following attributes: (1) operation type (add, retire), (2) version at which the change was made, (3) ID of the instance that was changed, and in the case of changes to slot values: (4) slot whose value has changed, and (5) slot value.

Difference tables contain the results of comparing an original version of a CIG with a revised version. Figure 1(b) shows an example of a difference table.

Version annotations allow maintaining several guideline versions in a single knowledge base. Each CIG element is annotated with version information. A logical model that enables such versioning of GLIF3 instances requires an extension of the GLIF3 model to represent versioning relations. GLIF3 is an object-oriented logical model whose classes have slots (attributes). Each slot is a binary relation between the class to which it belongs and the allowed data type of the slot's value. In addition, a class can inherit the slots of other classes. As shown in Figure 1(c), we extended GLIF3 by defining two types of versioning relations: (1) relations between GLIF3 classes and the Version_Info class, and (2) slot relations, which are ternary relations among a set of classes, a set of allowed types, and Version_Info. *Version_Info* includes slots for specifying the version ID, the change operation (add, retire), and the reason for change.

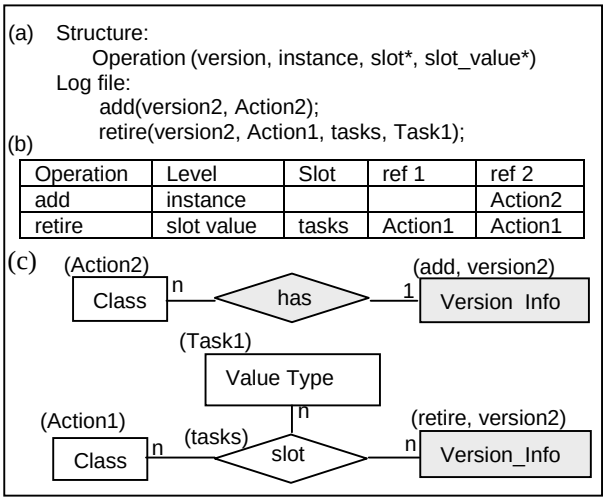

*Figure 1. Three ways of representing changes: (a) log file; (b) difference table; (c) version annotations used to extend the CIG ontology. * = optional; ref1, ref2 = references to instance in version 1 and 2. Rectangles denote ontology classes; Diamonds denote relations. Extensions to the CIGs ontology are shown in gray. Instances are shown in parentheses.*

3.2 Tools for representing changes

To represent and visualize changes, we leveraged the capabilities of the existing GLIF3 authoring tool[12] that was developed using the Protégé-2000 knowledge-modeling environment. We describe briefly tools that support log files and difference tables, and concentrate mainly on the tool that we developed to support versioning annotations.

Users can create log files expressing changes between versions. Logs could be automatically processed to execute changes on the old CIGs, generating new CIGs. This could be done, for example, through commands in Algernon – a rule-based inference system that has been interfaced with Protégé-2000[13].

CIG developers can use authoring tools to create a CIG specification and revise it. The CIG versions could be compared using tools such as PROMPTDIFF[14], which compares knowledge bases created in Protégé-2000. The difference table that PROMPTDIFF creates does not record the slot values that have changed. Instead, users can view the instances that have changed by clicking on table rows.

To support versioning annotations, we extended the capabilities of the Protégé-2000 GLIF3 authoring tool

so that it could serve as a CIG versioning tool. Our versioning tool can: (1) be used to create a new CIG specification or to create a version of a CIG by modifying an existing version, and (2) display versions of a CIG in a single view, highlighting differences between them. To support the requirements of the versioning tool we implemented the logical model of versioning relations. Since Protégé-2000 does not represent slots as ternary relations, we simulated ternary relations by changing the GLIF3 ontology:

- We added a *version_info* slot: a binary relation from GLIF3_Entity (i.e., a super class of all GLIF3 classes) to the Version_Info class.
- We transformed slots of simple types (e.g., STRING) into classes that have a slot of the simple type and a version_info slot
- We changed the cardinality of single-valued slots into multiple, so that we could keep slot values that originate in different versions

To add or retire an instance, we set its Version_Info to the current version and to the operation *add* or *retire*. When we want to retire a slot value, we create a new slot value, which is similar to the previous slot value except for its version_info. This ensures that other instances could still refer to the previous slot value with its original version_info.

We use the version type *change* as a high-level abstraction of instances that have slot values containing different version information.

3.3 Visualization of changes

We developed several visualization techniques to facilitate human understanding of the changes between versions. One technique enables users to view algorithms graphically, distinguishing their elements according to their version information. Users can start from the conceptual view of the algorithm and drill down to the details of each algorithm element, shown in forms. Currently Protégé-2000 enables only textual distinctions. By setting the browser key as the *version* slot of the entities, users can easily distinguish entities that were changed, retired, or added. Future plans include extension of Protégé's capabilities to allow color distinctions based on slot values.

Another visualization technique is to summarize changes in tabular format, for better visualization. We created queries in Protégé's axiom Language (PAL) that summarize changes to non-algorithmic guideline knowledge elements of GLIF3. GLIF3 uses two classes to specify non-algorithmic medical knowledge: Concept and Concept_Relationship (CR). *Concepts* (e.g., drug, diagnostic test) represent clinical semantic types around which guideline knowledge is organized. Concepts are specified by *name* and *ID* that are taken from a controlled terminology. *CRs* specify relations between pairs of concepts (e.g., Penicillin *is-a* Drug). CRs have slots for specifying the two concepts and the type of relation between them. Being GLIF3 entities, Concepts and CRs are associated with version information. Queries use version information to find knowledge elements that have changed.

4 Case study: guidelines for pneumonia

We used the GLIF3 versioning tool that we developed to represent two guideline versions for management of CAP, that were developed by the Infectious Disease Society of America in 1998 (version 1)[15] and in 2000 (version 2)[16]. Figures 2 and 3 depict screenshots of the GLIF3 versioning tool, showing part of the CAP algorithm and details of one of the algorithm steps that had changed.

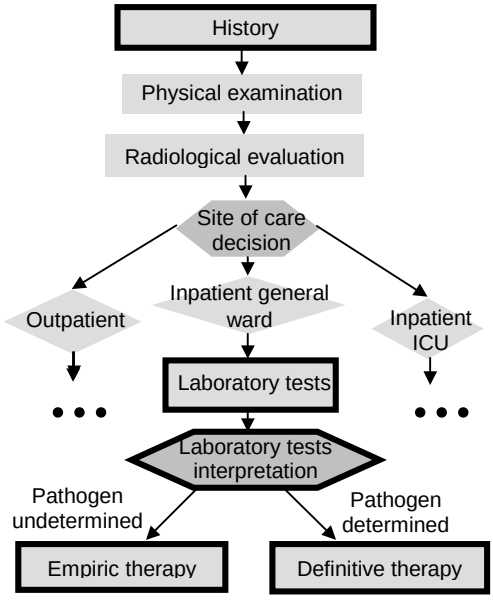

Figure 2. Part of the CAP algorithm showing elements that harbor changes in bold. Rectangles denote action steps; diamonds denote patient state steps; hexagons denote decisions. Three dots indicate 'outpatient' and 'inpatient ICU' algorithm parts, with a similar as the shown 'inpatient ICU' part. ICU: intensive care unit.

To encode the 1998 guideline in GLIF3, we designed a generic algorithm for management of CAP. The generic algorithm includes recommendations for general diagnostic tests (radiology), which are subsequently refined to specific tests (X_Ray), and drug groups, which are refined to specific medications. We expect the generic design to make most common

changes easily updateable. Reasons for common changes in recommendations include: new pathogen, pathogen resistance, new drug, drug toxicity, new diagnostic test, retired diagnostic test, new clinical evidence, refuted clinical evidence, and unknown reason. These reasons constitute the allowed values of the reason_for_change slot of the Version_Info class.

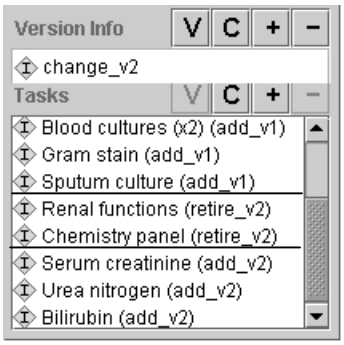

Figure 3. Details of the action step "Laboratory tests", shown in Figure 2. Instead of using colors to distinguish version information of each slot value, information is represented textually. v1-version 1; v2-version 2.

To represent non-algorithmic knowledge of the CAP guideline, we specified concepts belonging to four high-level concepts: drug, pathogen, clinical condition, and diagnostic test. We specified CRs of types: Diagnostic_Test *studies* Pathogen, (Clinical) Condition *has-likely* Pathogen, Pathogen *is-diagnostic-of* Condition, and Pathogen *is-not-diagnostic-of* Condition. We used CRs of type *is-a* are used to link a concept to its parent concept (e.g., Carbapenem is_a Drug). We created PAL queries to visualize concepts and CRs that have changed. We used separate queries for each type of CR. To query for changes in each of the high-level concepts separately, we utilized *is-a* CRs to trace each concept to one of the high-level concepts. Figure 4 shows one of the queries. Table 1 summarizes medical knowledge that changed between versions, calculated from query results.

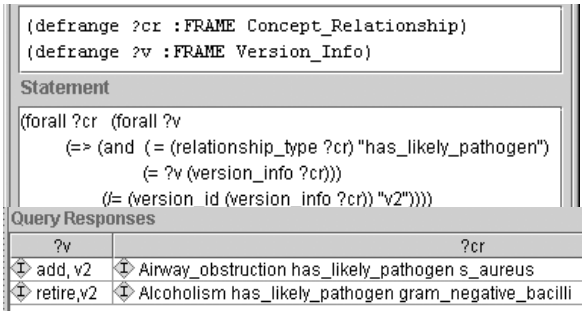

Figure 4. A Query for changes in CRs of type 'Condition has-likely Pathogen'. Only two of the results are shown

Table 1. Summary of changes between versions associated with clinical knowledge. For each knowledge item, the number of instances that were retired and added in version 2 are shown. ret-retire; v2-version 2

Knowledge Item	ret_v2	add_v2
Drugs	2	7
Patient clinical conditions (Conditions)	0	1
Pathogens	0	4
Diagnostic tests	0	0
Conditions have likely Pathogens	1	10
Pathogens are diagnostic of Conditions	1	0
Pathogens not diagnostic of Conditions	0	0
Diagnostic tests study Pathogens	0	4
Drug groups optional for Pathogens	4	5
Drug groups optional for Conditions	8	6

5 Discussion

In this paper we have described approaches for using a GLIF versioning tool to revise a CIG specification. Although we have concentrated on GLIF3, our approaches to versioning could easily be applied to other CIG formalisms that have an object-oriented or frame-based logical model, such as EON and PRODIGY[2], whose authoring tools are based on Protégé-2000.

The version-relations approach represents several versions of a CIG in a single knowledge base, while making the differences explicit. Institutions and/or organizations that are publishing guidelines may use the tool when updating versions. Using the versioning tool, guideline developers will be creating conceptual-level specifications in GLIF3. This would greatly simplify the development of a computable-level specification. Using the tool would enforce consistency, thus minimizing: knowledge gaps within a guideline, ambiguous text, and lack of justification for changes between versions.

Visualizing two guideline versions simultaneously and noting the reasons for changes can assist users in understanding the changes as well as facilitate and focus the implementation of revised CIGs. The need for revising CIGs can arise not only when institutions that publish guidelines revise them, but also when institutions adapt CIGs to their local environment.

The versioning approach that we pursued uses simple change operations (i.e., add, retire), but, at the same time, allows specification of reasons for changes. Knowing the reason for making a change in a CIG is important for clinicians who have been using a previous guideline version. The reasons for change were considered by the vocabulary evolution schemes. In those schemes, the reasons for changes are reflected in the taxonomy of change operations. Although some of the other approaches of model evolution, reviewed

in the Introduction, support a variety of syntactic as well as complex change operations, they do not represent reasons for changes. Although our tool does not support complex change operations, we use queries to aggregate basic changes into meaningful patterns of change. For example, changing the preference of a drug for a certain situation from an *alternative* to a *preferred* category is aggregated from the operations of retiring the rule that recommends the drug as an alternative option, and adding a rule that recommends the same drug as a preferred option.

The way in which we represent changes in medical knowledge could be used for indexing and searching CIGs and CIG versions. Here, we organize medical knowledge according to drugs, diagnostic tests, pathogens, clinical conditions, and relationships among these concepts. This can aid in selectively searching for CIG versions according to their content, which may exist in one version but not in the other.

Development of a tool to capture differences between different versions of guidelines may be a first step towards creation of mega-guidelines, which would encompass all published guidelines in a certain field, such as CAP. Based on a generic algorithm for addressing specific clinical circumstances, the tool could enable noting differences among guidelines that were created by different organizations. This could be of help to practicing physicians, and could increase the utilization of clinical guidelines, improving patient care.

Acknowledgements

We thank Samson Tu for his valuable comments.

References

1. Overhage JM, Tierney WM, Zhou XH, McDonald CJ. A Randomized Trial of "Corollary Orders to Prevent Errors of Omission.". *J Am Med Inf Assoc* 1997;4(5):364-375.

2. Peleg M, Tu SW, Bury J, Ciccarese P, Fox J, Greenes RA, et al. Comparing Computer-Interpretable Guideline Models: A Case-Study Approach. *J Am Med Inf Assoc* 2003;10(1):52-68.

3. Peleg M, Boxwala A, Ogunyemi O, Zeng Q, Tu S, Lacson R, et al. GLIF3: The Evolution of a Guideline Representation Format. Proc AMIA Symp; 2000. p. 645-649.

4. Cimino JJ. Formal Descriptions and Adaptive Mechanisms for Changes in Controlled Medical Vocabularies. *Meth Inform Med* 1996;35:202-210.

5. Oliver DE, Shahar Y. ChangeManagement of Shared and Local Versions of Health-Care Terminologies. *Meth Inform Med* 2000;39:278-290.

6. Noy NF, Klein M. Ontology Evolution: Not the Same as Schema Evolution. *Knowledge and Information Systems, in press* 2003.

7. Franconi E, Grandi F, Manderoli F. A Semantic Approach for Schema Evolution and Versioning in Object-Oriented Databases. In: Lecture Notes in Computer Science 2065; 2000. p. 85-99.

8. Roddick JF, Craske NG, Richards TJ. A Taxonomy for Schema Versioning Based on the Relational and Entity Relationship Models. In: International Conference on Conceptual Modeling / the Entity Relationship Approach; 1993; 1993. p. 137-148.

9. Casati F, Ceri S, Pernici B, Pozzi G. Workflow Evolution. *Data and Knowledge engineering* 1998;24(3):211-238.

10. Kradolfer M, Geppert A. Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Management. In: Proceedings. 3rd IFCIS International Conference on Cooperative Information Systems; 1999. p. 104-114.

11. Joeris G, Herzog O. Managing Evolving Workflow Specifications with Schema Versioning and Migration Rules: University of Bremen; 1999. Report No.: TZI Technical Report 15.

12. Peleg M, Patel VL, Snow V, Tu S, Mottur-Pilson C, Shortliffe EH, et al. Support for Guideline Development through Error Classification and Constraint Checking. Proc AMIA Symp; 2002. p. 607-11.

13. Gennari J, Musen MA, Fergerson RW, Grosso WE, Crubezy M, Eriksson H, et al. The Evolution of Protege: An Environment for Knowledge-Based Systems Development. *International Journal of Human-Computer Interaction* 2002;58(1):89-123.

14. Noy NF, Musen MA. PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions. AAAI/IAAI; 2002. p. 744-750.

15. Bartlett JG, Breiman RF, Mandell LA, File TM. Community-Acquired Pneumonia in Adults: Guidelines for Management. *Clinical Infectious Diseases* 1998;26:811-838.

16. Bartlett JG, Dowell SF, Mandell LA, File TM, Musher DM, Fine MJ. Practice Guidelines for the Management of Community-Acquired Pneumonia in Adults. *Clinical Infectious Diseases* 2000;31:347-382.