

Guideline-based decision support in medicine : modeling guidelines for the development and application of clinical decision support systems

Citation for published version (APA):

Clercq, de, P. A. (2003). *Guideline-based decision support in medicine : modeling guidelines for the development and application of clinical decision support systems*. [Phd Thesis 2 (Research NOT TU/e / Graduation TU/e), Electrical Engineering]. Technische Universiteit Eindhoven. <https://doi.org/10.6100/IR566704>

DOI:

[10.6100/IR566704](https://doi.org/10.6100/IR566704)

Document status and date:

Published: 01/01/2003

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Guideline-based Decision Support in Medicine

**Modeling Guidelines for the Development and Application of Clinical
Decision Support Systems**

Paul de Clercq

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

De Clercq, Paul A.

Guideline-based Decision Support in Medicine: Modeling Guidelines for the Development and Application of Clinical Decision Support Systems / by Paul A de Clercq. - Eindhoven : Technische Universiteit Eindhoven, 2003.

Proefschrift. - ISBN: 90-9016967-9

NUGI 981

Trefwoorden: medische informatica / richtlijnen / beslissingsondersteuning / kennissystemen

Subject headings: medical informatics / guidelines / decision support / knowledge systems

Druk: PrintPartners Ipskamp Enschede

Guideline-based Decision Support in Medicine

**Modeling Guidelines for the Development and Application of Clinical
Decision Support Systems**

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. R.A. van Santen, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 24 juni 2003 om 16.00 uur

door

Paul Adrianus de Clercq

geboren te Eindhoven

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. A. Hasman
en
prof.dr. H.H.M. Korsten

Copromotor:
dr.ir. J.A. Blom

Contents

Chapter 1

General Introduction	9
1 Guidelines in medicine	10
2 Active guideline-based decision support systems	11
3 Thesis overview	12
References	14

Chapter 2

Approaches for Creating Computer-Interpretable Guidelines that Facilitate Decision Support: a Review	17
1 Introduction	18
2 Areas	19
3 The Arden Syntax	21
4 The GuideLine Interchange Format (GLIF)	25
5 PROforma	34
6 Asbru	44
7 EON	51
8 Discussion	61
References	71

Chapter 3

The Application of Problem-Solving Methods and Ontologies for the Development of Shareable Guidelines	75
1 Introduction	76
2 The ontological guideline representation	79
3 The framework	82
4 Examples	84
5 Results	94
6 Discussion	95
Acknowledgements	98
References	98

Chapter 4

Design and Implementation of a Framework to Support the Development of Clinical Guidelines	101
1 Introduction	102
2 Materials and methods	103
3 The Gaston framework	106
4 Results	135
5 Discussion	136
References	139

Chapter 5

A Strategy for Development of Practice Guidelines for the ICU Using Automated Knowledge Acquisition Techniques	143
1 Introduction	144
2 Materials and methods	144
3 Results	151
4 Discussion	154
5 Conclusions	155
References	156

Chapter 6

Experiences with the Development, Implementation and Evaluation of Automated Decision Support Systems	159
1 Introduction	160
2 Automated feedback on test ordering in general practice	161
3 A real-time reminder system in Critical Care environments	167
4 A Multidisciplinary Psychoactive Drug Selection Advisory System	177
5 Consumer Health Records for managing chronic diseases	183
6 Discussion	191
References	192
Appendix: CritICIS questionnaire	195

Chapter 7

General Discussion and Conclusions	197
1 Introduction	198
2 The Gaston representation model	198
3 The Gaston guideline development environment	205
4 Current and future research	208
5 Conclusions	209
References	210

Contents

Summary	211
Samenvatting	215
Dankwoord	219
Curriculum Vitae	223

CHAPTER 1

GENERAL INTRODUCTION

1 Guidelines in medicine

During the last decade, studies have shown the benefits of using clinical guidelines in the practice of medicine [1] such as a reduction of practice variability and patient care costs, while improving patient care [2]. According to the Institute Of Medicine (IOM), a guideline is defined as: ‘a systematically developed statement to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances’ [3]. Although guidelines have been developed for more than 50 years, recently the emphasis has focused on the development of systematic and evidence-based guidelines as well as their evaluation and ease-of-use in daily practice. A variety of guidelines have been developed that focus on 1) different phases of the patient care process (e.g., patient screening, diagnosis, workup, referral and management), 2) different application domains (e.g., disease management, protocol-based care and consultation), and 3) different modes of use (e.g., clinical reference, knowledge source, education, quality assurance).

Although the potential application of guidelines in daily care is enormous, a number of difficulties exist related to the development and implementation of guidelines. One of them is the interpretation of the content of a guideline: the exact meaning of terms is not always defined, recommendations are not always clearly articulated and sometimes vague wording is used (e.g., what is meant when a guideline states: ‘start the prescription of an anti-hypertensive when the patient has a blood pressure that is *too high* for *too long a period*’).

Another problem has to do with the fact that creating and updating guidelines that keep up with state-of-the-art knowledge requires a huge amount of time and resources, which are often used inefficiently [1]. A related problem is the development and implementation of (inter)national guidelines on an institutional level. As substantial time and effort is needed to create good guidelines, there is an incentive to make guidelines sufficiently general to be shared among different institutions. Site-independent guidelines are difficult to use, however, without modifications to reflect the way in which medical care is delivered within a particular organization [4]. Most guidelines undergo changes to make them acceptable to health care providers within a particular setting. These changes must be valid and consistent with the original guideline. When guidelines are updated on an (inter)national level, these changes have to be propagated to the guidelines on an institutional level while keeping the local adaptations intact. This requires sophisticated versioning and adaptation methods. Although the importance of guidelines is increasingly

recognized, health care institutions typically pay more attention to guideline development than to guideline implementation for routine use in daily care [5].

Finally, there have only been limited efforts to evaluate the use and impact of guidelines in clinical practice. Although the use of guidelines is increasingly recognized as a method to improve the quality and cost-effectiveness, there is still little known whether they truly live up to these expectations. Most of these guidelines are written down as large documents in a textual format [6-9], which are often cumbersome to read and difficult to integrate in the patient care process. Also, studies have shown that clinicians are usually not familiar with textual guidelines and do not apply them appropriately during actual care [10].

2 Active guideline-based decision support systems

One of the problems with presenting guidelines as (structured) textual documents to care providers is that it is a passive method of decision support: the care provider must decide whether consultation of a guideline is necessary. Often, care providers are convinced that their actions agree with guideline standards and there is no need to consult the corresponding guideline in order to be sure. In reality however, these actions may oppose the guideline's intentions [11].

Implementing guidelines in active computer-based decision support systems promises to improve the acceptance and application of guidelines in daily practice because the actions and observations of care providers are monitored and advice is generated whenever a guideline is not followed. Various studies, covering a wide range of clinical settings and tasks, concluded that the use of these systems significantly improves the quality of care, especially when used in combination with clinical information systems such as Electronic Patient Record (EPR) systems [12]. It is stated that these decision support systems are in fact not only crucial elements in long-term strategies for promoting the use of guidelines [13] but also necessary for the future of medical decision making in general [14].

Computer-based clinical guidelines are increasingly applied in diverse areas such as policy development, utilization management, education, clinical trials, and workflow facilitation. Many parties are developing computer-based guidelines as well as decision support systems that incorporate these guidelines, covering a wide range of clinical settings and tasks [15]. Despite these efforts, only a few systems progressed beyond the prototype stage and the research laboratory. Building systems that are both effective in supporting clinicians and accepted by them has proven to be a difficult task. Yet, of the

few systems that were evaluated by a controlled trial, the majority showed impact [16].

Various difficulties are encountered with respect to the guideline development process, which ranges from the development of a guideline representation model to the implementation of actual decision support systems that operate in daily practice. Some of these difficulties are similar to the ones mentioned earlier, related to the development of guidelines in general such as:

- *How to interpret the content of a guideline;*
- *How to handle local adaptation and synchronization between (inter)national and local guidelines;*
- *How to evaluate guidelines and decision support systems in daily practice.*

In addition, new difficulties arise:

- *How to represent and share various types of guidelines using a formal and unambiguous representation;*
- *How to translate guidelines from a textual format into this formal representation;*
- *How to verify guidelines;*
- *How to interface guideline-based decision support systems with external patient information systems;*
- *How to provide decision support to a care provider in daily practice.*

3 Thesis overview

The project, described in this thesis, aims at answering the above-mentioned questions by developing and evaluating a generic approach that addresses various aspects related to the guideline development process such as *how to represent, acquire and implement computer-based guidelines*. The approach has led to the development of the Gaston framework, which is described and discussed in this thesis.

In parallel to this project that started in 1996, several other research groups also started working on developing generic methodologies for representing, acquiring and implementing computer-based guidelines [17-24], although each project had its own focus points. The current results of those projects are partly comparable to the results of this project. The various similarities and differences will be explained in the remaining part of this thesis.

This thesis is divided into four main parts: 1) description of the requirements, 2) description of the methods that led to the development of the Gaston approach, 3) evaluation of the approach and 4) general discussion.

3.1 Requirements

When this project started, there was no blueprint available that mentioned requirements for methodologies or approaches aimed at the development and implementation of computer-based guidelines. Therefore, first a literature review of existing approaches was conducted. **Chapter 2** describes and discusses existing approaches, after which a number of functional requirements are postulated that form a basis for the development of the Gaston approach. The first version of this chapter was written in 1997. However, it was recently updated to reflect the large amount of changes and updates that were made by the approaches during the last years. As a result, a number of the requirements that formed the starting point of the Gaston approach were also recognized by and implemented in other approaches.

3.2 Methods

Using the requirements in **Chapter 2**, **Chapter 3** and **Chapter 4** describe the methods used to develop the Gaston approach, which consists of a methodology for the development and implementation of computer-based guidelines and guideline-based decision support systems. **Chapter 3** focuses on the aspect of guideline representation. It describes a new guideline representation formalism that is based on the concepts of ontologies [25], primitives [26] and Problem-Solving Methods (PSMs) [27], which aims at improving the acceptance of sharable guidelines. It also shows some examples of guidelines that were represented in terms of the developed representation.

Chapter 4 describes the Gaston framework: a framework that, based on the requirements in **Chapter 2** and the guideline representation model in **Chapter 3**, facilitates the development and implementation of computer-based guidelines and guideline-based decision support systems. This chapter describes a guideline authoring environment that enables guideline authors to define guidelines in terms of the developed representation model, and a guideline execution environment that is able to execute guidelines and interfaces with external patient information systems. Also, the chapter describes in more detail the techniques behind the guideline representation model.

3.3 Evaluation

In order to evaluate whether the Gaston approach was able to facilitate the development and implementation of guideline-based decision support systems in different medical and application domains, a number of decision support systems were developed with the approach. These systems and experiences with the development of these systems are discussed in **Chapter 5** and **Chapter 6**. **Chapter 5** describes and discusses the first system that was created by means of the Gaston approach. This system, named CritICIS, was developed for use in Intensive Care Units (ICUs) and provided decision support to ICU health care workers by means of generating reminders when certain guidelines were not followed. This chapter describes the system as well as a retrospective evaluation of the guidelines that formed the CritICIS knowledge base.

Chapter 6 describes the experiences with a number of systems that were developed with the Gaston approach with respect to guideline representation, acquisition, verification and execution. The contents of this chapter, which describes and discusses systems in the areas of ICU, family practice, psychiatry and chronic disease management, is based on a number of articles that were published in a variety of journals [28-31].

3.4 General discussion

This thesis concludes with **Chapter 7**, which contains a general discussion, conclusions and suggestions for further research.

References

1. Grimshaw JM. Russel IT. Effects of Clinical Guidelines on Medical Practice: A Systematic Review of Rigorous Evaluation. *Lancet* 1993;342:1317-22.
2. Effective Health Care. Implementing Clinical Practice Guidelines: Can guidelines be used to improve clinical practice? *Effective Health Care* 1994;1(8).
3. Field MJ, Lohr KN (eds.). *Clinical Practice Guidelines: Directions for a New Program*. Washington, DC: National Academy Press 1990.
4. Fridsma DB, Gennari JH, Musen MA. Making Generic Guidelines Site-Specific. *Proc AMIA* 1996;:597-601.
5. Audet A, Greenfield S, Field M. Medical practice guidelines: current activities and future directions. *Ann Intern Med* 1990;113:709-14.
6. The Agency for Health Care Policy and Research. *Acute Low Back Pain in Adults: Assessment and Treatment Quick Reference Guide for Clinicians* 1994(14).
7. The Agency for Health Care Policy and Research. *Smoking Cessation Clinical Practice Guideline* 1996(18).
8. National Heart, Lung, and Blood Institute. *Guidelines for the Diagnosis and Management of Asthma. Expert Panel Report 2*. Washington: NIH 1997.
9. National High Blood Pressure Education Program. *The Sixth Report of the Joint National Committee on Detection, Evaluation, and Treatment of High Blood Pressure*. Washington: NIH; 1998.

10. Vissers MC, Hasman A, van der Linden CJ. Impact of a protocol processing system (ProtoVIEW) on clinical behaviour of residents and treatment. *Int J Biomed Comput* 1996;42(1-2):143-50.
11. Vissers MC, Hasman A. Building a flexible protocol information system with ready for use' web-technology. *Int J Med Inf* 1999;53(2-3):163-74.
12. East TD, Henderson S, Pace NL, Morris AH, Brunner JX. Knowledge engineering using retrospective review of data: a useful technique or merely data dredging? *Int J Clin Monit Comput* 1991;8(4):259-62.
13. Field MJ, Lohr KN (eds). *Guidelines for Clinical Practice: From Development to Use*. Washington, DC: National Academy Press, 1992.
14. James BC. Making it easy to do it right. *N Engl J Med* 2001;345(13):991-3.
15. Van der Lei J, Talmon JL. Clinical Decision-Support Systems. In: Van Bommel and Musen (eds). *Handbook of medical informatics*. Houten: Bohn Stafleu Van Loghum, 1997.
16. Johnston ME, Langton KB, Haynes RB, Mathieu A. Effects of computer-based clinical decision support systems on clinician performance and patient outcome. A critical appraisal of research. *Ann Intern Med* 1994;120(2):135-42.
17. Purves IN, Sugden B, Booth N, Sowerby M. The PRODIGY project--the iterative development of the release one model. *Proc AMIA Symp* 1999;:359-63.
18. Quaglini S, Stefanelli M, Cavallini A, Micieli G, Fassino C, Mossa C. Guideline-based careflow systems. *Artif Intell Med* 2000;20(1):5-22.
19. Herbert SI, Gordon CJ, Jackson-Smale A, Salis JL. Protocols for clinical care. *Comput Methods Programs Biomed* 1995;48(1-2):21-6.
20. Clayton PD, Pryor TA, Wigertz OB, Hripcsak G. Issues and structures for sharing knowledge among decision-making systems: The 1989 Arden Homestead Retreat. In: Kingsland LC (ed). *Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care*. New York: IEEE Computer Society Press. 1989:116-21.
21. Ohno-Machado L, Gennari JH, Murphy SN, Jain NL, Tu SW, Oliver DE, Pattison-Gordon E, Greenes RA, Shortliffe EH, Barnett GO. The guideline interchange format: a model for representing guidelines. *JAMIA* 1998;5(4):357-72.
22. Fox J, Johns N, Rahmzadeh A. Disseminating medical knowledge: the PROforma approach. *Artif Intell Med* 1998;14:157-81.
23. Shahar Y, Miksch S, Johnson P. The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines. *Artif Intell Med* 1998;14:29-51.
24. Musen M, Tu S, Das A, Shahar Y. EON: A Component-Based Approach to Automation of Protocol-Directed Therapy. *JAMIA* 1996;3:367-88.
25. Gruber TR. A translation approach to portable ontologies. *Knowledge Acquisition* 1993;5(2):199-220.
26. Wang D, Peleg M, Tu SW, Boxwala AA, Greenes RA, Patel VL, Shortliffe EH. Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines: A literature review of guideline representation models. *Int J Med Inf* 2002;68(1-3):59-70.
27. Chandrashekar B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* 1986;1:23-30.
28. Bindels R, De Clercq PA, Winkens RAG, Hasman A. A test ordering system with automated reminders for primary care based on practice guidelines. *Int J Med Inf* 2000;58-59(1):219-33.
29. De Clercq PA, Blom JA, Hasman A, Korsten HHM. Gaston: An architecture for the acquisition and execution of clinical guideline-application tasks. *Med Inform Internet Med* 2000;25(4):247-63.

30. De Clercq PA, Hasman A. Design of a Consumer Health Record for Supporting the Patient-centered Management of Chronic Diseases. *Medinfo* 2001;10(2):1445-9.
31. Van Hyfte DMH, De Clercq PA, Tjandra-Maga TB, Zitman FG, de Vries Robbé PF. Modelling the psychoactive drug selection application domain at the knowledge level. *Proc Belgium-Netherlands Conf on Artificial Intelligence* 1999;:187-8.

CHAPTER 2

APPROACHES FOR CREATING COMPUTER- INTERPRETABLE GUIDELINES THAT FACILITATE DECISION SUPPORT: A REVIEW

Submitted for publication

Paul A. de Clercq
Johannes A. Blom
Hendrikus H.M. Korsten
Arie Hasman

1 Introduction

1.1 Overview

During the last decade, studies have shown the benefits of using clinical guidelines in the practice of medicine [1]. Utilizing guidelines such as standard care plans, critical pathways and protocols in various clinical settings may lead to a reduction of practice variability and patient care costs, while improving patient care [2]. Although the importance of these guidelines is widely recognized, health care organizations typically pay more attention to guideline development than to guideline implementation for routine use in daily care [3]. However, studies have shown that clinicians are often not familiar with written guidelines and do not apply them appropriately during the actual care process [4].

Implementing guidelines in computer-based decision support systems promises to improve the acceptance and application of guidelines in daily practice because the actions and observations of health care workers are monitored and advice is generated whenever a guideline is not followed. Various studies, covering a wide range of clinical settings and tasks, concluded that the use of these systems significantly improves the quality of care, especially when used in combination with clinical information systems such as Electronic Patient Record (EPR) systems [5]. According to the Institute of Medicine (IOM), these decision support systems are in fact crucial elements in long-term strategies for promoting the use of guidelines [6].

Computer-based clinical guidelines are increasingly applied in diverse areas such as policy development, utilization management, education, clinical trials, and workflow facilitation. Many parties are developing computer-based guidelines as well as decision support systems that incorporate these guidelines [7]. The resulting products exhibit much redundancy and overlap since there is little standardization to facilitate sharing or to enable adaptation to local practice settings [8]. Yet considerable progress has been made and standardized approaches for guideline representation and sharing are central to these efforts [9].

This paper reviews approaches for developing and implementing computer-based guidelines that facilitate decision support. The goal of the review is to formulate a set of requirements related to guideline development and implementation that can be used in the process of developing new approaches or updating existing ones. The paper discusses five approaches,

after which a number of requirements are postulated that are based on the evaluation of each approach.

1.2 Methods

Although many fields contribute to the success of providing guideline-based decision support, often the main focus of researchers is on guideline representation and formalization issues. This paper, however, tries to evaluate approaches with the focus on providing decision support. Therefore, it not only addresses guideline representation issues, but also focuses on guideline acquisition, verification and execution.

The approaches that are mentioned in this review are selected, based on a literature search and the knowledge of the authors on existing approaches. Inclusion of a paper into the review was based on the following criteria, First of all, as this paper aims at defining requirements regarding the entire guideline development and implementation process, we selected approaches that each focus on certain aspects of this process (e.g., guideline representation, acquisition, verification or execution). Other criteria are lifetime and number of publications about the approach.

The literature search was conducted using the 'Medline' search engine, combined with proceedings of the AMIA, MEDINFO and MIE conferences, using the keywords 'guidelines', 'approach', 'decision support', 'representation', 'acquisition' and 'execution' in various combinations.

Taking into account the criteria, mentioned above, final inclusion of an approach as a relevant subject in the review was based on our subjective decision. Therefore, although we recognize that a number of other important approaches exist nowadays such as PRODIGY [10], PatMan [11] and DILEMMA [12], we have limited the number of refereed approaches (also to constrain the size of the review) to the following five: The Arden Syntax [13], GLIF [14], PROforma, [15], Asbru [16] and EON [17].

The remaining part of this paper defines a number of relevant areas with respect to the guideline development process, after which the selected approaches are discussed and evaluated. The paper finishes with a general comparison of all approaches, a number of requirements and a discussion.

2 Areas

By analyzing existing literature on representing and implementing computer-based guidelines [18, 14, 19, 20], combined with our own experience in this

field, we identified four areas that can be distinguished in the process of developing guideline-based decision support systems:

- Guideline modeling and representation;
- Guideline acquisition;
- Guideline verification and testing;
- Guideline execution.

For each of these areas, a number of general aspects can be formulated, which will serve as guiding principles in the remaining part of this paper when analyzing the various approaches and formulating the final requirements. The remaining part of this section describes the four areas and their aspects in more detail.

2.1 Guideline modeling and representation

To implement guidelines in computer-based decision support systems, the question how to represent guidelines is a critical issue. A formal and expressive model should provide 1) an in-depth understanding of the clinical procedures, addressed by the guideline, 2) a precise and unambiguous description of the guideline and 3) a means for automatic parsers to execute guidelines to facilitate decision support. A number of representation-related aspects can be formulated to fulfill the above-mentioned goals:

- *Primitives*: The set of building blocks, used to represent the guidelines (e.g., rules, nodes, frames, etc) must be expressive enough to capture the various aspects of a guideline. For example, as time and uncertainty play a very important role in guidelines (especially in complex treatment plans), a guideline representation should support these.
- *Complexity*: The representation must be able to represent various kinds of guidelines that may differ considerably in complexity and level of abstraction, for example by means of nesting or decomposition.
- *Knowledge types*: Guidelines contain a number of different knowledge types such as declarative knowledge (e.g., domain-specific knowledge) and procedural knowledge (e.g., inference or the method of decision support), which should be modeled separately.
- *Didactic and maintenance*: As the content of a guideline is not static but may change over time, the representation must be able store didactic and maintenance information such as author names, versioning information, purposes and detailed explanations.
- *Language*: The representation should be supported by a formal language (vocabulary, syntax and semantics), which has to be expressive enough to

capture all the aspects, mentioned in the above points. In addition, a parser must be able to execute the guidelines in order to provide decision support, which requires a syntax that must meet execution-time requirements such as compactness and execution speed.

2.2 Guideline acquisition

An important issue in the development of guidelines is the knowledge acquisition process. Knowledge Acquisition Tools (KA-Tools) are increasingly used to acquire knowledge directly from a domain expert. These tools may facilitate the knowledge acquisition process by helping domain experts formulate and structure domain knowledge used in guidelines, based on the underlying guideline model. The user interface of a knowledge acquisition tool must facilitate the entry of guidelines that are specific to a target guideline-application domain. Also, an update mechanism (e.g., version control) must be provided as guidelines may change over time.

2.3 Guideline verification and testing

For acceptance of computer-interpretable guidelines in daily clinical practice, guidelines must be unambiguous and syntactically as well as semantically correct. For example, incorrect advice (e.g., false alarms) is to be kept to a minimum. Verification tests may serve such a purpose. These tests include the detection of various types of logical and procedural errors. In addition, testing guidelines in a simulation environment (e.g., testing the guideline using a number of existing patient records) also increases their validity [21].

2.4 Guideline execution

To provide decision support, guidelines must be encoded in a format, interpretable by automatic parsers that are incorporated in guideline execution engines. Guideline execution engines must be optimized to meet execution-time requirements such as compactness and execution speed. Furthermore, the architecture of the guideline execution engine must be system-independent as well as application-independent so that the guideline engine can be used in multiple clinical domains.

3 The Arden Syntax

3.1 Introduction

Named after the Arden Homestead conference center, where the initial meeting was held, the first version of the Arden Syntax was developed in 1989 [13] as a response to the inability to share medical knowledge among different institutions. The Arden Syntax (based on the HELP [22] and RMRS [23] systems) is intended as an open standard for the procedural representation

and sharing of medical knowledge. It defines a representation for modular guidelines: Medical Logic Modules (MLMs) [24]. Each MLM contains a production rule that relates a set of input conditions to a particular set of actions to take. Most MLMs are triggered by clinical events (e.g., admission of a patient, storage of medical data). As a result, a number of logical decision criteria are evaluated, and, if appropriate, an action such as sending a message to a health-care provider is performed. The Arden Syntax focuses on the sharing of 'simple' modular and independent guidelines (e.g., reminders). It is not designed for complex guidelines that for example address treatment protocols. The Arden Syntax was accepted in 1992 as a standard by the American Society for Testing and Materials (ASTM). The current version of the Arden Syntax is Arden 2.0 [25], developed and published by the HL7 group.

3.2 Guideline model and representation

3.2.1 Medical Logic Modules

In the Arden Syntax, each guideline is modeled as a MLM that makes a single decision. Each MLM is an ASCII file, containing slots that are grouped into three categories: *maintenance*, *library* and *knowledge*. The *maintenance* and *library* categories describe the guideline's pragmatics (e.g., title, version, explanation and keywords) and the *knowledge* category describes the logic of an MLM. Figure 1 shows an example of a MLM that warns a health care provider whenever a patient's hematocrit value becomes too low. The remaining part of this section will explain the various parts of the MLM in more detail.

3.2.2 Maintenance and library Slots

As MLMs are to be shared among various institutions, the maintenance and library slots contain necessary documentation for each MLM. As shown in figure 1, *maintenance* slots include the MLM's (file)name, author, version, institution, specialist, date of last modification and validation status. The validation status is intended to document whether the MLM has been approved in a certain local institution. This slot may hold the values '*testing*', '*research*' (approved for clinical research), '*production*' (approved for clinical care) and '*expired*' (no longer in use). When a MLM is shared, the value of the *validation* slot should initially be set to '*testing*', indicating that a receiving institution must approve the MLM for use in clinical care. As MLMs usually require some form of local adaptation before they can be used in a certain institution, changing the value of the *validation* slot to '*production*' implies that the responsibility for the MLM is transferred from the authoring institution to the receiving institution. The name of the person who approves the MLM for

local use is stored in the *specialist* slot. As long as a MLM has not been approved for clinical care, the *specialist* slot has no value.

```

maintenance:

    title: Alert on low hematocrit;;

    filename: low_hematocrit;;

    version: 1.00;;

    institution: CPMC;;

    author: George Hripcsak, M.D. (hripcsa@cucis.columbia.edu);;

    specialist: ;;

    date: 1993-10-31;;

    validation: testing;;

library:

    purpose: Warn provider of new or worsening anemia.;;

    explanation: Whenever a blood count result is obtained, the hematocrit is checked
                  to see whether it is below 30 or at least 5 points below the previous
                  value.;;

    keywords: anemia; hematocrit;;

knowledge:

    type: data-driven;;

    data:

        blood_count_storage := event {'complete blood count'};

        hematocrit := read last {'hematocrit'};

        previous_hct := read last ({'hematocrit'}

            where it occurred before the time of hematocrit);;

    evoke: blood_count_storage;;

    logic:

        /* check that the hematocrit is a valid number */

        if hematocrit is not number then

            conclude false;

        endif;

        if hematocrit <= previous_hct-5 or hematocrit<30 then

            conclude true;

        endif;;

    action:

        write "The patient's hematocrit ("|| hematocrit ||") is low or falling

```

Figure 1: An example of an MLM, Arden Syntax keywords are shown in bold [24]

The slots in the library category are used for documentation and consist of the MLM's purpose, a more detailed explanation (which can for example be

shown to users when they receive MLM-generated messages) and a number of keywords (for example used to categorize MLMs).

3.2.3 Knowledge slots

The actual medical knowledge is stored into the *knowledge* category. This category consists of five mandatory slots (*type*, *data*, *evoke*, *logic* and *action*) and two optional slots (*priority* and *urgency*). Of these slots, the most important ones are *data*, *evoke*, *logic* and *action*.

Data slot

This slot is used to obtain the values of concepts that are mentioned in the MLM from local clinical information systems such as EPRs. For example, the line '*hematocrit* := **read last** {'*hematocrit*};' indicates that the value of the concept '*Hematocrit*' (used in the logical expression of the MLM in figure 1) corresponds to the last hematocrit value in for example an EPR. In practical use, the value of the term between the curly braces has to be acquired from the clinical information system. Similarly, the concept '*Previous_hct*' is defined as the hematocrit value before it. The terms between the curly braces are often institution-specific: the implementation and integration of the actual interface techniques are usually left to the local institutions [26].

Evoke slot

The *evoke* slot specifies the context in which an MLM should be executed. MLMs can be executed as a result of three different types of events: database operations, temporal events and external notifications. The first one is most commonly used. For example, the MLM in figure 1 is executed as a result of the '*blood_count_storage*' event (i.e., whenever a new blood count is added to the system's database). Similarly to the terms in the *data slot*, the terms between the curly braces (e.g., '*complete blood count*') are institution-specific.

Logic slot

The logic slot contains the actual decision criteria that may lead to a certain action. These logical expressions are implemented as production rules and contain concepts that are defined in the *data slot* (e.g., '*Hematocrit*'). The Arden Syntax supports various types of operators such as logical operators (e.g., '*or*', '*and*'), list operators (e.g., '*merge*', '*sort*'), temporal operators ('*after*', '*before*', '*ago*') and aggregation operators ('*sum*', '*average*'). The boolean operators use a three-valued logic, in which the value '*null*' is considered as unknown. Whenever the rule's premise is evaluated '*true*', a particular action that is specified in the *action slot* is carried out. When the premise is evaluated '*false*' or '*null*', the execution of the MLM ends.

Action slot

Once the logical expression evaluates to *'true'*, the *action* slot is executed, performing whatever actions are appropriate to the condition. Typical actions include sending a message to a health care provider, adding an interpretation to the patient record, returning a result to a calling MLM, and evoking other MLMs (nesting). For example, the MLM in figure 1 writes a message to the standard destination, stating that the patient's hematocrit value is low or falling (the `||` operator is a concatenation operator, inserting the actual hematocrit value of the patient into the message). Calling other MLMs is supported in the Arden Syntax by means of the *'call'* statement. Although a MLM can invoke other MLMs, the syntax itself does not support a general control structure to steer these invocations [27].

3.3 Guideline acquisition, verification and testing

Various acquisition tools have been developed to assist guideline authors writing MLMs. Examples include text-based editors where MLMs are typed in as free text, supported by syntax-checkers to improve verification [28] as well as systems that use a controlled vocabulary and 'wizards' to facilitate entering MLMs by unfamiliar users [29, 30].

3.4 Guideline execution

In order to execute MLMs, they have to be translated into a format interpretable by a guideline execution engine. A number of implementations for executing MLMs have been developed, including the use of pseudocode [31], C++ [32], Smalltalk and MUMPS. As the Arden Syntax leaves the implementation of patient data modeling entirely up to the local institutions, there are no standard mapping facilities to obtain values of required patient data during guideline execution.

User comments were collected and analyzed over a period of 26 months regarding the system that was in use at the Columbia-Presbyterian Medical Center. In this period, a total of 126 comments were made by health care providers. The majority of the given comments indicated that the messages were actually or at least potentially useful, although a minority indicated that they were unhelpful or actually harmful (a more detailed explanation is provided elsewhere [33]).

4 The GuideLine Interchange Format (GLIF)

4.1 Introduction

The GuideLine Interchange Format (GLIF) was developed to model guidelines in terms of a flowchart that consists of structured scheduling steps,

representing clinical actions and decisions. GLIF was developed by the Intermed Collaboratory [34] including researchers at Columbia University, Harvard University and Stanford University and was first published in 1998 [14]. The intended purpose of GLIF is to facilitate sharing of guidelines between various institutions by modeling guidelines in such a manner that the guidelines are understandable by human experts as well as by automatic parsers used in different clinical decision support systems. GLIF is an object-oriented representation, consisting of a set of classes that describe characteristic guideline entities (e.g., actions and decisions), attributes for those classes and data types for the attribute values.

In the first published version of GLIF (known as GLIF2), most of the attributes were text strings that were not easily interpretable by parsers. Although GLIF2 facilitated the description of more complex guidelines than for example the Arden Syntax did, it still had a number of deficiencies making it difficult to implement GLIF guidelines in decision support systems. As recognized by the current developers of GLIF [35], the model needed improvement in a number of areas. First, important attributes of guideline steps (e.g., criteria) needed to be specified more formally (instead of being described by means of text strings). Also, GLIF2 had no constructs that formally allowed the mapping of (patient) data elements in the guideline onto elements that are used in clinical systems such as Electronic Patient Record (EPR) systems, which made it difficult to incorporate GLIF guidelines in decision support systems, which are able to interact with EPRs in a generic way. Furthermore, the number of constructs in GLIF2 was rather limited, constructs that supported for example alternative decisions, iterations, (patient) states, exceptions and events were lacking. These issues have been addressed and have recently resulted in the development of a new version (GLIF version 3) [35], which is discussed in the remaining part of this section.

4.2 Guideline model and representation

4.2.1 Guideline steps

GLIF originated from combining a number of relevant features that were determined from an analysis of the characteristics of a number of existing guideline formalisms: 1) the earlier-mentioned Arden Syntax [13], 2) GEODE-CM, a system that combines guidelines with structured patient data entry and data retrieval from a clinical database [36], 3) MBTA, an architecture for building large knowledge-based medical systems, focused on providing reminders [37] and 4) EON, a component-based architecture for building decision support systems for supporting guideline-based care [17].

The GLIF model is object-oriented and consists of a number of classes that describe typical guideline characteristics (e.g., decisions and actions), attributes of those classes and data types for attribute values. In GLIF version 3, all classes, attributes and relations are described by means of Unified Modeling Language (UML) class diagrams [38].

A guideline, encoded in GLIF consists of a flowchart of guideline entities in which each entity is an instance of one of the above-mentioned classes. Figure 2 shows the main classes that are defined in GLIF version 3.

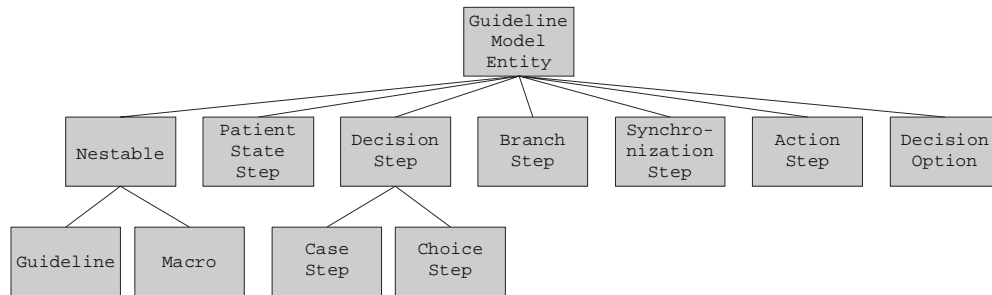


Figure 2: Overview of the main classes in GLIF version 3

The *Guideline* object encapsulates a (sub)guideline. This object contains a number of attributes that are administrative in nature (e.g., name and author) but also attributes that describe the capabilities of a guideline (e.g., the guideline's intention and eligibility criteria). A GLIF guideline consists of a collection of steps that are linked together in a directed graph (flowchart). GLIF defines five steps: decision steps, patient state steps, branch steps, synchronization steps and action steps.

Decision steps

Decision steps model decision points in a guideline and direct flow control from one guideline step to various alternatives. A case step is a decision step that contains a number of decision options, which are logical expressions (e.g., is the patient older than 12 year). Based on the outcome, the guideline flow is directed to the various alternatives (e.g., if the patient is younger than 12, then prescribe a pediatric dosage). Each decision option is expressed through a formal expression syntax (referred to as the Guideline Expression Language or GEL [39]), which is a superset of the Arden Syntax.

Another type of decision is the choice step. Choice steps represent situations where a guideline suggests preferences, but leaves the actual choice to an external agent. Similar to case steps, choice steps also contain a number of decision options that are linked to various alternative guideline steps.

However, the actual choice concerning which alternative is chosen is made by an external agent such as a user or an external software program.

Patient state steps

A patient state step serves two purposes. One purpose is to serve as a label that describes the current patient state that is achieved by means of previous steps. The other purpose is to serve as an entry point in the guideline, depending on the current patient's state (e.g., the patient revisits a family practitioner with a high blood pressure). Each patient state contains attributes that describe the state of the patient (e.g., the blood pressure is higher than 140/90 during the last week). Whenever this state occurs in practice, the guideline that contains the corresponding Patient state step is executed.

Branch and synchronization steps

Branch steps model a set of concurrent steps by directing flow to multiple parallel guideline steps and are used in conjunction with synchronization steps. Multiple guideline steps that follow a branch step always eventually converge in a corresponding synchronization step. When a branch that started at a preceding branch step reaches the corresponding synchronization step, a continuation attribute specifies whether all, some, or one of the preceding steps must have been completed before control can move to the next step. The continuation attribute is expressed as a logical expression.

Action steps

Actions steps model actions that are (or should be) performed in a guideline. Each action step defines a number of tasks that formally describe the actual tasks that are to be carried out. Three types of tasks are defined: 1) medically oriented actions such as a recommendation for a particular course of treatment, 2) programming-oriented actions such as retrieving data from an electronic patient record or supplying a message to a care provider, and 3) control-oriented actions that invoke nested structures such as (sub)guidelines or macros to support recursive specification. Similar to the use of macros in conventional programming languages, a macro provides a means for defining information needed to instantiate a predetermined set of steps. For example, GLIF defines an MLM-macro, which can be used to define a MLM. Internally, the macro consists of two steps: a decision step and an action step.

4.2.2 Medical ontology

Similar to the Arden Syntax, logical expressions and action specifications in GLIF contain references to actual patient data item values (e.g., the age of a patient) and clinical concepts (e.g., antibiotic, amoxicillin), which have to be

acquired during guideline execution from patient information systems such as EPRs. In order to facilitate sharing of guidelines among different institutions, GLIF aims at defining the structure of these patient data elements and medical concepts in accordance with standard data models and medical terminologies such as HL-7's Reference Information Model (RIM, also known as the Unified Service Action Model or USAM) [40] and the Unified Medical Language System (UMLS) [41]. GLIF divides mapping-related information into three layers: the core GLIF layer, the Reference Information Model (RIM) layer, and the Medical Knowledge layer.

The first layer, core GLIF, is part of the GLIF specification language and defines a number of elementary data items and relations that are used as variables in the guidelines. It does not contain information on how these items will be mapped to corresponding items in clinical information systems. For example, core GLIF defines a *Data_Item* class that represents concepts such as amoxicillin. Each *Data_Item* class contains a *name* attribute that specifies the name of the concept. Whenever guideline authors want to use expressions such as 'is amoxicillin being prescribed for more than a week', they refer to a *Data_Item* class that represents the concept amoxicillin without having to know where this information is stored in a clinical information system.

Internally, each *Data_Item* class contains a reference to a corresponding object in the RIM layer (second layer). The RIM layer provides a semantic hierarchy of medical concepts and attributes. Although different RIMs may be used, GLIF by default relies on the HL-7 RIM [40]. For example, this RIM defines several general classes such as *Medication*, *Observation* and *Procedure*, which represent medication, observations (e.g., diagnoses) and procedures (e.g., treatments). The *Medication* class contains attributes such as *Dosage_quantity*, *Doseform* and *Route*, which represent the quantity (e.g., 1000 mg), form (e.g., capsule) and route (e.g., intravenous) of the medication. Concepts in the RIM layer are linked (although not automatically) to data items in the core GLIF layer. Therefore, whenever guideline authors want to refer to concepts from the core GLIF layer (for example, when defining criteria in a Decision step), they can specify values for each attribute of the corresponding class in the RIM.

The Medical Knowledge layer (third layer) specifies the methods needed to interface with various medical knowledge sources and other information systems such as controlled terminologies (e.g., UMLS), knowledge bases and clinical information systems (e.g., EPRs). This layer will contain the information

to integrate developed guidelines with institution-specific information systems. However, this layer is still under development.

4.2.3 Guideline representation

Each guideline in GLIF consists of a set of nodes linked together in a temporally sequenced graph (flowchart), in which each node corresponds to an instance of one of the five classes. Figure 3 shows a graphical representation of a GLIF guideline concerning a simple vaccination guideline stating that children under 12 years should receive a pediatric dosage of a certain vaccine whereas health care workers or adults above 65 years should receive an adult dosage of the vaccine.

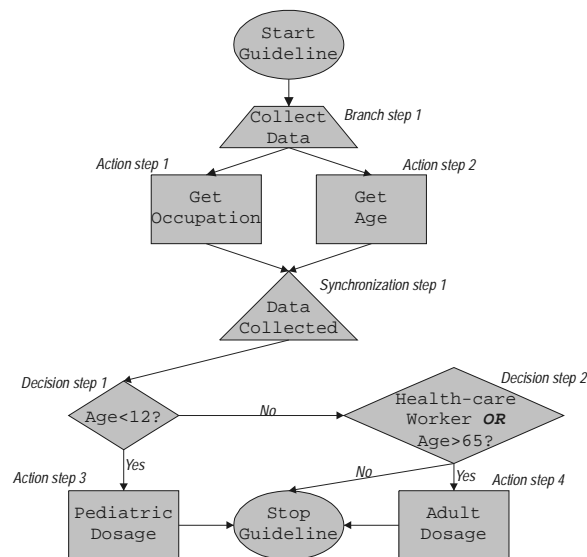


Figure 3: Graphical representation of a guideline in GLIF [14]

In order for guidelines to be 1) readable by humans, 2) interpretable by computers and 3) adaptable by different (local) institutions [8], GLIF allows for a specification of a guideline at three levels of abstraction: the conceptual level (level A), the computable level (level B) and the implementable level (level C).

The highest level is the conceptual level where guidelines are represented as flowcharts, which can be viewed by humans (e.g., guideline authors) but are not interpretable by decision support systems. At this stage, details such as the expression syntax, the contents of patient data elements, clinical actions and guideline flow are not formally specified. These specifications take place at the computable level, which then also allows for various verification checks of the guidelines such as logical consistency and completeness. Finally, at the implementable level, guidelines can be custom-tailored to particular

institutional information systems. At this stage, institution-specific procedures and mappings (which are usually non-sharable) are specified, among other things using the above-mentioned Medical Knowledge layer. Both the Medical Knowledge and the implementable layer are still under development.

4.2.4 Language

In GLIF2, guidelines were written as text in an existing language called ODIF (Object Data Interchange Format), which is a formal representation to represent objects and instances of objects in a text-like manner [14, 42]. In GLIF version 3, this syntax has been replaced with an XML-based syntax [43]. Figure 4 shows a small portion of a vaccine guideline in the XML-based syntax.

```
<a:Guideline rdf:about="%a;Vaccine_INSTANCE_00001">
  <a:name>Guideline for Vaccine X</a:name>
  <a:intention>Decide whether to recommend the Generic vaccine and at what dosage</a:intention>
  <a:algorithm>Vaccine_INSTANCE_00002</a:algorithm>
</a:Guideline>

<a:Algorithm rdf:about="%a;Vaccine_INSTANCE_00002">
  <a:first_step>Vaccine_INSTANCE_00003</a:first_step>
  <a:steps>
    Vaccine_INSTANCE_00003,Vaccine_INSTANCE_00004,Vaccine_INSTANCE_00005,Vaccine_INSTANCE_00006,
    Vaccine_INSTANCE_00007,Vaccine_INSTANCE_00008,Vaccine_INSTANCE_00009,Vaccine_INSTANCE_00010
  </a:steps>
</a:Algorithm>
```

Figure 4: A portion of the vaccine guideline in XML

The upper part shows an instance of the *Guideline* class, together with values for the name and intention attributes. In addition, the *Guideline* class in GLIF defines an *Algorithm* attribute, which contains a reference to another instance. The latter contains references to the first step and all actual steps present in the guideline. In GLIF, every instance is identified by means of an ID. For example, 'Vaccine_Instance_00001' refers an instance of the *Guideline* class and 'Vaccine_Instance_00002' refers to an instance of the *Algorithm* class. Instances 'Vaccine_Instance_00003' to 'Vaccine_Instance_00010' refer to the various steps in the vaccine guideline, not shown here (the contents of the *steps* attribute does not contain information about the sequence of the various steps, which is modeled in the attributes of the step instances themselves).

As mentioned earlier, GLIF defines the GEL formal expression language that is based on the expression grammar of the Arden Syntax. This language has been adapted in order that references to concepts and attributes from the core GLIF model are included in the grammar.

The GLIF model, representation and syntax are still under development. Currently, a variety of guidelines [44-46] are being specified in order to evaluate the various aspects of GLIF such as its three-level model and the

medical ontology. Also, GLIF intends to address other subjects such as guideline goals, probabilistic models and patient preferences.

4.2.5 Modeling tools

Currently, two tools are used to develop the GLIF model in terms of classes and attributes: Protégé [47] and GEODE [48]. These tools are also used for creating the relations between the core GLIF items and the concepts from the RIM and Medical Knowledge Layer. For example, figure 5 shows an example of a *Data_Item* from the core GLIF model.

Figure 5: Definition of the cough *Data_Item* in Protégé

In this case, the concept cough is entered as a *Data_Item* in Protégé. The form in the background shows the cough *Data_Item*, which is known by its name (e.g., *Cough*) to guideline authors. This item is linked to the *Observation* class of the HL-7 RIM or USAM (specified in the *data Model Source ID* attribute). As a result, the cough item receives the attributes that correspond to the *Observation* item in the USAM. In addition, the cough item is also linked to a concept in the UMLS terminology where it has the code 'C0010200'. This is shown in the foreground form that is brought up whenever a user double-clicks on the *Concept* item in the background form (the name *Cough* is used twice in this example: as the name that identifies the *Cough Data_Item* but also as the name of the *Cough* concept in the UMLS terminology). Whenever a RIM or a controlled terminology is not available or necessary (for example, when a guideline is solely created for viewing purposes) only the *Name* attribute in the cough *Data_Item* is filled in, leaving the *Data Model Class ID*, *Data Model Source ID* and *Concept* attributes blank.

4.3 Guideline acquisition, verification and testing

Besides model development tools, Protégé and GEODE are also used as knowledge acquisition tools to facilitate the entering of guidelines. Both tools visualize GLIF guidelines by means of flowcharts. For example, figure 6 shows part of a cough treatment guideline entered by means of the Protégé knowledge acquisition tool.

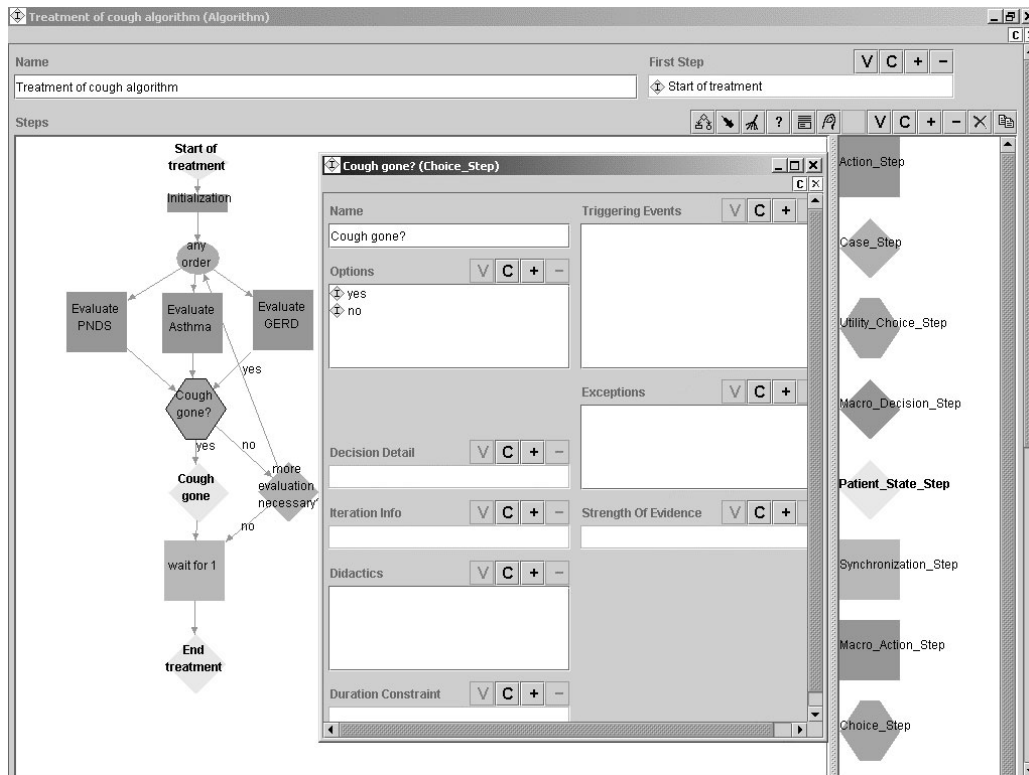


Figure 6: Part of a GLIF cough treatment guideline as a flowchart in Protégé

The left pane shows a graphical overview of the guideline in terms of a flowchart. The right pane shows an overview of all available guideline steps (e.g., *Action step*, *Patient state step*, etc). All objects in the right pane can be selected and dropped onto the left pane, thus creating the flowchart (level A). Selecting a step in the flowchart brings up a form in which the details of that step can be filled in (level B). For example, in figure 6, the *Cough gone* step has been selected which is a *Choice step*. As a result, the various alternatives and decision options can be specified (these are stored in the *Options* attribute). As shown in figure 6, each choice step contains more attributes which will not be explained further here. Protégé is a very generic knowledge acquisition tool that shows instances (e.g., actions or decisions) as forms, in which it is possible to assign values to each attribute (e.g., *Name*, *Options*, *Didactics*, etc) as shown in figure 6.

The introduction of a RIM combined with an expression syntax allows for performing verification tests, as patient data elements, logical criteria and control flow are formally defined. Although there are currently no tools available that are able to perform such tasks, GLIF researchers have indicated that verification tools are currently under development.

4.4 Guideline execution

For GLIF2, efforts have been undertaken to develop guideline execution engines such as the Partners Computerized Algorithm Processor and Editor (P-CAPE) tool [49] and a generic GLIF2 execution engine [50]. As GLIF version 3 is still under development (especially the medical ontology layer and the implementable level), GLIF version 3 guideline execution engines are still under construction. The most recent development is the GuideLine Execution Engine (GLEE), which is able to execute GLIF-encoded guidelines and can be integrated into the clinical information system of a local institution [51].

5 PROforma

5.1 Introduction

PROforma is a knowledge composition language supported by acquisition and execution tools with the goal of supporting guideline dissemination in the form of expert systems that assist patient care through active decision support and workflow management [15]. PROforma was developed at the Imperial Cancer Research Fund by John Fox and colleagues and aims at the development of reliable expert systems that assist patient care through active decision support and workflow management. The name PROforma is a concatenation of the terms proxy ('authorized to act for another') and formalize ('give definite form to').

5.2 Guideline model and representation

5.2.1 The domino model

PROforma addresses two aspects of the guideline development and implementation process. First, it defines an abstract model that represents the general clinical decision making process. This 'domino' model is shown in figure 7.

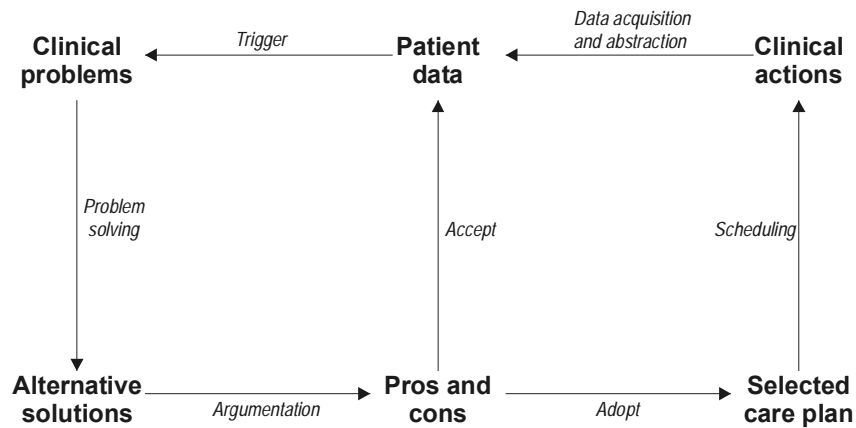


Figure 7: The PROforma clinical process domino model

This model is intended as a framework, which can be applied to specific domains by defining domain-specific knowledge (e.g., rules) needed to instantiate the kinds of inference, needed for that particular domain. Each node represents a certain clinical situation. Arrows refer to inference procedures that result in a transfer from one situation to another. The model assumes that a trigger may lead to the recognition of some kind of clinical problem, which requires a solution. The next step is to apply some kind of problem solving to identify possible solutions to the problem. These possible solutions are then evaluated to determine their strengths and weaknesses. Based upon the outcome of the evaluation, a care provider can decide to adopt a certain solution by selecting the corresponding care plan. Alternatively, a care provider can decide that additional information (for example, based on new patient data) is required to select the most favorable solution. Once a care plan has been adopted, the sequence of clinical actions, needed to execute the plan is scheduled and carried out. Finally, executing a care plan may involve new clinical actions that require additional clinical patient data such as relevant symptoms and additional lab data.

Based on the complexity of entered guidelines, a specific PROforma application may instantiate the entire domino model or only a section of it. An example of an instantiated British Thoracic Society (BTS) acute asthma management guideline in terms of the domino model is shown in figure 8. According to this guideline, the first clinical task on arrival of an asthma patient is to assess the severity of the patient's condition, for which there are four alternatives (mild, moderate, severe and life-threatening). The guideline recommends the collection of relevant data to permit the classification and proposes an appropriate decision if the BTS criteria are satisfied (the decision whether or not to accept the proposal is left to the physician). The level of severity determines the appropriate treatment routine, consisting of various

clinical tasks such as prescribing drugs, recording patient response and reviewing the patient's condition. If the patient deteriorates during this process, a rescue or other action may be triggered, requiring the patient to be admitted into a hospital, which may also lead to the collection of additional patient data as well as an adaptation of the treatment regime.

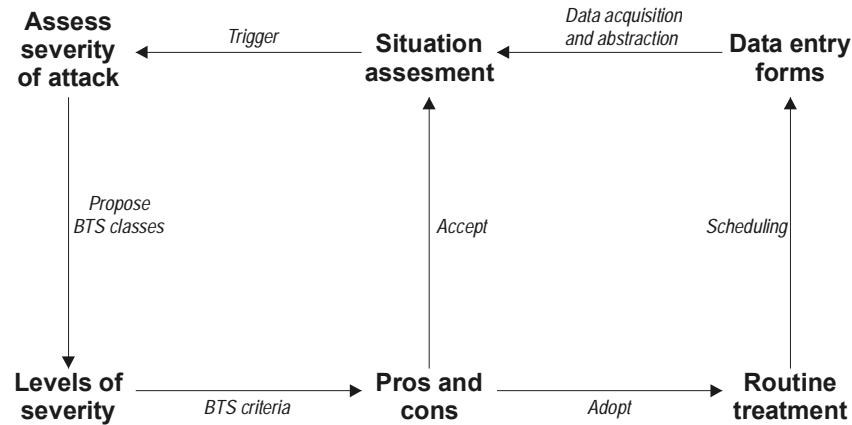


Figure 8: A BTS acute asthma management guideline, embedded in the domino model

Other examples of guidelines developed in terms of the PROforma domino model, include guidelines for drug prescribing and risk assessment and management [52].

5.2.2 The task ontology

In order to represent the domino model in terms of a formal language, PROforma defines a task ontology that contains a number of concepts, named tasks that are used to build guidelines (similar to the guideline steps in GLIF). Each guideline in PROforma is modeled as a plan that consists of a sequence of tasks. The PROforma task ontology defines four task classes, each with their own attributes: 1) plans 2) decisions, 3) actions and 4) enquiries (figure 9).

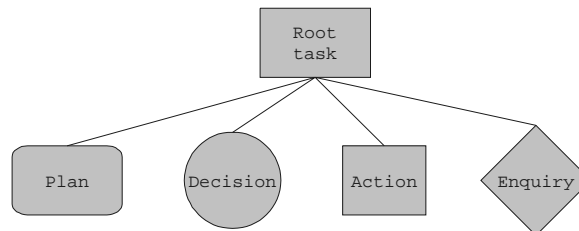


Figure 9: The PROforma task ontology

Root task

All tasks are derived from the root task. The root task contains a number of attributes that are common to all four derived tasks. These include administrative ones that hold a name, caption or description but also attributes that describe the capabilities of a task such as goals (e.g., *'achieve(normal_respiration)'*), pre- en postconditions (e.g., *'risk_level=severe'*), trigger conditions (e.g., *'peak_flow < 30'*) and cycles (e.g., *'cycle(integer, interval)'*).

Plans

Each plan models a (sub)guideline. Plans define 1) an ordered sequence of tasks, 2) logical and temporal constraints on their enactment and 3) circumstances in which a plan must be aborted or terminated (e.g., exceptions). Besides the common attributes that are defined in the root task, the plan task contains additional attributes such as *Components*, *Scheduling* and *Temporal constraints* and *Abort or Termination conditions*. The *Components* attribute is a container that holds a set of task instances, similar to the *Steps* attribute of the *Algorithm* class in GLIF. For example, a guideline that consists of 4 task instances (e.g., *'history'*, *'diagnosis'*, *'therapy'*, *'follow-up'*) is modeled through a *Plan* instance of which the *Components* attribute contains references to those four task instances.

The ordering between these task instances is defined by means of two sorts of constraints: scheduling constraints and temporal constraints. Scheduling constraints order tasks in a plan by means of qualitative conditions (e.g., the *'history'* task is executed *'before'* the *'diagnosis'* task). Temporal constraints order tasks by using temporal conditions (e.g., the *'follow-up'* task is executed *'after a period of ten weeks'*). By using these two types of constraints, tasks in a plan are not modeled as traditional flowcharts that order guideline elements usually only through scheduling constraints.

Another way of directing guideline flow in PROforma is through abort or termination conditions. Each PROforma task passes through a number of states such as *'dormant'*, *'in progress'*, *'aborted'*, *'terminated'* and *'performed'*. Every task is initially in a *'dormant'* state. Executing a certain task changes its state from *'dormant'* to *'in progress'*. Whenever a task is finished normally, the task's state becomes *'performed'*. However, it is possible to force the termination or abortion of a plan by means of the abort and termination conditions. For example, a plan that manages the treatment of hypertension aims at lowering the blood pressure to a normal value. When this plan will finish normally, the plan's postcondition (for example: *'BP=normal'*) will be true. However, whenever the blood pressure of a certain patient reaches a

normal level while the plan is still in progress, the treatment should be terminated earlier and the postcondition will be deemed to hold. However, when for some reason the blood pressure is falling rapidly, the treatment must be aborted and the postcondition will be false. In this plan, the termination condition could be defined as '*BP=normal*', whereas the abort condition could be defined as '*trend(BP)=falling*'. The values of the pre- and postconditions are used for example as trigger conditions that will activate other tasks.

Decisions

A decision is a task that represents a decision in a guideline about for example a choice of investigations, diagnoses or therapies. A decision consists of a set of possible outcome candidates plus various types of schemas (logical expressions) that support or oppose each candidate. The *Decision* class contains an attribute, which contains a list of all possible candidates. For example, a decision that addresses administering the right drug regarding liver diseases may contain a number of candidates, each candidate suggesting the prescription of a different drug.

Every candidate is associated with a set of schemas. Schemas consist of rules, qualitative symbols, quantitative weightings and certainty factors [53] and support (+) or oppose (-) candidates, establishing a preference order among them. For example, the fact that a patient is diagnosed with oesophagitis, combined with the fact that (s)he has no liver disease supports the prescription of cimetidine. This can be translated into an argument schema: '*diagnosis = oesophagitis and liver_disease = absent then cimetidine: +*'. Besides schemas, decisions also include mandatory data constraints. These state that certain data (e.g., '*presence of liver_disease*') has to be available before a decision can be taken.

Actions

An action is a task that a PROforma execution engine can request for enactment by an external agent (e.g., a clinical user or an external software program or hardware device). Such an action in PROforma usually exists of issuing a message to a user or calling an external program through a predefined Application Programming Interface (API). Examples are "*give ibuprofen, 10 mg*" that shows a message to a clinical user or '*call(print(leaflet1))*' that executes an external procedure in order to print a leaflet. In PROforma, actions are always atomic and are not decomposable.

Enquiries

Enquiries are used to acquire various kinds of information, such as clinical or administrative information. This information can be obtained from a clinical

user or can be directly extracted from an external software agent or hardware device (e.g., EPR or patient monitor). Therefore, as was the case with the definition of an action, the *Enquiry* class contains attributes that define the method of data retrieval such as '*use_form(height_and_weight)*', which refers to a procedure that provides a means for a clinical user to enter the patient's height and weight by showing a form. The *Enquiry* class also contains attributes that store information of the acquired variable such as its type (e.g., '*type=integer*') and allowed values (e.g., '*range=[100:230]*').

5.2.3 Guideline representation

Similar to GLIF, a guideline in PROforma is represented as a directed graph in which the nodes are task instances. Figure 10 shows an example of a guideline in terms of instances of plans (rounded rectangles), decisions (circles), actions (square rectangles) and enquiries (diamonds).

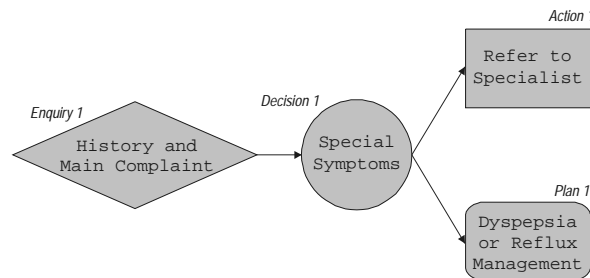


Figure 10: A guideline in terms of task instances

This guideline addresses the treatment of Dyspepsia. First, the patient's history and main complaint are acquired, after which the decision is made whether the patient must be referred or that the patient must be treated for dyspepsia or reflux (the '*dyspepsia or reflux management*' plan is a subguideline that contains tasks that describe the treatment of dyspepsia or reflux).

PROforma contains temporal as well as scheduling constraints. Therefore, the arrows in figure 10 may represent both these constraints and merely state that there is some kind of relationship between linked concepts.

5.2.4 Language

Guidelines in PROforma are stored (in terms of instances of task classes) using the Red Representation Language (R²L), a time-oriented knowledge representation language [54]. A guideline, written in R²L, is a declarative specification of tasks and their (inter)relationships organized in a hierarchy of plans and their components. An example of a PROforma guideline in R²L is shown in figure 11.


```

plan :: Protocol1 ;
  caption :: 'Management of weight loss (simplified)' ;
  precondition :: problem = weight_loss ;
  goal :: clinical_goal = manage : weight_loss ;
  component :: enquiry1 ;
  Component :: decision1 ;
    schedule_constraint :: completed(enquiry1) ;
  Component :: decision2 ;
    schedule_constraint :: completed(decision1) ;
  component :: plan1 ;
    schedule_constraint :: completed(decision2) ;
  Component :: plan2 ;
    schedule_constraint :: completed(decision2) ;
  component :: plan3 ;
    schedule_constraint :: completed(decision1) ;
end plan .

decision :: decision1 ;
  caption :: 'Diagnosis?' ;
  goal :: goal = manage : cancer ;
  source ::
    age; mandatory :: yes ;
    smoker; mandatory :: yes ;
    biopsy; mandatory :: yes ;
    pain: site; mandatory :: yes ;
    pain: time; mandatory :: yes ;
  choice_mode :: single ;
  support_mode :: symbolic ;
  candidate :: cancer ;
    argument :: ( age = elderly) + ;
    argument :: ( smoker = yes) + ;
    argument :: ( biopsy = positive) + ;
    argument :: ( pain: time = immediate) + ;
    argument :: ( pain: site = epigastric) + ;
    recommendation ::
      netsupport( decision1, cancer) >= 1 ;
  candidate :: peptic_ulcer ;
    argument :: ( age = young or age = adult) + ;
    argument :: ( biopsy = negative) + ;
    argument :: ( pain: site = epigastric) + ;
    argument :: ( pain: time = delayed) + ;
    recommendation ::
      netsupport( decision1, peptic_ulcer) >= 1 ;
end decision .

```

Figure 11: A part of a guideline in R²L [52]

Before execution, guidelines in the R²L language are translated into another language, called L_{R²L} ('Logic of R²L'), a language based on predicate logic. This language is used as input for the verification and execution modules (explained in the next sections).

5.3 Guideline acquisition

PROforma contains a number of tools to develop guidelines [55]. The PROforma task authoring environment enables guideline authors to define guidelines in terms of class instances (and attributes) of the task ontology. Figure 12 shows a part of a treatment protocol that has been entered in the task authoring environment.

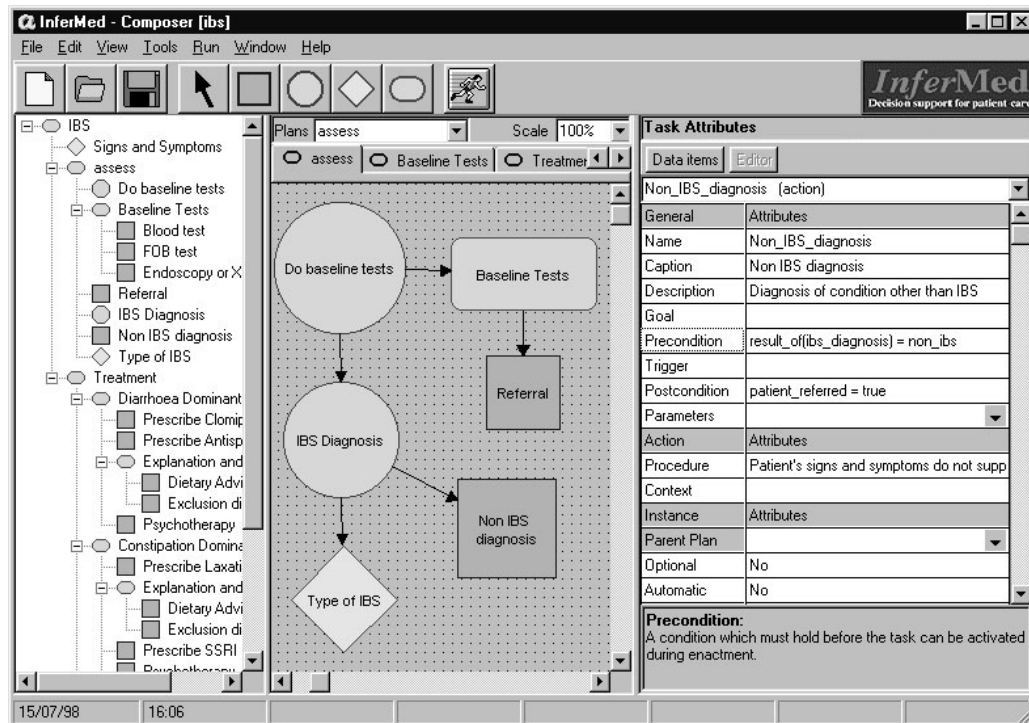


Figure 12: A part of a guideline, entered in the task authoring environment

The left pane of the task authoring environment shows a treelike overview of all (sub)plans that a guideline contains, whereas the middle pane shows a graphical representation of the currently selected plan. When a task is selected in the tree, the right pane shows its attributes. In this case, the 'non_IBS_diagnosis' action is chosen, which results in showing the relevant attributes such as common attributes like *Name*, *Caption*, *Goal* and *Conditions* as well as action-specific attributes such as *Procedure* (the content of the *Procedure* attribute holds in this case a text that is shown to the user during guideline execution). Entering guidelines in PROforma is a two-phased process. First, a graphical layout of the plan is specified in terms of instances of the four tasks, without entering attribute-specific values. The latter is done in the second phase where for each instance its attributes are filled in.

5.4 Guideline verification and testing

A major focus point of the PROforma approach is to increase the safety of guidelines. Unsafe situations may occur as a result of incorrect or incomplete knowledge as well as incorrect or incomplete reasoning strategies. In order to address these problems, the PROforma researchers developed a life cycle for the engineering of knowledge base systems [56].

In this lifecycle, guidelines that are acquired by means of the PROforma task authoring environment are stored in the R²L language, after which they are

processed by a verification tool to detect errors that are declarative in nature such as incorrect data types, invalid syntax, missing task values (e.g., missing candidates or decision rules), inconsistent data references and inconsistent scheduling or temporal constraints.

Guidelines are then translated into the L_{R2L} language. For example, figure 13 shows an L_{R2L} equivalent of a decision rule that states that '*intermittent nausea caused by the drug cisplatin lasts at least 20 hours and takes effect within 6 hours*'.

$$\forall t_1, \forall t_2 (\text{cisplatin}[t_1, t_2] \rightarrow \text{intermittent_nausea}[[t_2 + 6, t_2 + 26]])$$

Figure 13: A decision rule, translated in L_{R2L}

The guidelines in L_{R2L} are then processed by a PROLOG-like interpreter in the PROforma execution engine, which is embedded in a test environment. In this environment, users are able to view and evaluate guidelines (an example of the user interface of the execution engine is shown in figure 14 in the next section). After a certain test period, guidelines can be updated through the task authoring environment or transferred to the execution engine used in daily practice.

PROforma also defines an extension of the L_{R2L} language, called L_{safe} , which defines additional safety-related operators such as integrity and safety constraints [57].

5.5 Guideline execution

As mentioned in the previous section, the PROforma framework also contains a standard execution engine that executes entered guidelines by parsing and interpreting a L_{R2L} task definition. The execution engine is able to directly read and execute guidelines and can be interfaced through the API to various interfaces. Figure 14 shows an example of the PROforma execution engine user interface. The engine executes tasks according to a control regime in which tasks pass through a sequence of states (e.g., '*in progress*', '*terminated*' or '*abandoned*'), in which the sequence is determined by situations that are encountered by the system. When required, the engine collects information (e.g., from clinical users or external devices) and takes actions (e.g., sending a message).

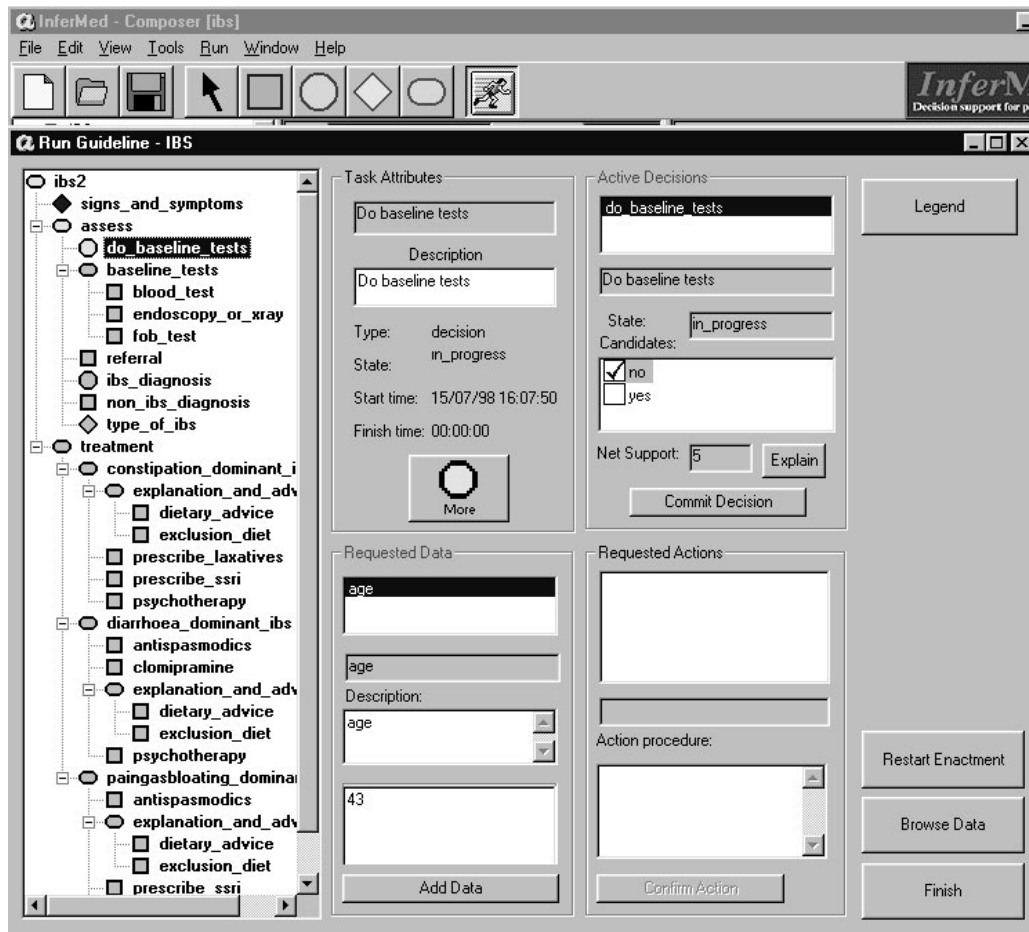


Figure 14: Execution of a guideline in the PROforma execution engine

The left pane shows an overview of all tasks and their current state, indicated by different colors of the task's icon. For example, the icon of the 'signs_and_symptoms' enquiry is blue, which indicates that a task is 'performed'. Furthermore, the icons of the 'assess' plan task and 'do_baseline_tests' decision task are yellow, indicating that these tasks are currently 'in progress'. Finally, the icons of all other tasks are gray, meaning that these tasks are still in the 'dormant' state.

The right pane shows a more detailed overview of the currently executed task. In this case, the 'signs_and_symptoms' enquiry task has been completed, resulting in the execution of the 'do_baseline_tests' decision task. Based on already known patient data (for example the patient's age, which is already filled in by the user), two possible candidates ('yes' or 'no') are shown, of which 'no' is recommended by the system, based on the currently evaluated schemas of this particular decision task. Before continuation, the user first has to commit to the decision.

Various decision support systems were developed and implemented using the *PROforma* approach. Also, a commercial version of *PROforma*, named *Arezzo*, has been developed by *InferMed* Ltd. Examples of developed decision support systems can be found there [58].

6 Asbru

6.1 Introduction

Asbru is a guideline representation formalism, developed at Stanford University and the Vienna University of Technology and is part of the Asgaard project [16], which focuses on the application and critiquing of time-oriented clinical guidelines. The Asbru language [59] is a plan representation language that represents clinical guidelines as time-oriented skeletal plans, which are plan schemata at various levels of detail. In order to manage these (often complex) skeletal plans, key aspects of Asbru are the representation of high-level goals (intentions), the representation of temporal patterns and time annotations, and the development of user interfaces to visualize developed plans.

6.2 Guideline model and representation

6.2.1 The intention-based model

Asbru uses an intention-based model to represent clinical guidelines as skeletal plans. Similar to the notion of plans in *PROforma*, a plan is a collection of other items. The Asbru model identifies a number of general tasks, which have to be carried out during the process of acquiring, testing and executing guidelines. Examples of these tasks are guideline verification and validation, applicability, execution, recognition and critiquing. Each task is performed by means of Problem-Solving Methods (PSMs), which are generic strategies to solve stereotypical tasks, independent of the system's application domain [60]. The knowledge needed to solve a certain task is defined by means of knowledge roles, which give an abstract description of the function domain knowledge has to play in a PSM. Knowledge roles are specified by a guideline author during guideline acquisition. The Asbru language introduces the following knowledge roles: preferences, plan intentions, conditions, effects and a plan body. In Asbru, the content of a plan (the plan body) always consists of other plans, until a plan is no longer decomposable. The latter is referred to as an action. In Asbru, guidelines entirely consist of plans and actions. The functionality of each plan is modeled by means of a number of knowledge roles. This, in contrast to other approaches where the functionality of a guideline is described in terms of its primitives such as enquiries, decisions, actions (*PROforma*), decision steps, action steps and choice steps

(GLIF). Asbru defines the following knowledge roles, which are part of each plan:

Preferences

Preferences bias or constrain the applicability of a plan to achieve a certain goal. Examples of preferences are 1) '*select-method*', a matching heuristic to determine the applicability of the entire plan (e.g., '*exact-fit*' or '*roughly-fit*'), 2) '*resources*', a specification of forbidden or obligatory resources (e.g., in certain cases of a pulmonary infection treatment, surgery is prohibited and antibiotics must be used), and 3) the applied '*strategy*' (e.g., '*aggressive*' or '*normal*').

Intentions

One of the key aspects of Asbru is the representation of intentions of a plan: high level goals at various levels of a plan. Besides aiding in the selection of the most appropriate plan, intentions are primarily used in the process of providing decision support. For example, in a guideline for the treatment of hypertension, one possible course of action may be the prescription of beta-blockers in order to lower the blood pressure. However, it is possible that a physician for some reason decides not to use beta-blockers, but aims at lowering the blood pressure in another way. Although the physician follows the plan's intentions, (s)he technically does not follow it, so a guideline execution program that monitors the physician's actions may critique the physician that (s)he is not following the plan. However, if the guideline execution program recognizes from the plan's intentions that its goal has been reached it will not generate a critique, which will improve the acceptance of the system.

Intentions are defined as temporal patterns of provider action and patient states that must be maintained, achieved or avoided. Four categories of intentions are defined:

1. *Intermediate state*: the patient states that must be maintained, achieved or avoided (e.g., weight gain levels of slightly low to slightly high).
2. *Intermediate action*: the provider actions that should take place during the execution of the plan (e.g., monitor blood glucose one a day).
3. *Overall state pattern*: the overall pattern of a patient state that should hold after finishing the plan (e.g., patient has an adequate glucose level).
4. *Overall action pattern*: the overall pattern of provider actions that should hold after finishing the plan (e.g., patient has visited dietician regularly for at least three months).

Conditions

Conditions are also temporal patterns and are used to change the state of a plan. In Asbru, similar to the PROforma approach, plans are in a certain state during execution time (e.g., '*activated*', '*suspended*', '*aborted*' and '*completed*'). Conditions need to hold at particular plan steps to induce a particular state transition of the plan instance. Asbru defined a number of condition categories such as '*filter-preconditions*' and '*setup-preconditions*' that need to hold if a plan is considered applicable, '*suspend-conditions*' that determine when an active plan must be (temporarily) suspended, '*abort-conditions*' that determine when an active or suspended plan has to be aborted and '*completed-conditions*' that determine when a plan is (successfully or not) completed.

Effects

Effects describe the relationship between plan arguments and measurable effects by means of mathematical functions (e.g., the insulin dose is inversely related in some manner to the level of blood glucose). Effects may include probabilities that specify the probability of the effect's occurrence.

Plan body

The plan body is a set of actions or plans that have to be performed whenever the preconditions hold. A plan is composed of other plans, which are performed according to the plan's type. Asbru defines three plan types: '*sequential*', '*concurrent*' and '*cyclical*', the aspects of which are described by means of the *plan* '*subtype*' attribute. Examples of possible subtypes are '*DO-ALL-TOGETHER*' that indicates that all plans in the plan body must be completed concurrently, '*DO-SOME-TOGETHER*' that indicates that all plans are executed in parallel and that some plans must be completed (a '*continuation-condition*' specifies which plans have to be completed), '*DO-SOME-ANY-ORDER*' that indicates that all plans are executed sequentially whereby the order of execution is determined by the '*continuation-conditions*', and '*DO-EVERY*' that indicates a cyclical plan. In the last case, optional temporal and continuation arguments are specified (e.g., whenever a plan is started, ended and repeated). Each plan is decomposed into subplans until a non-decomposable plan (called an action) is encountered.

6.2.2 Temporal patterns and time annotations

Important in Asbru are time annotations: specifying temporal aspects of a plan. A time annotation specifies four points in time relative to a reference point, which can be a specific or abstract point in time, or a plan's state transition. In this manner, Asbru allows for a representation of uncertainty in starting time, ending time and duration. These four points are: the earliest

starting shift (ESS), latest starting shift (LSS), earliest finishing shift (EFS) and latest finishing shift (LFS). Two durations can also be defined: The minimum duration (MinDu) and maximum duration (MaxDu). Together, these data specify the temporal constraints within which an action must take place, or a condition must be fulfilled in order to trigger. Figure 15 shows a schematic view of the time annotation, including an example as used in a guideline that addresses the management of diabetes [16]. The Asbru temporal representation also supports the concept of temporal abstractions, in which guideline authors are able to specify expressions such as ‘has the patient suffered from a *second* episode of anemia of *at least moderate severity*’.

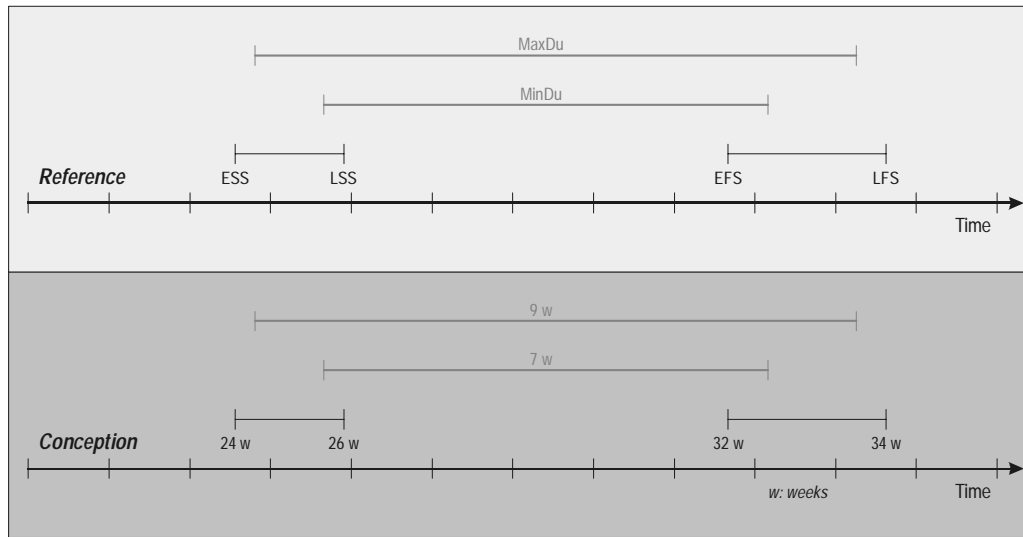


Figure 15: Asbru time annotation. The upper part of the figure presents the generic annotation. The lower part shows a particular example representing the time annotation, which means ‘starts 24 to 26 weeks after conception, ends 32 to 34 weeks after conception, and lasts 7 to 9 weeks’

6.2.3 Guideline representation

In Asbru, a guideline is represented by means of a plan, which consists in turn of a collection of other subplans. Plans are executed sequentially or in parallel. As mentioned earlier, plans that have been started can be suspended, aborted or completed (based on the plan’s conditions). When a plan is completed, the next plan in the sequence (if any) is executed (only one plan at a time can be activated). Figure 16 shows the representation of a guideline for the treatment of Infants’ Respiratory Distress Syndrome (IRDS).

This guideline consists of 4 plans that are executed sequentially. The most important plan (‘one-of-controlled-ventilation’) consists internally of three subplans (‘controlled-ventilation’, ‘permissive-hypercapnia’ and ‘crisis-

management'), which are also executed sequentially, although in this case the order of the sequence depends on the outcome of the '*one of controlled ventilation*' plan's '*continuation-conditions*' (which specify the severity of the I-RDS disease).

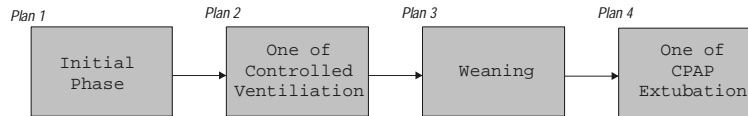


Figure 16: Representation of a guideline in terms of plans

6.2.4 Language

The formal syntax of the Asbru language is defined in Backus-Naur Form (BNF) [59]. The guidelines are encoded in a LISP-like language, as shown in figure 17.

The first paragraph shows the main I-RDS guideline, which contains four subplans that should be executed sequentially (see also figure 16). As mentioned earlier, the '*one-of-controlled-ventilation*' plan consists of three subplans that are sequentially executed in some order before continuing (shown partly in the second paragraph). The third paragraph shows one of these subplans ('*controlled-ventilation*') in more detail. The aim of this plan is to maintain a normal level of the blood-gas values and the lowest level of mechanical ventilation (as defined in the context of controlled ventilation therapy). The plan is activated when the Peak Inspiratory Pressure (PIP) is smaller than or equal to 30 and the transcutaneously assessed blood-gas values are available for at least one minute after activating the last plan instance initial-phase. The plan must be aborted when the PIP is greater than 30 or the increase of the blood-gas values is too steep for at least 30 seconds. Every 10 seconds, the abort conditions are evaluated. The plan is completed successfully when the FiO_2 is smaller than or equal to 50%, the PIP is smaller than or equal to 23, the breathing frequency is smaller than or equal to 60, the patient is not dyspnoeic, and the blood gas values are normal or above the normal range for at least three hours. The complete conditions are evaluated every 10 minutes. The body of the plan again consists of two subplans ('*one-of-increase-decrease-ventilation*' and '*observing*') that are executed sequentially.

Besides the BNF-based and LISP-like syntaxes, an XML-based version of the Asbru syntax was also recently defined and published [61].

```

(PLAN I-RDS-therapy ...
...
(DO-ALL-SEQUENTIALLY
  (initial-phase)
  (one-of-controlled-ventilation)
  (weaning)
  (One-of-cpap-extubation)))

(PLAN one-of-controlled-ventilation ...
...
(DO-SOME-ANY-ORDER
  (controlled-ventilation)
  (permissive-hypercapnia)
  (crisis-management)
  CONTINUATION-CONDITION controlled-ventilation))

(PLAN controlled-ventilation
  (PREFERENCES (SELECT-METHOD BEST-FIT))
  (INTENTION:INTERMEDIATE-STATE (MAINTAIN STATE(BG) NORMAL controlled-ventilation *))
  (INTENTION:INTERMEDIATE-ACTION (MAINTAIN STATE(RESPIRATOR-SETTING) LOW controlled-ventilation *))
  (SETUP-PRECONDITIONS (PIP (<= 30) I-RDS *now*)
    (BG available I-RDS [[_, _], [_, _], [1 MIN, _] (ACTIVATED initial-phase-1#)]))
  (ACTIVATED-CONDITIONS AUTOMATIC)
  (ABORT-CONDITIONS ACTIVATED
    (OR (PIP (> 30) controlled-ventilation [[_, _], [_, _], [30 SEC, _], *self*])
      (RATE(BG) TOO-STEEP controlled-ventilation [[_, _], [_, _], [30 SEC, _], *self*])))
  (SAMPLING-FREQUENCY 10 SEC))
  (COMPLETE-CONDITIONS
    (FiO2 (<= 50) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*])
    (PIP (<= 23) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*])
    (f (<= 60) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*])
    (state(patient) (NOT DYSPNEIC) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*])
    (STATE(BG) (OR NORMAL ABOVE-NORMAL) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*])
    (SAMPLING-FREQUENCY 10 MIN))
  (DO-ALL-SEQUENTIALLY
    (one-of-increase-decrease-ventilation)
    (Observing)))

```

Figure 17: A portion of the I-RDS guideline, encoded in Asbru [16]

6.3 Guideline acquisition, verification and testing

In contrast to approaches such as GLIF and PROforma, the developers of Asbru have chosen not to visualize guidelines by means of a flowchart, mainly as they feel that visualizing time and intentions through flowcharts is a very difficult task. Instead, a tool named AsbruView was created that uses the

concept of metaphor graphics to visualize guidelines [62]. In AsbruView, plans are visualized as running tracks and the various types of conditions are visualized by means of traffic signs and other controls. This visualization is known as the topological view. Figure 18 shows part of the I-RDS guideline in the AsbruView topological view.

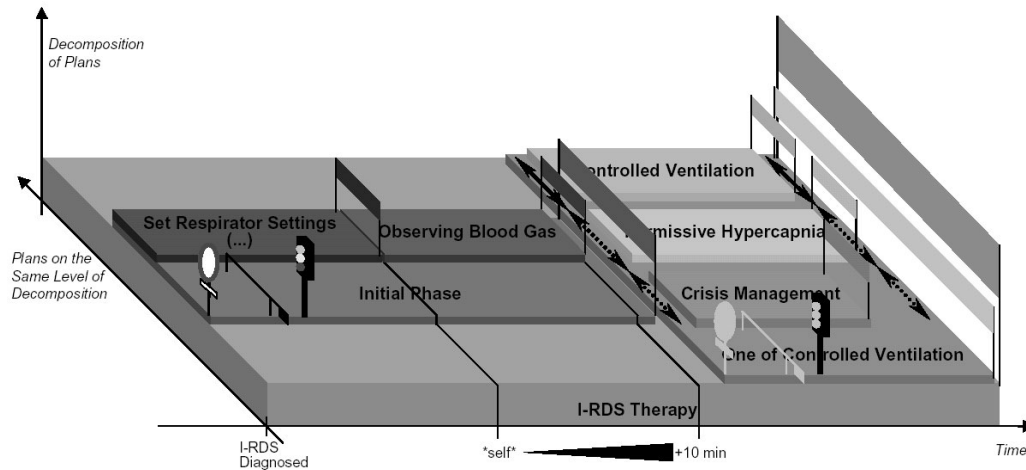


Figure 18: The I-RDS guideline, visualized in AsbruView. Of the four I-RDS subplans, only the 'initial-phase' and the 'one-of-controlled-ventilation' plans are shown here [62]

The length of the track represents time, the depth represents (sub)plans that are on the same level of decomposition and the height represents the various levels of decomposition. In this case, the main guideline ('I-RDS-Therapy') contains two sequential subplans: 'initial-phase' and 'one-of-controlled-ventilation' (The 'weaning' and 'one-of-CPAP-extubation' plans in figure 16 are omitted here). The 'initial-phase' plan also contains two sequential subplans ('set-respirator-settings' and 'observing-blood-gas') and the 'one-of-controlled-ventilation' subplan contains the four earlier-mentioned subplans (see also figure 17). The time dimension is only symbolic: a plan's size does not reflect its actual duration.

Furthermore, AsbruView uses other metaphors to symbolize conditions. For example, the 'no entrance with exceptions' traffic sign symbolizes the filter preconditions and the turnpike (barrier) sign symbolizes setup preconditions. Furthermore, each traffic light includes three kinds of conditions. The red light symbolizes the abort-condition, the yellow light the suspend-condition and the green light the reactivate-condition. The finishing flag, finally, symbolizes the complete condition, which specifies when the plan has reached its goal and can be considered successful.

In AsbruView, plans can be depicted by means of two different views: the Topological view and the Temporal view [63]. Most common is the topological view (shown in figure 18) that displays relationships between plans, without an explicit timeline. In contrast, the recently developed Temporal view focuses on the temporal dimensions of plans and conditions by showing plans as explicit guidelines. The Temporal view uses the time annotation that specifies four points in time relative to a reference point, also shown in figure 15. Figure 19 shows a portion of the I-RDS guideline, visualized through the Temporal view.

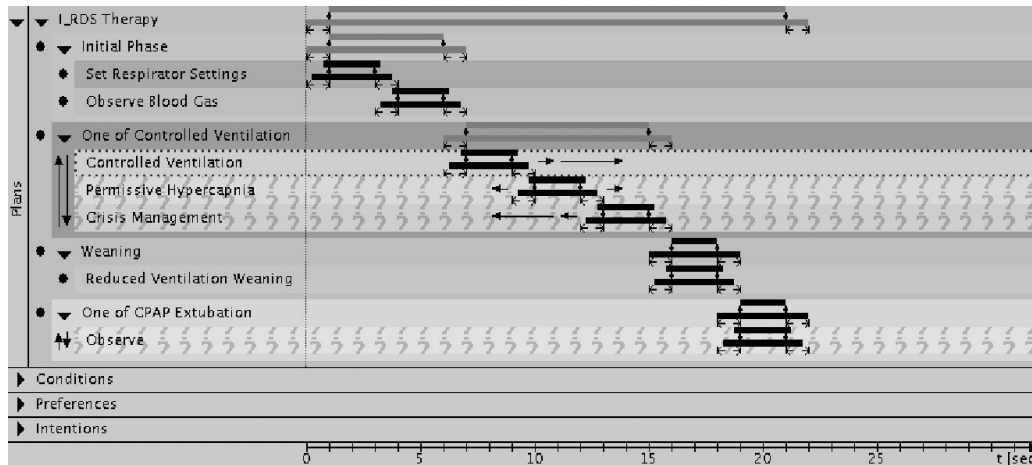


Figure 19: The I-RDS guideline, visualized in AsbruView through the Temporal view [63]

As the Asbru language is formally defined, entered guidelines can be verified to detect various types of logical and procedural errors [64]. These tools are currently under development.

6.4 Guideline execution

Tools and software that facilitate the execution of guidelines, written in Asbru are currently under development.

7 EON

7.1 Introduction

7.1.1 Overview

EON, also developed at Stanford University, is a component-based architecture used to build decision support systems that reason about guideline-directed care [17]. The EON architecture consists of several components that facilitate the acquisition and execution of clinical guidelines. Similar to GLIF (as mentioned earlier, EON was one of the approaches from

which GLIF originated), the guideline model of EON, called Dharma [19], is object-oriented and consists of classes that describe guideline entities as a sequence of structured temporal steps. The Dharma model is non-monolithic, meaning that the guideline model can be extended with additional classes that capture new guideline behavior. Besides the Dharma guideline model, the EON architecture also contains a number of run-time components, used to construct execution-time systems.

7.2 Guideline model and representation

7.2.1 The Dharma guideline model

In contrast with for example GLIF and *PROforma* that model guidelines in terms of a fixed number of primitives (e.g., decisions, actions), the researchers of EON argue that -for the purpose of providing decision support- a fixed number of primitives is not sufficient to model all sorts of guidelines, as guidelines may differ considerably in variability and complexity. Instead, they propose a non-monolithic (non-closed) guideline model, which consists of a standard set of primitives that can be extended with task-specific submodels, resulting in additional classes of primitives that are matched to the knowledge requirements of different guidelines.

In the Dharma model, guidelines manage patient behavior, consisting of decisions and actions that may lead to dependent changes in patient states over time (figure 20).

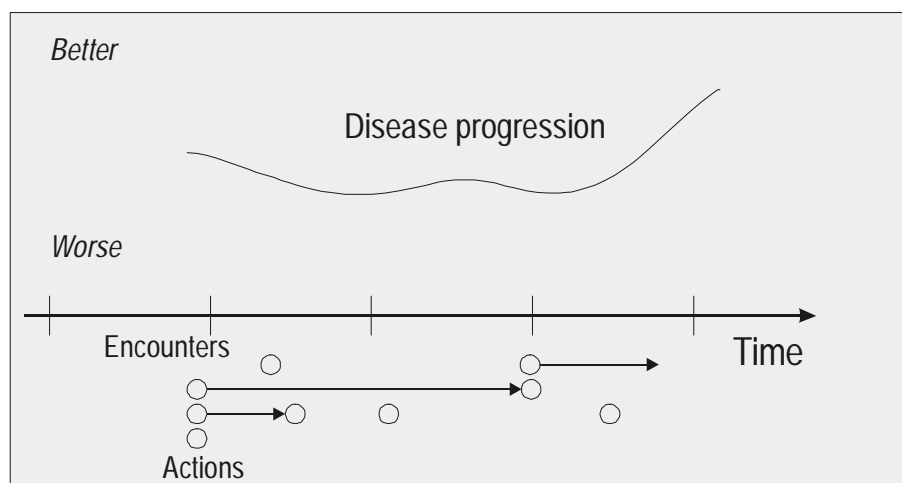


Figure 20: Conceptual model of the patient management process

In this conceptual model of multi-encounter patient management, decisions are made during encounters between healthcare providers and patients. Actions such as writing a prescription or requesting a laboratory test, are

carried out during encounters and may (in)directly lead to a change in the patient state (e.g., the progression of a disease). Some actions can start activities that extend over time. In order to define guidelines according to this conceptual model, they are represented in terms of a number of key characteristics, represented by primitives. Examples of EON primitives are scenarios, decisions, actions and goals. These primitives form the core guideline ontology. As the model is object-oriented, these characteristics are represented by means of a set of classes, attributes for those classes and data types for the attribute values.

Scenarios

A scenario is a (partial) characterization of the state of a patient (e.g., the patient is currently being prescribed a low-dosed steroid). In a scenario, eligibility conditions specify the necessary conditions for a patient to be in this scenario. Scenarios allow a clinician to synchronize the management of a patient with the corresponding parts of (a portion of) a guideline and are commonly used as entry points in a guideline. In the Dharma ontology, a scenario is always followed by a decision or action step. Each scenario in an actual guideline is an instance of the *scenario* class, which contains several attributes such as an attribute that specifies the eligibility criteria and an attribute that specifies the step that follows the current scenario (similar to GLIF). Scenarios allow a clinician to synchronize the management of a patient to situations handled by a guideline.

Scenarios can be applied in several ways. First of all, they can be used as 'entry points' in a guideline. However, they can also serve to model exceptions, which represent exceptional situations that rarely occur. As expressing everything in a guideline can be impractical, a guideline author may want to partition the guideline into normal situations that cover usual cases and exceptions. The Dharma ontology defines two classes of exceptions: 1) exceptions that are repairable (i.e., those that lead a patient back to a scenario covered by the guideline), and 2) exceptions that are not repairable, so that patient has to be managed outside this guideline.

Decisions

A decision represents a choice from a set of competing alternatives. In the Dharma core ontology, two basic types of decisions are defined (by means of two subclasses): decisions that model '*if-then-else*' choices and decisions that require making a heuristic choice from a set of pre-enumerated alternatives. Regarding the latter, making a choice among the alternatives is aided by preferences as determined by rule-in and rule-out conditions that support or oppose each alternative (similar to the concept of schemas in *PROforma*). If a

rule-out condition evaluates to true, then the corresponding alternative is rejected. If the rule-out condition does not apply and a rule-in condition evaluates to true, the corresponding alternative is marked as preferred. If neither evaluates to true, then the preference for the choice can be determined by a default preference associated with each alternative. The Dharma model supports different ways of expressing decision criteria, which are explained in more detail later.

Actions

Actions are instantaneous acts that lead to changes in the state of the world such as collecting patient data, displaying a message to the user or starting a drug regimen. Actions are used heavily throughout guidelines modeled in EON.

Whereas actions refer to instantaneous acts, activities model processes that take place over time. Activities have states that can change from time to time. These changes are usually the result of actions specified in a guideline, as actions are able to start a new activity, stop an ongoing activity or change the attribute values of an ongoing activity. Activities have states that are characterized by a set of attributes such as a dose level (e.g., *'low'*, *'high'*, *'medium'*) and a frequency (e.g., *'twice a day'*) of a drug regime.

Finally, the model also includes actions that refer to a set of other actions or a subguideline. Similar to GLIF, examples of such actions are actions that model branching and synchronization constructs in order to execute parallel tasks.

Goals

Every step can be associated with a goal. The notion of goals is comparable with the notions of intentions in Asbru, although less sophisticated. In the Dharma ontology, goals are represented as boolean criteria (e.g., *'reduce the arterial blood pressure to less than 130/85 within three weeks'*). The format of these criteria is explained later.

7.2.2 The Patient data model

The patient data model defines classes and attributes in order to represent patient data. For example, the patient data model defines a *Patient* class, whose instances hold demographic information about specific patients, a *Qualitative_Entry* class that describes qualitative observations about patients, a *Numeric_Entry* class that stores results of quantitative measurements, an *Adverse_Event* class that models adverse reactions to specific substances, a *Condition* class that represent medical conditions that persist over time, and

two intervention classes, *Medication* and *Procedure*, that model drugs and other medical procedures that have been recommended, or used. The patient data model defines characteristics regarding demographic and clinical conditions of specific patients. It does not aim at modeling the entire patient (e.g., replicate the structure of an EPR), but models only those distinctions that are relevant for the purpose of defining guidelines and protocols. The ideas behind the EON patient data model are very similar to those of the GLIF Reference Information Model (RIM).

7.2.3 The Medical-specialty model

The medical-specialty model consists of a medical domain ontology that models the structure of domain concepts (e.g., drugs and treatments) in terms of organized classes, relations and attributes. The medical-specialty model represents different sorts of domain-specific information. For example, in the context of hypertension management, hyponatremia as a contraindication for the use of thiazide may be defined as a serum sodium measurement that is less than 135 mg/dl. In the medical-specialty model, the concept of *Hyponatremia* is then defined in terms of the range of values, related to the concept of *Serum_sodium measurement*. Concepts from the medical-specialty model can be linked to concepts from the patient data model. In the above-mentioned hyponatremia example, the hypertension management guideline may contain an instance of the *Numeric_Entry* class (defined in the patient data model). This class defines a *Domain_term* attribute, which refers in this case to the concept *Serum_Sodium* from the medical-specialty model. The medical-specialty model is very similar to the GLIF medical knowledge layer.

7.2.4 Modeling tools

Protégé is used as a modeling tool to define the classes and attributes that form the core guideline model, the patient data model and the medical-specialty model. Also, additional classes that are derived from the core guideline model that introduce additional functionality are defined and entered by means of Protégé.

7.2.5 Guideline representation

Similar to GLIF, guidelines are represented in EON by temporally sequenced graphs (flowcharts) of instantiated classes. For example, figure 21 shows part of an influenza-vaccination guideline.

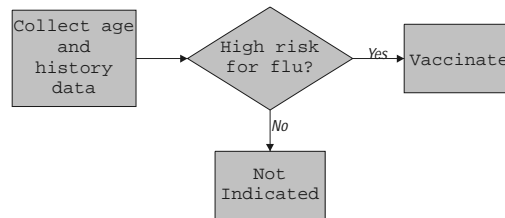


Figure 21: Graphical representation of an influenza-vaccination guideline in EON

This guideline states that patients with a high risk for flu must be vaccinated. The risk is based on the patient's age and history and is defined as '*age > 65 or presence of chronic heart or pulmonary problems*'. The influenza-vaccination guideline is modeled as an '*if-then-else*' condition that consists of three actions and one decision. The first action involves the collection of data, necessary to determine whether the patient has a high risk for flu, modeled by a decision. Depending on the outcome of that decision, a corresponding action can be carried out (e.g., warn that this patient must be vaccinated). The '*collect age and history*' action step is an instance of the *Consultation_Action_Step* class, which is not in the core Dharma guideline model, but is a derived class that defines an action step that acquires consultation-based data (e.g., by means of asking questions to a physician). Regarding temporal aspects, EON has adopted a subset of the Asbru temporal language to represent temporal information.

7.2.6 Language

The EON model itself does not define a formal language regarding the guideline model or the guidelines but uses the internal frame-based Resource Description Format (RDF) of Protégé [65] to describe the models as well as the guidelines. Although the focus of the EON project is not on defining a formal syntax for representing guidelines in general, it does particularly address the subject of how to describe criteria that are used in decisions. EON defines three different criterion languages.

First, common but relatively simple criteria can be expressed as boolean criteria in terms of a set of object templates. Criteria encoded in this object-based language evaluate to true, false, or unknown. An example of such a criterion is '*diabetes mellitus is present and the most recent serum creatinine is less than normal*'.

According to the researchers of the EON project, such a criterion language is not expressive enough to capture more complex criteria such as '*is an authorized medication present that is contraindicated by some medical condition*'. To represent such criteria, the Protégé Axiom Language (PAL) is

used, which is embedded into the Protégé development environment. The PAL constraint language is a subset of the first-order predicate logic Knowledge Interchange Format (KIF) syntax [66].

Finally, it is possible to write complex temporal criteria such as *'presence of an episode of uncontrolled blood pressure that overlaps with lisinopril medication and that started within two weeks after the initiation of lisinopril'*. These are written as temporal queries, which during guideline execution are translated to database queries [67].

Examples of criteria that were written in PAL or as temporal queries are shown in the next two sections.

7.3 Guideline acquisition, verification and testing

Besides defining the various EON models, Protégé is also used as a Knowledge Acquisition tool where guideline authors are able to enter and view guidelines. Protégé takes as input the Dharma guideline model, a Patient Data Model and a Medical-Specialty Model (explained in more detail in the next section) to create a Knowledge Acquisition Tool. Figure 22 shows portion of a guideline for the treatment of breast-cancer, visualized in Protégé (see also figure 6).

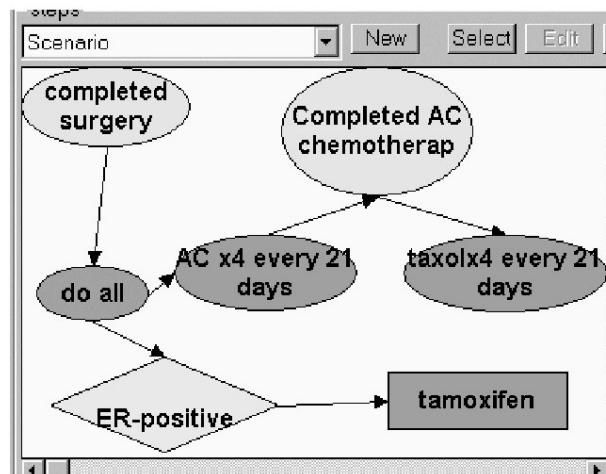


Figure 22: Part of a guideline that addresses the management of breast-cancer

In this figure, the oval entitled *'do all'* represents a specific action step that models branching. Furthermore, statements such as *'taxol x4 every 21 days'* are repetitions of actions involving the drug taxol as a prescribable item. Besides actions, the part of the breast-cancer management protocol, shown in figure 22 also contains 2 scenarios (*'completed surgery'* and *'completed AC chemotherapy'*) and a decision (*'ER-positive'*).

As mentioned in the previous section, PAL is used to formally describe complex criteria, used in decisions. Figure 23 shows examples of PAL criteria that check the existence of drug contraindications.

```
(defrange ?current_med :FRAME Medication)
...
(exists ?current_med
  (exists ?med_class
    (and (subclass-of
          (drug_name ?current_med) ?med_class)
      (exists ?contraindication
        (and (Absolute_Contraindications
              ?med_class ?contraindication)
          (exists ?problem
            (subclass-of
              (domain_term ?problem)
              ?contraindication)))))))
```

Figure 23: Simplified PAL criteria to check the existence of contraindicated medications

These criteria state that, for each current medication, its contraindications from the medical-specialty knowledge base have to be determined, and to see if there is any patient-data instance that suggests the presence of one of these contraindications. PAL makes full use of Protégé's frame-based knowledge model. For example, variables can range over instances of Protégé classes (e.g. the variable '*?current_med*' ranges over instances of the *Medication* class) and attributes of classes (e.g. *Absolute_Contraindications*). Protégé contains a structured editor that facilitates guideline developers in writing these complex logical criteria. However, such criteria are usually not formulated and entered by domain experts that are not trained in logic.

7.4 Guideline execution

To facilitate the development of guideline execution engines, EON defines an execution architecture that contains components for guideline execution and interfacing third-party information systems. Figure 24 shows an overview of the execution architecture [68].

The heart of the execution architecture is formed by the Padda Guideline Execution Server (or Padda Server), which applies a clinical guideline to patient data queried from an information system's database and generates advisories [69]. Within the Padda Server, a knowledge-base handler manages access to the guideline knowledge base and the patient data model via the application-programming interface provided by Protégé. For a specific guideline and patient, the Padda Server must determine if the guideline is applicable to the patient, and subsequently, implement a model of interaction with the outside world (e.g., information systems or clinicians). The Padda Server uses patient data to suggest that a patient is in a specific scenario, and

that, as a result, tasks such as laboratory tests should be performed. The server may also suggest that certain alternatives at a decision point are preferred. However, users are always allowed to override the system's conclusions. For the Padda server to communicate with other information systems (e.g., EPRs), an interface specification has been defined. This specification, written in Common Object Request Broker Architecture Interface Definition Language (CORBA IDL) consists of methods with which client and server interact with each other as well as a description of the data structures that are passed between the server and clients.

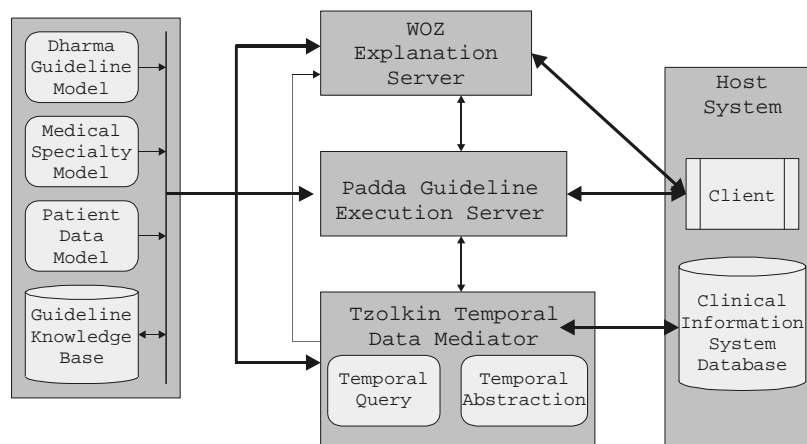


Figure 24: An overview of the EON execution architecture

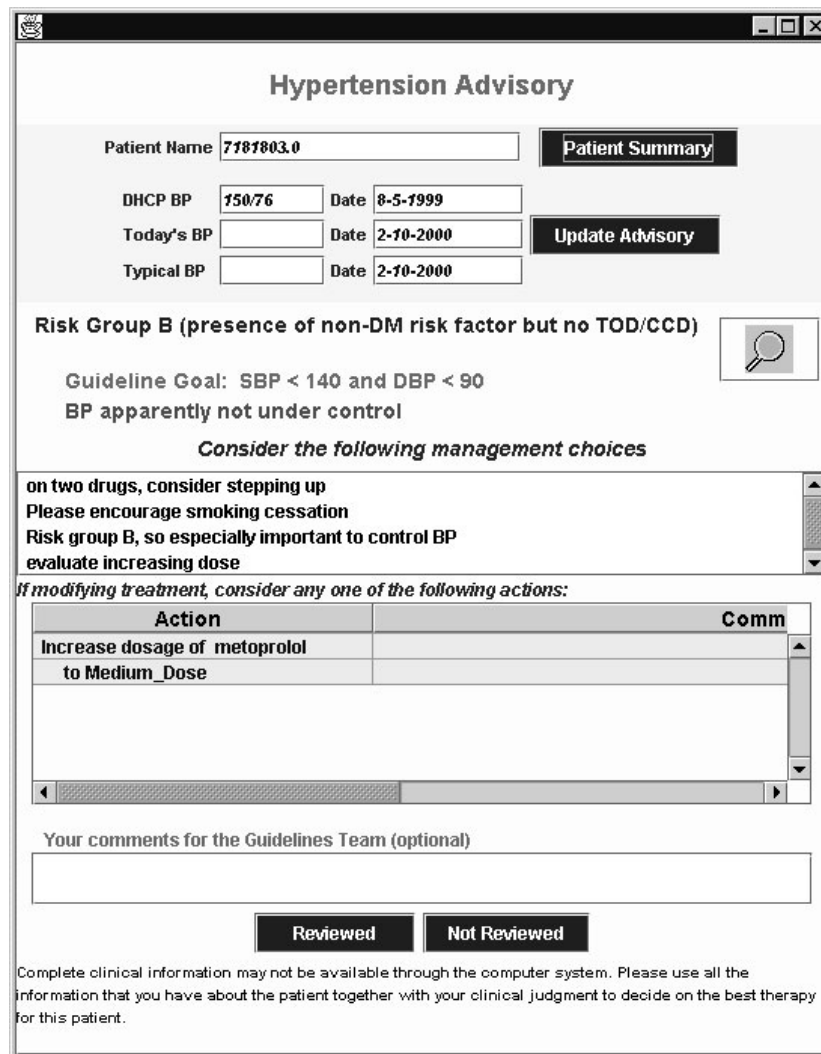
To evaluate specific patient situations, available patient data must be mapped to the terms and relations that are used in the guideline. The linking of concepts in the patient data model to corresponding concepts in an information system (e.g., an EPR) is done through the Tzolkin data mediator [67]. This component performs two functions. First, it maps concepts from a particular patient data model to corresponding concepts from the data model of the host system. Second, it maps terminology in the medical-specialty model (e.g., as names of laboratory test results) to the terminology used in the host information system. As mentioned earlier, criteria often contain complex temporal expressions. Making abstractions from time-stamped patient data (e.g., an episode of uncontrolled blood pressure) and comparing temporal sequences of occurrences (the episode of uncontrolled blood pressure overlaps the use of lisinopril and it started within two weeks after initiation of lisinopril) is often necessary. For formulating these types of criteria, the Tzolkin temporal data mediator contains a temporal query language that is able to define such temporal expressions. During runtime, the Tzolkin data mediator translates these temporal queries to 'standard' database queries (e.g., SQL). Figure 25 shows an example of a Tzolkin temporal query.

```

TEMPORAL SELECT domain_name
VALID INTERSECT(Condition, Medication)
FROM Condition, Medication
WHERE domain_name = "UNCONTROLLED_BP" AND
      drug_name = "lisinopril"
WHEN start(Condition) AFTER start(Medication) AND
      start(Condition) BEFORE
      (start(Medication) + weeks(2))

```

Figure 25: An example of a temporal query that checks for the existence of an episode of uncontrolled blood pressure that overlaps with administration of lisinopril but occurring within two weeks of initiating lisinopril




Hypertension Advisory

Patient Name: **Patient Summary**

DHCP BP: Date:

Today's BP: Date: **Update Advisory**

Typical BP: Date:

Risk Group B (presence of non-DM risk factor but no TOD/CCD) 

Guideline Goal: SBP < 140 and DBP < 90
BP apparently not under control

Consider the following management choices

on two drugs, consider stepping up
Please encourage smoking cessation
Risk group B, so especially important to control BP
evaluate increasing dose

If modifying treatment, consider any one of the following actions:

Action	Comm
Increase dosage of metoprolol to Medium_Dose	

Your comments for the Guidelines Team (optional)

Reviewed **Not Reviewed**

Complete clinical information may not be available through the computer system. Please use all the information that you have about the patient together with your clinical judgment to decide on the best therapy for this patient.

Figure 26: Advice, given by the WOZ component of the ATHENA hypertension advisory system

Finally, the WOZ (Wizard of OZ) component provides explanation services [70]. An example of an advisory, provided by the WOZ component is shown in figure 26. This advisory was issued by the ATHENA system, a decision

support systems that manages the treatment of hypertension, which was implemented using the EON execution engine [71].

8 Discussion

8.1 Comparison

8.1.1 Overview

Each approach focuses on different aspects of guideline representation, development and implementation. The Arden Syntax and GLIF approaches focus on guideline standardization, *PROforma* on execution aspects, Asbru on the representation and visualization of complex temporal plans, and EON on the development of an architecture that supports the development and implementation of guidelines. These different focus points have their implications regarding the representation, acquisition, verification and implementation of guidelines as shown in the previous sections.

8.1.2 Guideline Modeling and representation

Primitives

The representation model of the Arden Syntax differs from other approaches, as it is the only approach that models each guideline as an independent modular rule. As a result, the Arden Syntax is most suitable for representing simple guidelines such as alerts in reminder systems.

The GLIF, *PROforma*, Asbru and EON approach all model guidelines in a similar way, in terms of primitives (steps, tasks or plans) that describe the control structure of a guideline. GLIF and EON have very similar models, as they were partly developed by the same groups and researchers. The main difference is that the GLIF model, just as *PROforma* and Asbru, contains a fixed number of primitives, while the EON set of primitives is extendible. The basic primitives however such as primitives that represent decisions, actions and patient states (entry points) are present in both GLIF and EON. MLMs contain similar constructs such as decisions (*logic* slot), actions (*action* slot) and patient states (*evoke* slot). However, these can only be used to model modular rule-based guidelines that perform simple actions (e.g., provide alerts). Primitives that describe decisions and actions are also present in *PROforma*. Although *PROforma* does not provide explicit support for defining patient states, it is possible to model these through constructs like triggers and pre- and postconditions [72]. The *PROforma* enquiry task is viewed as an action in the GLIF and EON models. In Asbru, the basic primitive is an action: every (sub)plan eventually consists of actions. In contrast to other approaches where the functionality is described in terms of primitives, Asbru uses

knowledge roles such as preferences, intentions, conditions and effects for this purpose.

All approaches support some form of temporal reasoning, of which the Asbru approach contains the most sophisticated structures. EON and GLIF both adopt a subset of the Asbru temporal language. In order to be compatible with the Arden Syntax, the GLIF Expression Language (GEL) also defines a number of operators that are defined in the Arden Syntax such as '*before*', '*after*' and '*ago*'. Similar constructs are also available in the *PROforma* expression language. The Arden Syntax and GLIF support a limited form of uncertainty in terms of a three-valued logic ('*true*', '*false*' and '*unknown*'). *PROforma* is the only approach that contains expressive constructs for describing uncertainty aspects of a guideline. In contrast with the issue of representing temporal aspects, the representation of uncertainty in guidelines is not regarded as a critical issue in general.

Complexity

All models except for the Arden Syntax provide explicit support for nesting of guidelines in order to model complex guidelines in terms of subguidelines (GLIF and EON) or subplans (*PROforma* and Asbru). For this purpose, GLIF, EON and *PROforma* contain an *Action* primitive that may contain a reference to a subguideline or subplan. In Asbru, each plan body contains a number of subplans until a non-decomposable plan (also called *Action*) is encountered. Although the Arden Syntax is able to call other rules in the *Action* slot, there is no general way of controlling these invocations.

EON, *PROforma* and Asbru also support the use of goals and intentions to formally specify a guideline on a higher level of abstraction. Of these techniques, the Asbru intension model is the most sophisticated.

GLIF defines different layers of abstraction, which allows guideline authors only to view the general control structure (flowchart) of a guideline before specifying all the necessary details. EON uses a non-monolithic approach: the Dharma guideline model is based on a core model, which can be extended with submodels depending on the complexity of the guideline (e.g., '*if-then-else*' rules versus complex treatment guidelines).

The representations of Asbru and EON also allow for the abstraction of temporal data to facilitate the specification of complex temporal expressions.

Except Asbru, all approaches support the concept of referenced subguidelines. In Asbru, subplans are 'embedded' in a plan, meaning that this

subplan is not known outside the embedding plan. As a result, a certain subplan is not sharable with other plans outside the embedding plan.

GLIF also supports the representation of common guideline structures through Macros, which facilitates the reuse of guidelines that are used often (e.g., ‘if-then’ rules such as MLMs).

Knowledge types

Besides the knowledge that defines the control structure (e.g., rules, primitives, plans, sequences), every guideline also contains domain-specific knowledge such as medical knowledge (e.g., terminology) and knowledge concerning the patient (e.g., the patient’s symptoms or history).

The Arden Syntax contains no support of separating these types of knowledge, as each reference to a domain-specific item is stored as a label in the *data* slot of a MLM. As a result, a MLM does not ‘know’ for example that amoxicillin is an antibiotic. Also PROforma and Asbru contain no explicit support for modeling domain-specific knowledge or for using standard terminology systems. GLIF addresses this problem by modeling domain-specific knowledge by means of defining a Medical Ontology that contains three different layers: the core GLIF layer, the RIM layer and the Medical Knowledge layer. EON takes a very similar approach by defining the Dharma guideline model, the Patient Data Model and the Medical-Specialty Model. Currently, both the layers in GLIF as well as the models in EON are still partly under development.

Besides invoking subguidelines, a guideline may consist of various types of actions such as medically oriented actions (e.g., recommending a particular course of treatment) and programming-oriented actions (e.g., supplying a message to a care provider). In the Arden Syntax, actions (stored in the *action* slot) are usually programming-oriented as they are used to generate reminders or alerts. This is also the case in the PROforma approach, as a PROforma action is a programming-related task that is carried out by the execution engine through an Application Programming Interface (API). GLIF and EON both support these two types of actions. Finally, Asbru does not support programming-related actions.

Didactic and maintenance

Didactic and maintenance information concerns information about authors, versioning, purposes and detailed explanations. The Arden Syntax, GLIF and EON approaches are all able to hold various kinds of information such as the guideline’s author, version, institution, keywords, validation (e.g., ‘research’,

'testing', 'production') and explanation. In *PROforma* and Asbru, it is not possible to store didactic and maintenance-related information (besides a name and explanation).

Language

All approaches except EON have defined a language that entirely describes the representation through a formal syntax: the Arden syntax, *PROforma* and Asbru use BNF (the latest version of Asbru is also in XML-format) and GLIF uses UML. EON relies on the internal syntax of Protégé. For each approach, the syntax captures all aspects that are defined in the corresponding representations.

Regarding the guidelines itself, the Arden Syntax describes guidelines in terms of a semi-structured ASCII format (see figure 1), GLIF describes guidelines in an XML format (see figure 4), Asbru in a LISP-like syntax (see figure 17) and *PROforma* in the R^2L language. EON uses a description that is very similar to that of GLIF, with the main exception that GLIF describes expressions in the Guideline Expression Language (GEL) while EON describes expressions by means of the three different criterion languages.

PROforma is the only approach, which makes a distinction between a declarative language (e.g., R^2L), used during the guideline acquisition phase and a procedural language (e.g., L_{R2L}) that is processed by a general interpreter (e.g., PROLOG) in an execution engine. All other approaches require a custom-developed execution engine, in which the different procedural aspects of the guideline are encoded programmatically (e.g., a number of Java or C procedures that each executes a certain primitive).

In order to facilitate the translation from a declarative language to a procedural language, the *PROforma* representation language contains constructs that are filled in during guideline acquisition but are execution-related. For example, *PROforma* defines an execution state that denotes the state of a guideline during execution (e.g., 'in progress', 'aborted', 'terminated', 'performed'). This is in contrast with EON and GLIF that define patient states which are used during execution to determine the applicability of a guideline (as mentioned earlier, *PROforma* is also able to model patient states implicitly through constructs like triggers and pre- and postconditions). Similar to *PROforma*, Asbru also contains the concept of guideline execution states.

8.1.3 Guideline acquisition

The developers of the Arden Syntax have not developed tools that facilitate the process of guideline acquisition, although various acquisition tools were created by third parties. GLIF and EON use Protégé as the main knowledge acquisition tool (see figures 6 and 22), in which guidelines are entered as flowcharts. As mentioned earlier, every primitive is shown as a generic form in Protégé (see also figure 6). The advantage is that the user interface is created automatically by Protégé. The disadvantage is that there is limited guidance as each instance is shown as a separate form and guideline authors can get 'lost' when there are too many forms open.

The GLIF Expression Language is similar to the Arden Syntax. During knowledge acquisition in Protégé, GLIF expressions are still entered as strings (e.g., `'test_name = "Serum_Potassium"'`), which has to be parsed in order to extract the various kinds of information such as the different operators and used domain terms. Therefore, it is possible for guideline authors to type in erroneous criteria if there is no syntax checker available. As EON contains three different criterion languages, each guideline author has to decide which of these three languages (s)he will use. Although the EON architecture contains tools that partly facilitate the structured entry of these languages, only guideline authors that are skilled in writing logic and database queries will be able to write complex criteria using PAL logic or temporal queries.

PROforma uses a language (R^2L) that has a predicate logic language (L_{R^2L}) underlying, which has the advantage that guidelines are defined in a formal manner. However, the PROforma tasks are very basic and 'low-level', so that it may be difficult for guideline authors to enter guidelines, as they often do not view guidelines in terms of schemas, pre-and post conditions and predicate logic, making PROforma more like a guideline programming language than an abstract representation. PROforma contains a very elaborate tool for guideline acquisition. The acquisition tool facilitates guideline authors using a sophisticated graphical editor, as shown in figure 12. However, guideline authors may interpret the constraint satisfaction graph as a standard flowchart. This is not the case however, as the arrows between task instances can represent different types of constraint. Also, guideline authors are required to specify execution-time information such as guideline execution states, which may differ from an author's viewpoint of a guideline.

Acquiring guidelines by means of using graphical metaphors has become one of the focus points of Asbru. AsbruView uses sophisticated visualization techniques to facilitate the acquisition of complex guidelines. In contrast to

GLIF, EON and PROforma, the Asbru researchers have chosen not to model guidelines through flowcharts but by means of metaphor graphics such as running tracks or traffic lights. It has still to be proven which visualization technique is the most suited.

8.1.4 Guideline verification and testing

PROforma is the only approach that has developed tools, which -based on a sound formal language- verify entered guidelines by detecting a number of possible logical and procedural errors such as incorrect data types, invalid syntax or attribute values, critical missing values or concepts and inconsistent constraints. Although for other approaches, tools for guideline verification and testing are reported to be in development, no results have been published so far.

8.1.5 Guideline execution

EON and PROforma have developed execution engines which are able to process guidelines developed in the corresponding languages. Also, these two approaches have published results on the development and implementation of actual decision support systems. PROforma is the only approach that has developed a commercialized version. Both systems are able to communicate with clinical information systems and users through standard Application Programming Interfaces (APIs) or communication protocols (e.g., CORBA).

A number of third parties have implemented decision support systems that are able to execute Arden Syntax guidelines for use in their local institutions. However, these are often not reusable in other environments.

The development and implementation of execution engines have not been a major focus point of the GLIF developers until now. Recently, they have started the development of the GuideLine Execution Language (GLEE), although the development is still in its very early stages.

As mentioned earlier, no publications are known that address the development of guideline execution engines, which are able to execute Asbru guidelines.

8.2 Requirements

8.2.1 Overview

The descriptions and comparisons in the previous sections show that each approach has a number of strong and weak points. This section formulates

requirements that were distilled from these strong and weak points in the areas of guideline representation, acquisition, verification and execution that can be used in the process of developing new approaches or improving existing ones.

8.2.2 Guideline Representation

Primitives

A guideline representation must contain a set of primitives that is able to represent all facets of simple as well as complex diagnostic and treatment guidelines. These primitives must be understandable on a functional level by guideline authors and on an executable level by computerized decision support systems.

A guideline representation formalism must support at least the two necessary basic building blocks: actions and decisions. In order to be able to specify guideline-oriented actions (e.g., '*prescribe new medication*' or '*diagnose patient with hypertension*') as well as programming-oriented actions (e.g., '*get all drugs from an EPR*' or '*give message to user*') a guideline representation must:

1. Provide a very expressive language that enables the specification of all above-mentioned actions in a limited set of action-related primitives (e.g., the GEL language in GLIF, the R²L language in PROforma or the Asbru expression language)

or

2. Provide the ability to derive new classes from the existing ones that define new functionality (e.g., the non-closed DHARMA model in EON).

Other important primitives in a guideline representation model are primitives that influence guideline flow such as entry/exit points (e.g., patient state primitives) and repetition/loops (e.g., synchronization steps or the Asbru plan type).

Temporal logic is a very important issue in guideline modeling. Guidelines usually refer to complex temporal constructs to describe for example drug prescription schemes. Therefore, a guideline representation model must contain an expressive means of modeling temporal expressions (e.g., Asbru's temporal logic).

The truth-value of a decision can not always be evaluated as ‘*true*’ or ‘*false*’, for example in the case of missing data (e.g., the patient’s medical history is not known). Guideline models must be able to handle such situations (e.g., using the relatively simple three-valued logic in GLIF or the more complex R²L language in PROforma).

Complexity

The guideline representation formalism must be able to represent various kinds of guidelines, that may differ considerably in complexity in a consistent manner such as relatively simple guidelines that model independent modular rules (e.g., MLMs in the Arden Syntax or MLM-macros in GLIF), but also complex guidelines such as clinical trials or treatment plans. In order to represent these various types of guidelines in a consistent manner, the formalism must be able to represent guidelines on multiple levels of abstraction such as nesting, task or guideline decomposition (e.g., subguidelines or subplans in GLIF, EON, PROforma and Asbru), and specifying the guideline’s intention or goal (e.g., Asbru’s intentions).

Knowledge

Computer-interpretable guidelines that are used for active decision support must be integrated with existing clinical information systems such as Electronic Patient Record (EPR) systems. Concepts that are used in a guideline such as patient demographics, results of laboratory tests, indications and drugs must be explicitly defined so that they can be mapped to entries in a clinical information system. To facilitate the (re)use of a guideline among different institutions and systems, the reasoning knowledge (e.g., the used methods or primitives) must be separated from domain-specific knowledge (e.g., used drugs or laboratory tests). Also, the representation should support the use of standard data models and medical terminologies such as HL-7, UMLS (e.g., the three-layered approaches in GLIF and EON) and SNOMED [73].

Furthermore, in order to further facilitate the sharing of guideline-based decision support systems and to increase the acceptance of (national) guidelines in local institutions, actions that are programming-related must be separated from actions that are not. In this manner, institution-specific actions (e.g., sending an email to a physician vs. showing a message on a screen) are defined separate from the knowledge that describes the guideline itself. For example, guidelines may contain an additional ‘layer’ that describes such actions, independent of the guideline process. This is supported by GLIF and EON as it is possible to describe multiple kinds of tasks for each action such as decision support-related or programming-related tasks.

Didactic and Maintenance

A guideline representation must be able to hold didactic and maintenance-related information such as author names, versions, (literature) references, sources and referees. Especially versioning-related information is very important, as guidelines are usually dynamic (the contents may change rapidly over time) and national guidelines may be adapted to local institutions.

Language

A guideline representation should define a formal language that is able to capture all the requirements mentioned above, in an unambiguous way. On the one hand, a representation must be abstract enough so that it is interpretable by Knowledge Acquisition Tools (KA-Tools) and guideline authors who do not have a logical or modeling background are able to define the process (e.g., flow), decision criteria and actions in a guideline (e.g., the decision criteria in GLIF). On the other hand, the representation must be interpretable 1) by verification tools to test guidelines and 2) by automatic parsers to execute guidelines (e.g., the L_{R2L} language in *PROforma*).

8.2.3 Guideline acquisition

A very important issue in the development of guidelines is the acquisition process. Each approach should be supported by KA-Tool (e.g., AsbruView, Protégé and the *PROforma* KA-Tool). Although based on a representation model, these tools must visualize guidelines from the viewpoint of a guideline author who may have little notion of the precise structure of the underlying language. The user interface of the KA-Tools must be flexible enough to visualize guidelines on different level of complexity (e.g., macros in GLIF must be visualized differently than entire flowcharts). Also, the various types of knowledge (e.g., domain knowledge, reasoning knowledge and supporting knowledge) must be visualized separately, depending on the role of the guideline author. Finally, mechanisms that support multi-user and version control must be provided.

8.2.4 Guideline verification and testing

To obtain unambiguous and syntactically as well as semantically correct guidelines, verification tools (e.g., the *PROforma* verification tools) must be provided to detect various kinds of errors such as errors concerning incompleteness, inconsistencies, conflicts and (partial) tautologies, invalid- or self-references and infinite loops. For example, in EON, action primitives can represent the starting and stopping of a certain drug. A guideline verification test should know and detect that these events are related. For example, it should not be possible to stop a drug before starting it. Therefore, a '*stop drug*' action is never supposed to occur before a '*start drug*' action. In GLIF,

every step that follows a branch step must always end at the corresponding synchronization step. Verification tests must be able to reason with temporal constructs in order to detect time-related errors. Finally, a simulation or test environment must be available where guidelines can be tested against actual or simulated patient data.

8.2.5 Guideline execution

As mentioned above, guidelines must be encoded in a format, interpretable by automatic parsers that are incorporated in guideline execution engines. Every approach must include such an engine that is able to execute guidelines in various environments. Therefore, the guideline engine must be able to interface with various clinical information systems in a consistent manner, for example by mapping concepts from the guideline to corresponding items in a clinical information system (e.g., the concept *Drug* in a guideline must be mapped to a drug table of an information system's database). Also, actions that a guideline performs must be configurable as they may differ in various local situations (e.g., send an e-mail in a certain situation in contrast to issuing an on-screen alert in another one). This implies a component-based approach in which each component performs a specific task such as reasoning or interfacing. The encoded format as well as the guideline execution engine must meet execution-time requirements such as compactness and execution speed.

8.3 Conclusions

In the last decade, most of the attention is focused on the areas of guideline representation models and underlying languages. However, the real benefit lays in structuring and guiding the whole guideline development process: in order to successively implement decision support systems that will be used in daily practice, all the four areas (representation, acquisition, verification and execution) must be taken into account. This is not a trivial task. Comparing the various approaches, mentioned in this paper shows that design specifications made in one area (e.g., guideline representation) have implications in other areas (e.g., guideline execution).

For example, Asbru defines a guideline representation language that has a very rich set of temporal constructs. However, a general guideline execution engine still has to be developed that can be used in daily practice. Another example is *PROforma* that focuses on guideline execution. This is reflected in the guideline model: each primitive in the *PROforma* task ontology can easily be mapped to a corresponding component in a guideline execution engine. However, during guideline acquisition, all guidelines have to be defined in terms of those primitives, which makes *PROforma* a more low-level language.

Although significant progress has been made during the last years, especially regarding guideline representation, several issues that relate to guideline implementation and guideline-based decision support still have to be addressed more extensively. Examples of such issues are how to implement national guidelines as well as local adaptations of those guidelines and how to increase the shareability of generic guideline execution engines among different intuitions. Various solutions may be developed that address these issues such as the development of versioning methods that enable synchronization between national and local guidelines and the development of standard interfaces to different external information systems.

In order to create an approach that is successful, an acceptable compromise between all areas must be reached with the above-mentioned requirements as starting points. In this compromise, a balance must be maintained between the aspects of abstractness, expressiveness, formalization, acquisition and execution.

References

1. Grimshaw JM, Russel IT. Effects of Clinical Guidelines on Medical Practice: A Systematic Review of Rigorous Evaluation. *Lancet* 1993;342:1317-22.
2. Effective Health Care. Implementing Clinical Practice Guidelines: Can guidelines be used to improve clinical practice? *Effective Health Care* 1994;8:1-12.
3. Audet A, Greenfield S, Field M. Medical practice guidelines: current activities and future directions. *Ann Intern Med* 1990;113:709-14.
4. Vissers MC, Hasman A, van der Linden CJ. Impact of a protocol processing system (ProtoVIEW) on clinical behaviour of residents and treatment. *Int J Biomed Comput* 1996;42(1-2):143-50.
5. East TD, Henderson S, Pace NL, Morris AH, Brunner JX. Knowledge engineering using retrospective review of data: a useful technique or merely data dredging? *Int J Clin Monit Comput* 1991;8(4):259-62.
6. Field MJ, Lohr KN (eds). *Guidelines for Clinical Practice: From Development to Use*. Washington, DC.: National Academy Press, 1992.
7. Van Der Lei J, Talmon JL. Clinical Decision-Support Systems. In: Van Bommel and Musen (eds). *Handbook of medical informatics*. Houten: Bohn Stafleu Van Loghum, 1997.
8. Fridsma DB, Gennari JH, Musen MA. Making Generic Guidelines Site-Specific. *Proc AMIA Symp* 1996;:597-601.
9. Position statements from the Invitational Workshop: Towards Representations for Sharable Guidelines. Available at <http://www.glif.org/workshop/statement.htm>.
10. Purves IN, Sugden B, Booth N, Sowerby M. The PRODIGY project--the iterative development of the release one model. *Proc AMIA Symp* 1999;:359-63.
11. Quaglini S, Stefanelli M, Cavallini A, Micieli G, Fassino C, Mossa C. Guideline-based careflow systems. *Artif Intell Med* 2000;20(1):5-22.
12. Herbert SI, Gordon CJ, Jackson-Smale A, Salis JL. Protocols for clinical care. *Comput Methods Programs Biomed* 1995;48(1-2):21-6.

13. Clayton PD, Pryor TA, Wigertz OB, Hripcsak G. Issues and structures for sharing knowledge among decision-making systems: The 1989 Arden Homestead Retreat. In: Kingsland LC (ed). Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care. New York: IEEE Computer Society Press. 1989;:116–21.
14. Ohno-Machado L, Gennari JH, Murphy SN, Jain NL, Tu SW, Oliver DE, Pattison-Gordon E, Greenes RA, Shortliffe EH, Barnett GO. The guideline interchange format: a model for representing guidelines. *JAMIA* 1998;5(4):357-72.
15. Fox J, Johns N, Rahmzadeh A. Disseminating medical knowledge: the *PROforma* approach. *Artif Intell Med* 1998;14:157-81.
16. Shahar Y, Miksch S, Johnson P. The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines. *Artif Intell Med* 1998;14:29-51.
17. Musen MA, Tu SW, Das A, Shahar Y. EON: A Component-Based Approach to Automation of Protocol-Directed Therapy. *JAMIA* 1996;3:367-88.
18. Wang D, Peleg M, Tu SW, Shortliffe EH, Greenes RA. Representation of Clinical Practice Guidelines for Computer-Based Implementations. *Medinfo* 2001;10:255-9.
19. Tu SW, Musen MA. A flexible approach to guideline modeling. *Proc AMIA Symp* 1999;:420-4.
20. Boxwala AA, Tu SW, Zeng Q, Peleg M, Ogunyemi O, Greenes RA, Shortliffe EH, Patel VL. Towards a Representation Format for Sharable Clinical Guidelines. *J Biomed Inform* 2001;34(3):157-69.
21. De Clercq PA, Blom JA, Hasman A, Korsten HHM. A strategy for development of practice guidelines for the ICU using automated knowledge acquisition techniques. *Int J Clin Monit Comput* 1999;15:109-117.
22. Pryor TA, Gardner RM, Clayton PD, Warner HR. The HELP system. In: Blum BI ed. *Information Systems For Patient Care*. New York: Springer Verlag, 1984; 109-28.
23. McDonald C, Overhage JM, Dexter PR, Tierney WM, Suico JG, Zafar A, Shadow G, Blevins L, Warvel J, Meeks-Johnson J, Lemmon L, Glazener T, Belsito A, Lindbergh D, Williams B, Cassidy P, Xu D, Tucker M, Edwards M, Wodniak C, Smith B, Hogan T. The Regenstrief Medical Record System 1999: Sharing Data Between Hospitals. *Proc AMIA Symp* 1999;(1-2):1212.
24. Hripcsak G, Ludemann P, Pryor TA, Wigertz OB, Clayton PD. Rationale for the Arden Syntax. *Comput Biomed Res* 1994;27(4):291-324.
25. Clinical Decision Support & Arden Syntax Technical Committee of HL7, inventor Arden Syntax for Medical Logic Systems, version 2.0. Draft revision. USA. July 7, 1999.
26. Pryor TA, Hripcsak G. Sharing MLMs: an experiment between Columbia-Presbyterian and LDS Hospital. *Proc Annu Symp Comput Appl Med Care* 1993;:399-403.
27. Sherman EH, Hripcsak G, Starren J, Jenders RA, Clayton P. Using intermediate states to improve the ability of the Arden Syntax to implement care plans and reuse knowledge. *Proc Annu Symp Comput Appl Med Care* 1995;:238-42.
28. Gao X, Shahsavar N, Arkad K, Ahlfeldt H, Hripcsak G, Wigertz O. Design and function of medical knowledge editors for the Arden syntax. *Medinfo* 1992;:472-7.
29. Jenders RA, Dasgupta B. Assessment of a knowledge-acquisition tool for writing Medical Logic Modules in the Arden Syntax. *Proc AMIA Symp* 1996;:567-71.
30. Bang M, Eriksson H. Generation of development environments for the Arden Syntax. *Proc AMIA Symp* 1997;:313-7.
31. Hripcsak G, Cimino JJ, Johnson SB, Clayton PD. The Columbia-Presbyterian Medical Center decision-support system as a model for implementing the Arden Syntax. *Proc Annu Symp Comput Appl Med Care* 1991;:248-52.

32. Kuhn RA, Reider RS. A C++ framework for developing Medical Logic Modules and an Arden Syntax compiler. *Comput Biol Med* 1994;24(5):365-70.
33. Hripcsak G, Clayton PD. User comments on a clinical event monitor. *Proc Annu Symp Comput Appl Med Care* 1994;:636-40.
34. The Intermed Collaboratory. Homepage available at <http://smi-web.stanford.edu/projects/intermed-web/>.
35. Peleg M, Boxwala AA, Ogunyemi O, Zeng Q, Tu S, Lacson R, Bernstam E, Ash N, Mork P, Ohno-Machado L, Shortliffe EH, Greenes RA. GLIF3: The Evolution of a Guideline Representation Format. *Proc AMIA Symp* 2000;:645-9.
36. Stoufflet PE, Ohno-Machado L, Deibel SRA, Lee D, Greenes RA. GEODE-CM: A state-transition framework for clinical management. In: Cimino JJ (ed). *Proceedings of the Twentieth Annual Symposium on Computer Applications in Medical Care*. Philadelphia: Hanley and Belfus, 1996:924.
37. Barnes M, Barnett GO. An architecture for a distributed guideline server In: Miller RA (ed). *Proceedings of the Nineteenth Annual Symposium on Computer Applications in Medical Care*. Philadelphia: Hanley and Belfus, 1995:233-7.
38. Object Management Group. Unified Modeling Language (UML) specification. Available at <http://www.rational.com/uml/index.jtmpl>. version 1.3; 1999.
39. Peleg M, Ogunyemi O, Tu SW, Boxwala AA, Zeng Q, Greenes RA, Shortliffe EH. Using Features of Arden Syntax with Object-Oriented Medical Data Models for Guideline Modeling. *Proc AMIA Symp* 2001;:523-7.
40. Schadow G, Russler DC, Mead CN, McDonald CJ. Integrating Medical Information and Knowledge in the HL7 RIM. *Proc AMIA Symp* 2000;:764-8.
41. Lindberg C. The Unified Medical Language System (UMLS) of the National Library of Medicine. *J Am Med Rec Assoc* 1990;61(5):40-2.
42. Pattison-Gordon E. ODIF: Object Data Interchange Format. Boston, MA: Decision Systems Group, Brigham and Women's Hospital; 1996. Report No.: DSG-96-04.
43. W3C. Extensible Markup Language (XML). Available at <http://www.w3.org/XML/>; 2000.
44. Advisory Committee on Immunization Practices A. Prevention and Control of Influenza. *Morbidity and Mortality Weekly Report* 2000;49(RR03):1-38.
45. Irwin RS, Boulet LS, Cloutier MM, Gold PM, Ing AJ, O'byrne P, et al. Managing Cough as a Defense Mechanism and as a Symptom, A Consensus Panel Report of the American College of Chest Physicians. *Chest* 1998;114(2):133S-181S.
46. American College of Cardiology/American Heart Association/American College of Physicians-American Society of Internal Medicine. Guidelines for the Management of Patients with chronic Stable Angina. *J Am Col Cardiol* 1999;33:2092-2197.
47. Grosso WE, Eriksson H, Fergerson RW, Gennari JH, Tu SW, Musen MA. Knowledge Modeling at the Millennium (The Design and Evolution of Protégé-2000). *Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Mangement (KAW'99)*, Banff, Canada, October 1999.
48. Greenes RA, Boxwala A, Sloan WN, Ohno-Machado L, Deibel SRAA. Framework and Tools for Authoring, Editing, Documenting, Sharing, Searching, Navigating, and Executing Computer-based Clinical Guidelines. *Proc AMIA Symp* 1999;:261-5.
49. Zielstorff RD, Teich JM, Paterno MD, Segal M, Kuperman GJ, Hiltz FL, Fox RL. P-CAPE: A High-Level Tool for Entering and Processing Clinical Practice Guidelines. *Proc AMIA Symp* 1998;:478-82.
50. Boxwala AA, Greenes RA, Deibel SRA. Architecture for a Multipurpose Guideline Execution Engine. *Proc AMIA Symp* 1999;:701-5.
51. Wang D, Shortliffe EH. GLEE - A Model-Driven Execution System for Computer-Based Implementation of Clinical Practice Guidelines. *Proc AMIA Symp* 2002;:855-9.

52. Fox J, Das S. The Logic of Medical Decision Making. In: Fox J, Das S. Safe and Sound: Artificial Intelligence in Hazardous Applications 2000: 31-54.
53. Fox J, Das S. Arguments about beliefs and Actions: Decision making in the Real World. In: Fox J, Das S. Safe and Sound: Artificial Intelligence in Hazardous Applications 2000: 55-76.
54. Fox J, Das S. The RED Knowledge Representation Language. In: Fox J, Das S. Safe and Sound: Artificial Intelligence in Hazardous Applications 2000: 191-206.
55. Fox J, Das S. Constructing Intelligent Systems. In: Fox J, Das S. Safe and Sound: Artificial Intelligence in Hazardous Applications 2000: 77-116.
56. Fox J, Das S. Safety First. In: Fox J, Das S. Safe and Sound: Artificial Intelligence in Hazardous Applications 2000: 191-206.
57. Fox J, Das S. A formalization of safety. In: Fox J, Das S. Safe and Sound: Artificial Intelligence in Hazardous Applications 2000: 77-116.
58. InferMed. Homepage available at <http://www.infermed.com>.
59. Miksch S, Shahar Y, Johnson P. Asbru: A Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans. Proceedings of the Seventh Workshop on Knowledge Engineering Methods and Languages (KEML-97), Milton Keynes, UK.
60. Chandrashekar B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. IEEE Expert 1986;1:23-30.
61. Asbru 7.2 syntax. Available at <http://www.ifs.tuwien.ac.at/asgaard/asbru/#Syntax>.
62. Miksch S, Kosara R, Shahar Y, Johnson PD. AsbruView: Visualization of Time-Oriented, Skeletal Plans. The Fourth International Conference on Artificial Intelligence Planning Systems 1998, Carnegie-Mellon University, Pittsburgh, PA, 11-18. 1998.
63. Kosara R, Miksch S. Metaphors of movement: a visualization and user interface for time-oriented, skeletal plans. Artif Intell Med 2001;22(2):111-31.
64. Duftscheid G, Miksch S. Knowledge-based verification of clinical guidelines by detection of anomalies. Artif Intell Med 2001;22(1):23-41.
65. World Wide Web Consortium, Resource Description Framework (RDF). Available at <http://www.w3.org/RDF/>
66. Knowledge Interchange Format: Draft Proposed American National Standard (dpANS). 1998.
67. Nguyen JH, Shahar Y, Tu SW, Das AK, Musen MA. Integration of Temporal Reasoning and Temporal-Data Maintenance into a Reusable Database Mediator to Answer Abstract, Time-Oriented Queries: The Tzolkín System. Journal of Intelligent Information Systems 1999;13(1-2):121-45.
68. Tu SW, Musen MA. Modeling Data and Knowledge in the EON Guideline Architecture. Medinfo 2001;10:280-4.
69. Tu SW, Musen MA. From guideline modeling to guideline execution: defining guideline-based decision-support services. Proc AMIA Symp 2000;:863-7.
70. Shankar RD, Musen MA. Justification of Automated Decision-Making: Medical Explanation or Medical Argument? Proc AMIA Symp 1999;:395-9.
71. Goldstein MK, Hoffman BB, Coleman RW, Musen MA, Tu SW, Advani A, Shankar R, O'Connor M. Operationalizing Clinical Practice Guidelines While Taking Account of Changing Evidence: ATHENA, an Easily Modifiable Decision-Support System for Management of Hypertension in Primary Care. Proc AMIA Symp 2000;:280-4.
72. Bury JP, Saha V, Fox J. Supporting 'scenarios' in the PROforma guideline modeling format. Proc AMIA Symp 2001;:870-4
73. Spackman KA, Campbell KE, Cote RA. SNOMED RT: a reference terminology for health care. Proc AMIA Symp 1997;:640-4.

CHAPTER 3

THE APPLICATION OF PROBLEM-SOLVING METHODS AND ONTOLOGIES FOR THE DEVELOPMENT OF SHAREABLE GUIDELINES

Published in:
Artificial Intelligence in Medicine 2001;22(1):1-22

Paul A. de Clercq
Arie Hasman
Johannes A. Blom
Hendrikus H.M. Korsten

1 Introduction

1.1 Overview

Recently, studies have shown the benefits of using clinical guidelines in the practice of medicine [1]. Utilizing guidelines such as standard care plans, critical pathways and protocols in various clinical settings may lead to the reduction of practice variability and patient care costs, while improving patient care [2]. Computer-based clinical guidelines are increasingly applied in diverse areas such as policy development, utilization management, education, conduct of clinical trials, and workflow facilitation. Many parties are developing computer-based guidelines as well as decision support systems that incorporate these guidelines [3]. There is little standardization to facilitate sharing of guidelines or to enable adaptation to local practice settings [4]. However, currently efforts are made to introduce standardized approaches for guideline representation and sharing [5]. This paper discusses some of the suggested representations and discusses their weak and strong points, and demonstrates and discusses a new approach that extends earlier-developed formalisms. The overall goal of this approach is to improve the acceptance of shareable guidelines and decision support systems in daily care by facilitating the guideline acquisition and execution phases.

1.2 Guideline representation formalisms

Requirements for a sharable guideline representation language have been formulated [6, 7] and include the possibility to represent temporal logic, branching and sequencing, patient data elements, (eligibility) criteria, actions and decompositions of actions.

1.2.1 Modeling in terms of primitives

A common approach to satisfy these requirements has been to model guidelines in terms of primitives that represent steps such as actions, decisions and plans. Examples of these representations are the Arden Syntax [8], *Proforma* [9], the GuideLine Interchange Format (GLIF) [6], Asbru [10] and EON [7]. The Arden Syntax, a language intended as an open standard for the procedural representation and sharing of medical knowledge, does not satisfy all above-mentioned requirements. It defines a representation for modular decision rules that are encoded as Medical Logic Modules (MLMs). Each MLM contains a production rule that relates a set of input conditions (e.g., patient data from a hospital information system) to a particular set of actions to take (e.g., send reminders or alerts to a clinician). But since MLMs include only terms and no further qualifying domain knowledge in the rule's premise, it is difficult to reason about a domain in terms of clinical concepts and

strategies used to solve problems in that domain [11]. Nevertheless, the Arden Syntax has been accepted as a standard by a large number of researchers and developers in the medical community.

In order to meet the above-mentioned requirements more fully, representations such as GLIF and *Proforma* were developed that define a richer set of primitives (e.g., primitives that represent decisions, branching and actions). In addition, languages were developed that are able to reason with complex temporal logic (e.g., Asbru and EON).

1.2.2 Modeling in terms of Problem-Solving methods

In parallel, various research groups developed representation formalisms that concentrated on the abstract behavior of decision supports systems in general. These formalisms express the notion that the behavior of decision support systems can be described by means of two independent classes of reusable components: 1) domain ontologies and 2) Problem-Solving Methods or PSMs.

Domain ontologies provide a domain of discourse [12]; they model entities and relationships for a particular domain of interest such as intensive care or psychiatry. Problem-solving methods represent generic strategies to solve stereotypical tasks, independent of the system's application domain [13]. Clancey, for example, identified heuristic classification as a recurring strategy in various rule-based systems such as MYCIN [14]. PSMs such as heuristic classification are role-limiting by nature [15], meaning that the PSMs impose specific problem-solving roles on domain knowledge. These problem-solving roles are referred to as knowledge roles, which give an abstract description of the function domain knowledge has to play. When refining a PSM to a certain domain, the knowledge roles are mapped onto domain knowledge. PSMs can be reused to solve similar problems in different application domains by using different domain ontologies. PSMs are decomposable into subtasks, which can be executed by submethods. When no longer decomposable, a submethod is referred to as a primitive PSM or mechanism.

During the last decade, a number of different approaches have been developed to represent the behavior of decision support systems in terms of domain ontologies and PSMs (although each one is based on specific viewpoints and methodologies). Well-known approaches are *CommonKADS* [16], OCML [17], Protégé [18] and UPML [19].

1.2.3 Primitives vs. PSMs

When comparing guidelines that consist of primitives to guidelines that consist of PSMs, each approach has its strong points as well as its shortcomings. An advantage of the PSM-based approach is the separation of domain-specific knowledge and domain independent methods, which increases the reusability and shareability. A primitive is more difficult to reuse because often domain and procedural knowledge are intertwined. Also, as PSMs pre-define the global control structure, in a Knowledge Acquisition Tool or KA-Tool the author only has to specify the knowledge component [20, 21]. This is in contrast to the primitive-based approach where also the control structure has to be explicitly stated.

However, most of the guideline representations that are used in clinical practice today do not use the notions of PSMs or ontologies, for several reasons. As PSMs are domain-independent representations, the visualization of a PSM through a KA-Tool may be too abstract for a domain expert to enter domain knowledge efficiently. Most importantly, however, certain types of protocols used in daily practice do not easily fit the highly formalized formats used in a PSM.

1.2.4 Combining primitives and PSMs: a new approach

In order to represent and implement guidelines in various application domains, this paper argues that the granularity and abstraction level of a guideline representation formalism must reflect the guideline's characteristics. It defines a new approach, in which a guideline can be represented in terms of 1) primitives to construct the guideline's control structure explicitly as well as 2) PSMs to model guidelines that perform stereotypical tasks. Also, guidelines may contain subguidelines in order to solve multiple tasks. Two types of ontologies are defined: domain ontologies and method ontologies. As mentioned earlier, domain ontologies model domain-specific knowledge in terms of entities, attributes and relations. Method ontologies [22] model concepts such as primitives, PSMs and guidelines similarly. Furthermore, our approach also defines a method library, which consists of a number of available PSMs.

Primitives are used 1) to describe single guideline steps, and 2) to describe the internal structure of PSMs. The model is non-monolithic, meaning that ontologies can be extended to capture new guideline characteristics. The remaining part of this paper describes this approach. Section 2 presents an overview of the ontological representation. Section 3 presents a framework that facilitates the guideline acquisition and execution stages. Examples of PSMs and guidelines that were developed by means of this framework are

shown in section 4. Section 5 discusses a number of developed systems and section 6 discusses the approach in comparison with other ones.

2 The ontological guideline representation

2.1 Representing domain knowledge

To represent domain ontologies, the Entity-Relationship (ER) model [23] was used. Figure 1 shows a section of a domain ontology that was developed by means of the ER model. This particular domain ontology was designed for use in Intensive Care Units (ICUs) and consists of entities, relations and attributes related to the ICU domain such as drugs, diseases and treatments and relationships such as the `Has_Interactions` relation.

Domain_Entity	
.....Treatment	
.....Drug	Drug
.....Antibiotic	Name -> string
.....Circulation	ID -> number
.....Acetylcysteine	Dose -> number
.....Adenosine	Unit -> symbol (values: mmol/l, mg/kg, mg/pill)
.....Amiodarone	Has_Interactions ->* Interaction_Relation
.....Amlodipine	
.....Beta-Blocker	
.....Cardio-surgical	
.....Disease	Interaction_Relation
.....Indication	Name -> string
.....Laboratory_Test	Target -> Drug
Relation_Entity	Severity -> symbol (values: normal, severe, contraindicated)
.....Interaction_Relation	

Figure 1: Part of a domain ontology. The left column shows a class hierarchy of entities that describe a particular domain (ICU). The right column presents a more detailed view of two classes and their attributes (not all attributes are shown). Each attribute has a type such as integer, string or symbol and is by default inherited by the subclasses (attributes that are inherited from other classes are shown in italic). In this example, the `Interaction_Relation` class models an interaction between two drugs. By means of the `Interaction_Relation`'s `Severity` attribute, each interaction can be characterized as normal, severe or contraindicated. Attributes may refer to one instance (e.g., each drug has only one dose) or multiple instances (e.g., each drug may have various interactions). If an attribute refers to multiple instances, an asterisk follows the arrow

Concepts in a domain ontology may also contain references to patient records or terminology servers where the actual data can be found during the execution of a guideline. Although this enables the reuse of domain ontologies (e.g., a single domain ontology can be linked to multiple patient record systems or terminology servers), incompatibilities may exist between concepts from the domain ontology and concepts from a patient record or terminology server. This problem is also referred to as the mapping problem [24].

2.2 Representing guidelines

2.2.1 Method ontologies

As mentioned earlier, guidelines are represented by a set of primitives or by means of a PSM. Similar to domain ontologies that describe domain-specific knowledge, method ontologies can be used to model primitives, PSMs and guidelines in terms of entities, attributes and relations. A core method ontology was developed that defines the characteristics of a primitive, a PSM, a guideline and related concepts. Figure 2 shows a part of the core method ontology that was developed to model various categories of guidelines.

Guideline_Entity	Primitive
.....Guideline	Name -> string
.....PSM	Author -> string
.....Primitive	Explanation -> string
.....Control_Primitive	Goal -> K_Of_N_Criteria
.....Decision	Visualization -> Visualization_Definition
.....Boolean_Criterion	Parameters ->* Intermediate_Role
.....K_Of_N_Criteria	Procedure -> Procedure_Definition
.....Branching	
.....Synchronization	PSM
.....Action	Name -> string
Knowledge_Role	Author -> string
.....Input_Role	Explanation -> string
.....Output_Role	Goal -> K_Of_N_Criteria
.....Intermediate_Role	Visualization -> Visualization_Definition
Definition	Control -> Control_Structure
.....Procedure_Definition	Description -> string
.....Visualization_Definition	Knowledge_Roles ->* Knowledge_Role
Control_Structure	Guideline
	Name -> string
	Author -> string
	Explanation -> string
	Goal -> K_Of_N_Criteria
	Visualization -> Visualization_Definition
	Control -> Control_Structure
	Task_Description -> string
	Validation -> symbol (values: test, Production)
	Target_Users -> string
	Eligibility_Criteria -> K_Of_N_Criteria
	Abort_Criteria -> K_Of_N_Criteria

Figure 2: A section of the core method ontology that describes the guideline model.

The left column shows a hierarchy of classes that represents primitives, PSMs, guidelines and related concepts. It also presents a number of primitives that are used to describe single guideline steps such as decisions and actions. The right column shows the three main classes in detail (again, not all attributes are shown).

2.2.2 Primitives

In the guideline model, primitives represent both non-decomposable parts in a guideline (e.g. decisions and actions) similar to earlier-mentioned representations, and non-decomposable parts in PSMs. These primitives are based on version 2.0 of GLIF [6]. This specification (which originates from an earlier version of EON) defines four types of primitives that are commonly used to describe guidelines, such as 1) *Action* primitives that specify clinical

actions, 2) `Decision` primitives that model decision points in a guideline, 3) `Branching` primitives that direct the guideline flow to multiple (parallel) paths and 4) `Synchronization` primitives that converge paths that previously diverged by means of a `Branching` primitive. The `K_of_N_Criteria` primitive for example, derived from the `Decision` primitive represents a logical statement that directs the flow of the guideline depending on its evaluation (true or false). The statement contains a number of criteria and is evaluated as true if at least a certain number (κ) of all criteria (N) is true.

In addition to the attributes that are administrative in nature (e.g., name, author and an explanation), a primitive in the method ontology also contains additional attributes such as the `Parameters`, `Visualization` and `Procedure` attributes, that define the primitive's capabilities. Parameters are intermediate roles, used to further define primitives but also may contain mappings to parameters from other primitives, to concepts from the domain ontology or to knowledge roles from a PSM. Visualization information is used by a system developer to define the characteristics of a primitive-specific user interface in a KA-Tool. Finally, the `Procedure` attribute contains execution-time information (executable code), used by a decision support system that incorporates the primitive.

2.2.3 Problem Solving Methods

PSMs model stereotypical processes that may occur in a guideline (e.g., heuristic classification or risk-assessment). A PSM differs from a primitive in a number of aspects. First, PSMs define input and output knowledge roles, used for communication outside the PSM (e.g., to exchange information with another PSM that uses this PSM to solve a subtask). It also contains a description of the used strategy (in the current version of the ontological model, this description is stated in an informal way) and a goal that formally describes the goal of the PSM. In contrast to primitives, PSMs have a control structure that describes the internal structure of the PSM in terms of subcomponents. This structure may refer to subtasks (that are solved by other PSMs), but also to primitives. Similar to primitives, PSMs also contain visualization information defining a specific user interface for use in a KA-Tool. However, as the PSM has access to all the visualization information of the subcomponents in the control structure (in terms of knowledge roles), this information can be used by the PSM to construct user interfaces by combining visualization information from all subcomponents, as will be illustrated later. The actual implementation procedure of each primitive is hidden from the PSM.

2.2.4 Guidelines

The `Guideline` class describes an entire (sub)guideline, which is associated with a task. This task can be solved by processing a set of primitives or by selecting an appropriate PSM. Similar to a PSM, the internal structure of a guideline is described by a control structure (in terms of subcomponents). The control structure of a guideline contains a set of primitives or a reference to a single PSM. In case of the PSM, the rationale behind this limitation is that each guideline must solve a task, which is executed by a single PSM (the PSM however, can use subtask decomposition to solve the task). In order for a guideline to solve multiple tasks, subguidelines are used.

The visualization information of a guideline 1) creates a flowchart in case the control structure consists of a number of elements or 2) utilizes the visualization information of the PSM in case the guideline consists of a single PSM.

Furthermore, the `Guideline` class defines several guideline-specific attributes such as a task attribute that (informally) describes the task that has to be solved, eligibility criteria that may evoke a guideline, abort criteria that may abandon it and temporal criteria (e.g., this guideline is to be executed 4 times a day). Other guideline-specific attributes are a `validation` attribute that indicates whether the guideline has been approved for routine use (`production`) or is still in the test phase (`test`), and a `Target_Users` attribute that denotes the intended users of the guideline (e.g., administrators, physicians or nurses). Finally, a guideline also contains an attribute that formally defines the goal of the solved task.

3 The framework

To facilitate the representation and development of guidelines and corresponding decision support systems by means of primitives, PSMs and domain ontologies, a framework that supports this methodology has been developed. This framework consists of a suite of tools that support the various stages in guideline development. Figure 3 shows the process view of the framework.

The process consists of four stages:

1. Develop, derive or reuse application-specific domain and method ontologies.
2. Develop or reuse libraries of PSMs.
3. Develop guidelines in terms of PSMs and primitives with a KA-Tool.

4. Automatically translate these guidelines into a more efficient symbol-level representation, which can be read in and processed by an execution-time interpreter.

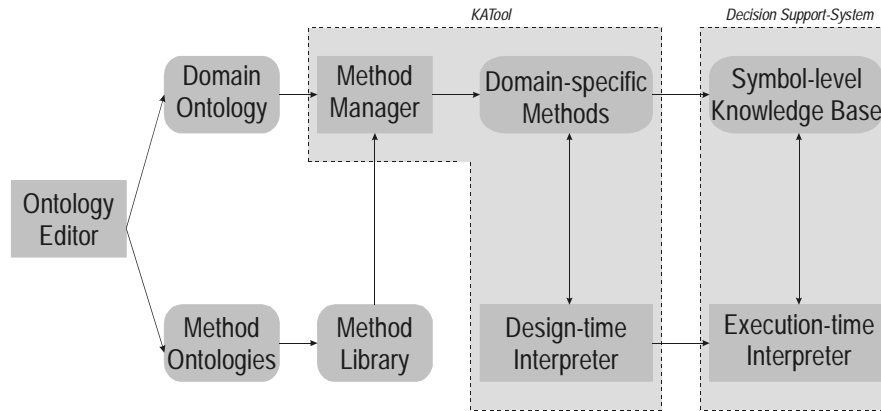


Figure 3: Process view of the framework. Rounded rectangles represent models (e.g., ontologies), straight rectangles represent modules (e.g., programs)

Several tools support each stage. The ontology editor, developed in the Protégé project [18] was used to facilitate the development of domain and method ontologies. A separate KA-Tool was developed (not automatically generated by Protégé), consisting of a kernel of which the functionality is extended by loading additional plug-ins. Finally, a run-time environment was developed that executes guidelines, acquired through the KA-Tool. A more detailed and technical description of the framework can be found elsewhere [25].

The first stage involves developing or reusing domain and method ontologies by defining hierarchies of entities, attributes and relations. Depending on the requirements of the method ontology, existing domain ontologies can be extended with new attributes or relations.

A method library is a collection of available PSMs and primitives created in the second stage that can be used in the third stage for the definition of guidelines to solve certain tasks. As PSMs usually describe rather abstract problem-solving behavior, a knowledge engineer uses the method manager to define application-specific PSMs by refining the PSM's knowledge roles. In addition, the method manager supports the creation of not only application-specific but also domain-specific methods by mapping concepts from the domain ontology onto the corresponding knowledge roles in the method ontology.

The third stage utilizes the KA-Tool, containing a design-time interpreter that loads the required primitives and domain-specific methods and creates a user interface that enables guideline authors to develop guidelines. Primitives are

represented in a flowchart, whereas PSMs are visualized by utilizing the visualization information of the PSM itself. The latter enables guideline authors to enter domain information regarding the corresponding task without any knowledge of the internal control structure of the PSM.

When instructed, the KA-Tool combines the control structure of each guideline and PSM and creates a structure that consists solely of primitives, creating a symbol-level knowledge base that is processed by an execution-time interpreter that executes the implementation modules that are attached to each primitive.

4 Examples

This section describes three examples of guidelines -developed using the framework described in this paper- to illustrate the versatility of this methodology. The first example shows the representation of rule-based guidelines (e.g., MLMs) by means of the framework, whereas the second example shows the application of a selection PSM to solve common domain-independent tasks (e.g., the detection of drug interactions). Finally, the third example shows a complex hypertension guideline that consists of a number of subguidelines, some of which contain only primitives, whereas others use a single PSM to solve the subguideline's task.

4.1 Situation-Action Rules

4.1.1 Representing Situation-Action Rules

The Situation-Action Rule (SAR) model represents a guideline by means of a production rule that performs an action (e.g., generating a message or writing data to a database) whenever the premise of a rule evaluates to true (*'IF conditions THEN Action'*). This model is very natural for certain classes of decision support systems such as reminder systems [26, 27]. All SARs share an identical format and perform similar actions. Therefore, it is possible to define the control structure of a SAR in terms of primitives, similar to a PSM. However, a SAR is not a PSM in the true sense of the definition, as it does not utilize a specific strategy to solve a class of tasks, but uses an inference method (e.g., forward chaining) to process a collection of rules. Nevertheless, we have included this example to illustrate that with our approach it is possible to define rule-based guidelines (which is still a very common approach in the medical community) as well as complex PSMs and guidelines by means of the same representation and methods.

In terms of a general inference strategy, a SAR performs the following steps:

1. Validate all conditions. If a condition requires the execution of another SAR (referred to as an intermediate rule in the SAR model), execute that rule first.
2. If all conditions are satisfied, perform a certain action such as generating a reminder or executing another SAR.

From this description it is clear that SARs are not PSMs, as they contain no particular problem-solving strategy, but rely on a rule-specific inference mechanism (e.g., forward chaining). Translating this strategy into a control structure in terms of primitives is rather straightforward, as there are only two steps to be taken in order to execute each SAR guideline: . Therefore, only two classes are needed to execute each SAR: the `K_of_N_Criteria` class (in which $\kappa=N$) and (a subclass of) the `Action` class. For this purpose, the earlier-discussed core method ontology was extended with classes that represent actions. Figure 4 shows part of an extended method ontology, used for defining SARs.

<pre> Guideline_EntityGuidelinePSMPrimitiveControl_PrimitiveDecisionBoolean_CriterionK_Of_N_CriteriaBranchingSynchronizationActionOutputShow_ReminderCondition </pre>	<pre> K_Of_N_Criteria Name -> string Author -> string Explanation -> string Satisfied_Step -> Guideline, Primitive Otherwise_Step -> Guideline, Primitive Criteria ->* Condition Show_Reminder Name -> string Author -> string Explanation -> string Reminder_Message -> string </pre>
---	---

Figure 4: An extended method ontology, for defining SARs. Classes from included ontologies are shown in italics. The `K_of_N_Criteria` class is inherited from the core ontology (in contrast with figure 3, the capability attributes are not shown here). It models a decision based on a number of criteria by means of a `Criteria` attribute that holds the given criteria, and the `Satisfied_Step` and `otherwise_Step` that are references to (sub)guidelines or primitives that may follow the `K_of_N_Criteria` primitive. The `Condition` class is an auxiliary class that contains the mappings to the applied domain concepts. Furthermore, this ontology also defines a `Show_Reminder` action used by reminder systems to generate advice

This particular method ontology was developed for use in the CritICIS system, a real-time reminder system that provides decision support in ICUs [28]. The ontology extends the core method ontology with a `Show_Reminder` primitive, which models the action of issuing a reminder.

4.1.2 Authoring Situation-Action Rules

Each primitive contains information to visually represent itself in a KA-Tool by means of mapping knowledge roles onto the user interface of the KA-Tool. From the viewpoint of the guideline's author, each SAR performs a single task, of which (s)he only has to specify the conditions and desired actions. Based on the control structure and visualization information of the SAR strategy, the KA-Tool design-time interpreter is able to construct a user interface. It provides a means for entering conditions and actions. For example, figure 5 shows the KA-Tool containing the domain ontology and SAR guidelines for the CritICIS system.

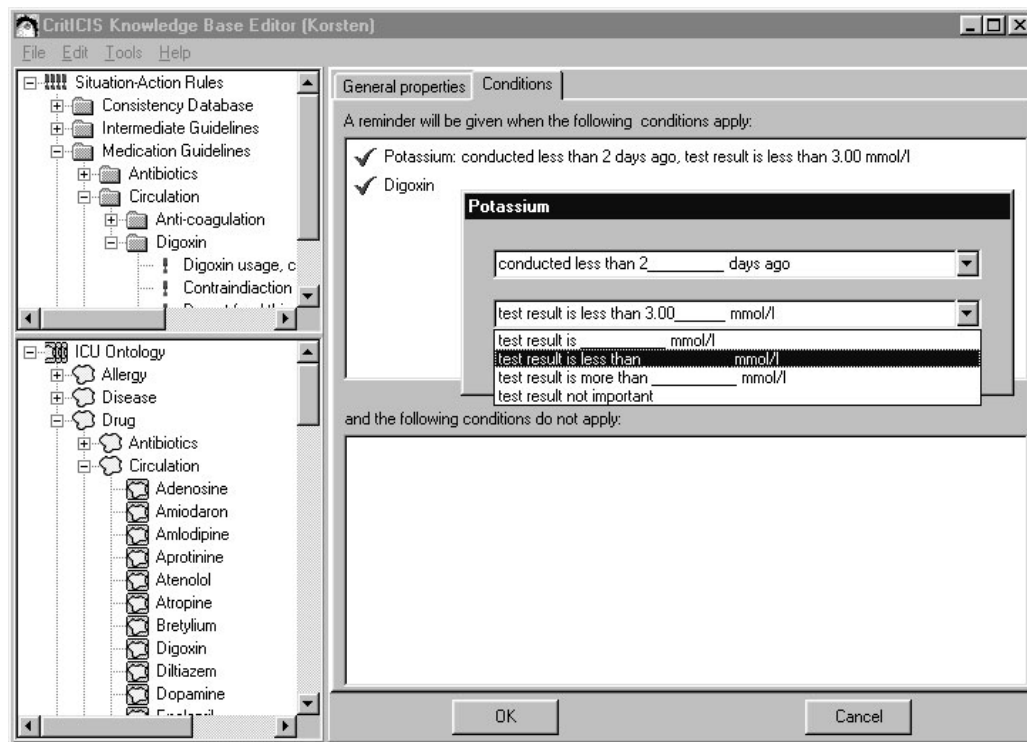


Figure 5: The conditions of a SAR in the KA-Tool. The Potassium pop-up window shows the attributes of the potassium laboratory test

The KA-Tool consists of three panes. The upper left pane presents an overview of all designed guidelines, whereas the lower left pane shows all concepts defined in an application-specific domain ontology. Whenever a guideline is selected in the upper left pane, the design-time interpreter constructs a user interface, shown in the right pane. The design-time interpreter hides the guideline structure and maps all attributes onto two pages: a 'General properties' page and a 'Conditions' page. In the first page (not shown here) all common attributes (e.g., name, author and explanation) can be entered, as well as a reminder message (linked to an instance of the

`Show_Reminder` class). The second page presents all conditions, taken from the `K_of_N_Criteria` instance. In this example, all conditions are satisfied when a laboratory test, carried out less than two days ago, returns a potassium concentration that is lower than 3 mmol/l and the drug Digoxin has been prescribed as well. The conditions consist of instantiated entities from the domain ontology, which are selected from the domain ontology pane and dragged to the right pane. This operation implements the AND-operator. The OR-operator is implemented by dragging a domain entity from the domain ontology pane atop an existing condition in the right pane. Finally, it is also possible to edit the attribute values of a domain entity.

4.1.3 Executing Situation-Action Rules

As mentioned earlier, each primitive contains a reference to an implementation procedure. The execution-time interpreter uses this information when executing guidelines. In this example, the execution-time interpreter executes the procedures corresponding to the `K_of_N_Criteria` and `Show_Reminder` primitives. Figure 6 shows a reminder, generated by the CritICIS system.

The screenshot shows a Microsoft Access window titled "Microsoft Access - [Medication]". The main window displays a patient's medication list for Patient: 13043255510 Smith. The list includes columns for Medicine, Freq., Unit + Pharm. form, Administration, Begin, End, and Day. A reminder dialog box is overlaid on the window, titled "Reminder regarding patient Smith (Room 1, Bed 2)". The dialog box contains a warning icon and the text: "The combination of a low potassium value (< 3 mmol/l) and digoxin can be potentially hazardous for the patient." Below this, it states: "This patient has a potassium concentration, less than 3 mmol/l and receives the drug digoxin. However, this combination can be potentially hazardous for the patient. More information about this topic can be found [here](#)." The dialog box has buttons for OK, Cancel, and <<Less. The background window shows a list of medications including Bretylium, Cefazidim, Allopurinol, Angiotensin 2, Atropine, Labetalol, Digoxine, and Allopurinol. A menu on the right side of the window lists various options: Barcodes, Complication, Treatment, Examination, Medication, Transfusion, Reports, View reports, Monitor, Lab, SOP, and TISS. At the bottom of the window, there are buttons for Add Medication, Stop Medication, Pumpdose, OK, Change Medication, and Info.

Medicine	Freq.	Unit + Pharm. form	Administration	Begin	End	Day
Bretylium		μG/KG/Min	5	mg/kg	i.v.	5/28/98 KTN
Cefazidim	3	X DGS	1000	mg	i.v.	5/28/98 KTN
Allopurinol	1	μG/KG/Min	300	ml	i.v.	5/28/98 KTN
Angiotensin 2		2.5	mg	i.v.	5/28/98 KTN	
Atropine	BONUS	0.5	mg/ml	i.v.	5/28/98 KTN	
Labetalol	VLG.AFSPH	1	mg/ml	i.v.	5/28/98 KTN	
Digoxine	1					
Allopurinol	1					

Figure 6: A reminder generated by the CritICIS system. When the SAR's conditions evaluate to true, the reminder system generates and displays a reminder, shown overlaying the user interface of a specific Computer-based Patient Record system

4.2 Event-Based Modular Tasks

4.2.1 Representing Event-Based Modular Tasks

Representing complex guidelines is usually more difficult than representing 'simple' reminders [11]. Van der Lei and Musen developed a model that defines four particular classes of common domain-independent tasks such as selection tasks, preparation tasks, monitoring tasks and responding tasks [29]. All tasks that fall into these four classes are characterized as Event-Based Modular Tasks (EBMTs) [30], as each task is modular and can be solved by means of a sequence of steps that are executed whenever a pertinent event occurs (e.g., starting a new drug). This event is usually generated by an external source such as a Computer-based Patient Record (CPR). These tasks can also be used for guideline implementation.

The remaining part of this paragraph uses the class of selection tasks as an example. Selection tasks check whether a physician's selected action or decision is appropriate, and if not provide feedback. An example of a selection task is the detection of drug interactions. A selection PSM with the following general strategy was developed to solve these tasks:

1. Determine the selected treatment. As PSMs that solve EBMTs are triggered via a pertinent event, this step is automatically executed first.
2. Conclude which constraints are violated. For example, are there current treatments that are not compatible with the selected treatment?
3. Report all violated constraints. This PSM does not try to 'fix' violated constraints (as is done for example by the propose-and-revise PSM), but merely reports violated constraints to the user that started the treatment. This approach is usually applied in critiquing systems [31], which is the type of decision support systems intended by the developers of the selection tasks.

Further specification of three knowledge roles refines this PSM: 1) specification of the treatment, 2) specification of the constraints, and 3) specification of the way in which violations are reported. The first two are input roles, whereas the last one is an output role.

The drug interactions task warns against unwanted combinations of drugs. By specifying the required knowledge roles, the selection PSM is able to report drug interactions. In order to solve this particular task, the PSM must take the following steps:

- Determine all drugs that have known interactions with the newly prescribed drug
- Determine all drugs that are being prescribed
- Report all drugs that are known interactions as well as being prescribed as interactions

The first step requires access to a domain ontology to determine all possible interactions, whereas the second step requires access to the CPR to obtain all prescribed drugs. Finally, the third step issues a reminder. To describe the control structure of the selection and similar PSMs, the method guideline ontology was extended with new primitives, as shown in figure 7.

<pre> Guideline_EntityGuidelinePSMPrimitiveControl_PrimitiveDecisionBoolean_CriterionK_Of_N_CriteriaBranchingSynchronizationActionSet_OperatorAdd_Entities_By_ExternalAdd_Entities_By_RelationLogical_OperatorOutputShow_ReminderGenerate_Advice_From_EntitiesSet </pre>	<pre> Logical_Operator Name -> string Author -> string Explanation -> string Input_Sets ->* Set Output_Set -> Set Operation -> symbol (values: OR, AND, XOR, NOT) Generate_Advice_From_Entities Name -> string Author -> string Explanation -> string Entities_Set -> Set Message_Mask -> string </pre>
--	---

Figure 7: A section of the derived EBMT method ontology. It defines various primitives that represent operations on sets of domain entities, such as `Add_Entities_By_External`, which is used to obtain a set of prescribed drugs from an external source (e.g., CPR), `Add_Entities_By_Relation`, used to determine all known interactions of the newly prescribed drug, and `Logical_Operator`, used to determine the conjunction of two sets. Furthermore, the ontology also defines the `Generate_Advice_From_Entities` primitive, which is used to report known interactions. Finally, The `Set` class is a data structure, used to store a number of domain entities (e.g., drugs)

Figure 8 presents a part of the control structure of the refined selection PSM in terms of used primitives. By applying different refinements, this PSM could be used to solve all selection tasks. Besides selection tasks, other EBMTs such as monitoring tasks, preparation tasks and responding tasks can also be solved by PSMs, similar to the one that was used to solve the selection tasks. The example, presented in this section, describes a PSM that was refined with simple straightforward mappings (although even this example has been somewhat simplified for the convenience of the reader). In many situations however, mapping terms from one (domain or method) ontology to another is

not very straightforward. Problems arise if mappings are not one-to-one or, even worse, there are semantic differences between the various ontologies [32].

```

Branching_1 of Branching
  (Parameters:Selection_Method = all_off
    Order_Constraint = any_order)
    Branches = Add_Entities_By_Relations_1,
              Add_Entities_By_External_1)

Add_Entities_By_Relations_1 of Add_Entities_By_Relations
  (Parameters:Source_Class = Drug
    Source_Relation = Has_Interactions)
    Output_Set = Set_1
    Successor = Synchronization_1)

Add_Entities_By_External_1 of Add_Entities_By_External
  (Parameters:Source_Class = Drug
    Output_Set = Set_2
    Successor = Synchronization_1)

Synchronization_1 of Synchronization
  (Parameters:Continuation = wait_for_all
    Successor = Logical_Operator_1)

Logical_Operator_1 of Logical_Operator
  (Parameters:Input_Sets = Set_1,Set_2
    Operation = AND
    Output_Set = Set_3)

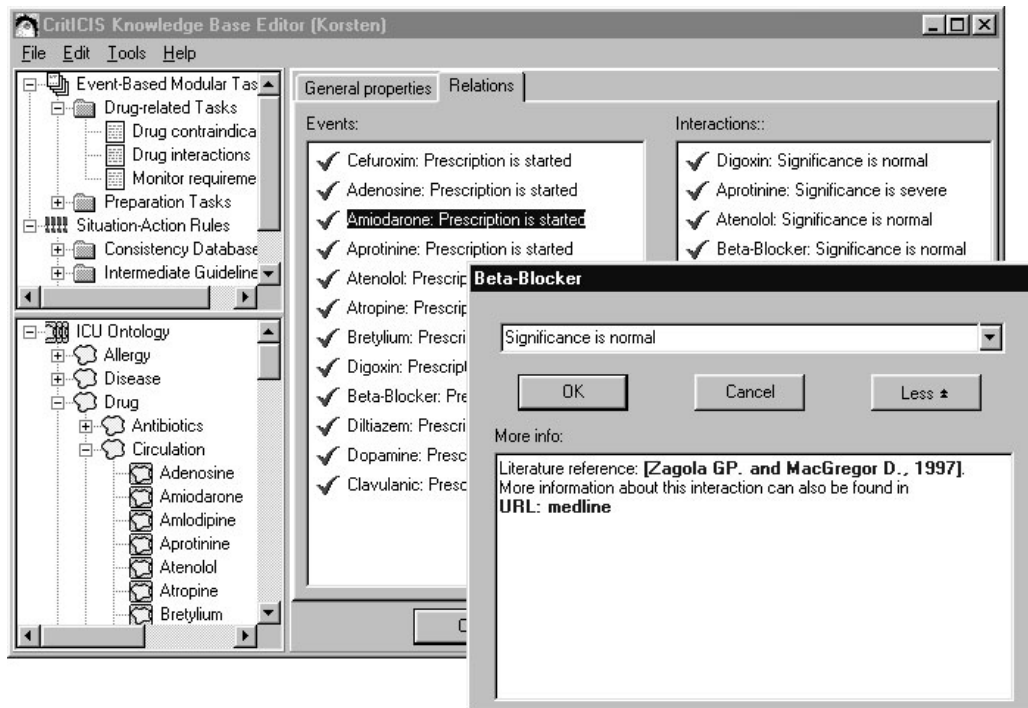
Generate_Advice_From_Entities_1 of Generate_Advice_From_Entities
  (Parameters:Entity_Set = Set_3
    Message_Mask = '%event.name% and %Entity_Set.name%
                  are known interactions')

```

Figure 8: The control structure of the selection PSM, refined to report drug interactions. The control structure of the selection PSM consists of six primitives. Each primitive is characterized by means of parameters, which may refer to global structures that contain information that is used throughout the control structure (e.g., Set_1, Set_2 and Set_3). The Branching and Synchronization primitives are used to determine all known interactions as well as prescribed drugs (details on the attributes of the Branching and Synchronization classes are described elsewhere [6]). The Add_Entities_By_Relations_1 instance collects all known interactions by means of the Has-Interactions relation whereas the Add_Entities_By_External_1 instance collects all prescribed drugs from the CPR. The Logical_Operator_1 instance determines all known interactions that are also prescribed, which are reported by the Generate_Advice_From_Entities_1 instance (the reminder message is generated from the Message_Mask attribute). Set_1, Set_2 and Set_3 are instances of the Set class

4.2.2 Authoring Event-Based Modular Tasks

Although the control structure of PSMs that are used to solve EBMTs is more complex than that of SARs, each PSM still performs a single task from the viewpoint of the guideline's author. Figure 9 shows the corresponding KA-Tool (also taken from the CritICIS system). It uses the visualization information from the PSM and provides a means for entering domain-specific knowledge such as drugs and their interactions.



*Figure 9: The user interface generated by the selection PSM to represent the drug interaction task. It represents this PSM by means of an Events pane and an Interactions pane, where each entity in the Event pane denotes a newly prescribed drug. The content of the Interactions pane depends on the selected event in the Events pane and lists all known interactions of the newly prescribed drug. Similar to the KA-Tool that visualizes a SAR, drugs are selected in the application ontology pane and dragged onto the Events or Interactions pane. Every drug that is linked to an event by dragging it to the Interactions pane creates a *Has_Interactions* relation in the application ontology between the event and the dragged drug (and also the other way around, as this relation is bilateral)*

4.2.3 Executing Event-Based Modular Tasks

Similar to the previous example, PSMs that solve EBMTs are executed by means of the procedures attached to each primitive. In the CritICIS system, for example, whenever an ICU physician prescribes a new drug for a given patient, the CPR system activates CritICIS with a *Prescribe_New_Drug* event. The CPR system also supplies additional parameters such as the patient's ID and the name of the started drug. Among other tasks, this event causes the execution of the refined drug interaction PSM, which retrieves from the domain ontology all known drugs that have an interaction relation with the started drug and queries the CPR to determine whether one of them is present. Whenever this is the case, the system performs one or more actions. Similar to the reminder shown in figure 6, CritICIS reports violated constraints by means of pop-up windows [30].

4.3 A complex guideline for the treatment of hypertension

4.3.1 Representing complex temporal guidelines

Usually complex guidelines include various scenarios and temporal and branching logic. In order to represent these complex guidelines, the method ontology has been extended with new primitives, inspired by the recent EON protocol model [7]. This model defines guidelines by means of a number of concepts, such as scenarios, decisions, actions and activities. Figure 10 shows a section of the method ontology extended with primitives that represent these concepts.

<pre> Guideline_EntityGuidelinePSMPrimitiveControl_PrimitiveDecisionBoolean_CriterionK_Of_N_CriteriaBranchingSynchronizationActionActivityStart_ActivityChange_ActivityEnd_Activity Activity_EntityMonitoringTreatment </pre>	<pre> Start_Activity Name -> string Author -> string Explanation -> string Current_Activity -> Activity_Entity Starting_Value -> symbol (values: minimum, Default, maximum) Treatment Activity_Class -> Domain_Entity Activity_Attributes -> * Attribute Activity_Start -> Time_Annotation Activity_End -> Time_Annotation </pre>
---	---

Figure 10: A section of the extended ontology, developed to model guidelines that contain complex branching and temporal logic

The `start_Activity` class models an action that starts a new activity (e.g., start a new treatment). The `starting_value` attribute of this class specifies initial values of the new activity. The `Treatment` class is an example of an activity that models a treatment such as prescribing a new drug. The `Time_Annotation` class models temporal points in terms of a reference point and additional (optional) attributes to represent uncertainty in time. This class is used by the `Activity_Start` and `Activity_End` attributes of the `Activity_Entity` class that specify the start and endpoints of an activity. Other attributes of this class are the `Activity_Class` and the `Activity_Attributes` attributes that refer to the domain entities, specified in the activity (e.g., the dose of a drug).

4.3.2 Authoring complex temporal guidelines

An example of a complex temporal guideline is a guideline for the treatment of hypertension that was developed in the Medical Guideline Technology (MGT) project [33]. As this guideline (translated from a paper version [34]) does not rely on a single PSM to solve its task, the design-time interpreter uses the

standard flowchart to represent the guideline's control structure. Figure 11 shows the corresponding KA-tool, representing the hypertension guideline. It contains a number of subguidelines. Some of these guidelines are defined in terms of primitives (e.g., primitives used to determine the most favorable initial drug), but there are also subguidelines that are defined by means of a PSM (e.g. a refined selection task to determine drug interactions and a refined version of Propose-and-Revise to substitute non-effective drugs for other ones).

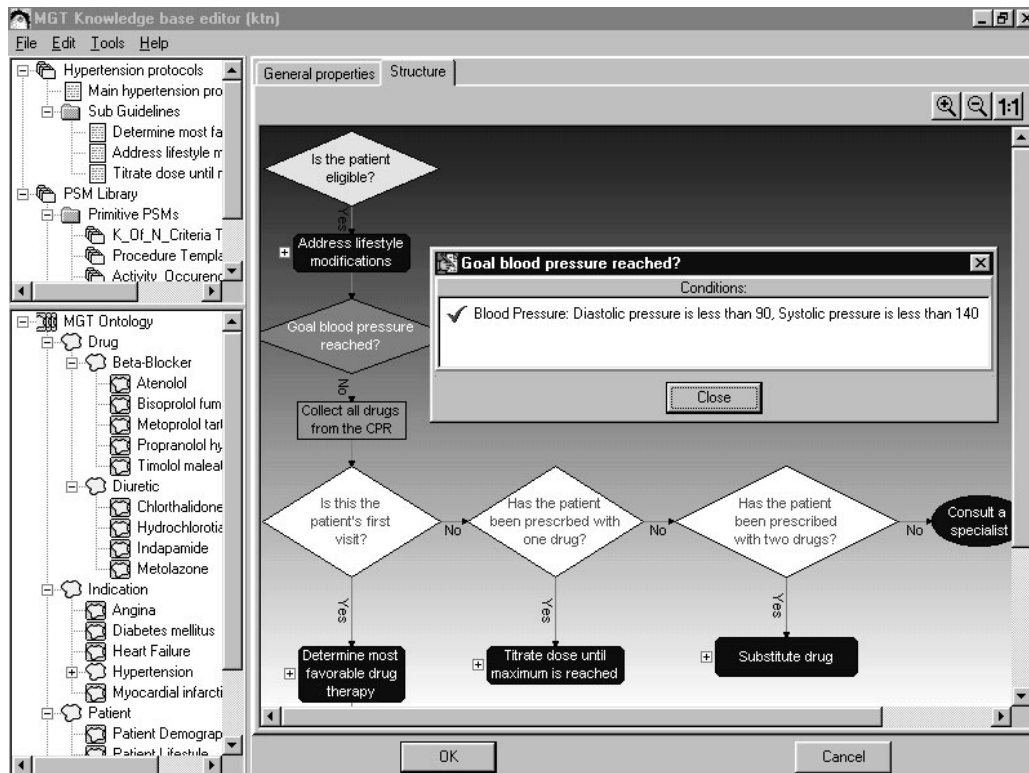


Figure 11: The hypertension guideline, shown in the KA-Tool. The 'Goal Blood pressure reached?' step is selected, which is an instance of the *k_of_N_Criteria* primitive. As a result, a user interface is created that enables guideline authors to define criteria in terms of domain ontology concepts

As the guideline structure is no longer hidden from the guideline author, knowledge acquisition becomes a two-phase process. The first phase consists of describing the guideline's structure in terms of primitives and subguidelines (flow control). In this phase, all primitives and subguidelines are treated as black boxes with no domain-specific content. In order to build the guideline by means of these boxes, the method manager containing all available PSMs and primitives is also loaded in the KA-Tool and shown in the upper left pane (figure 11). The second phase consists of specifying domain knowledge that is required by the various primitives and PSMs such as known

interactions or compelling indications. Note that all SAR and EBMT guidelines, described in the previous examples, also consist of two layers. However, as the corresponding control structure is already defined, the first phase is executed automatically.

4.3.3 Executing complex temporal guidelines

Again, this guideline is executed by means of the execution-time interpreter, which processes the symbol-level knowledge base and executes attached implementation procedures. However, as the hypertension guideline is incorporated in a system that generates web-based advice, the output ontology is extended with primitives that are able to generate HTML-pages.

5 Results

The methodology and tools described in this paper were used to develop a number of guidelines and decision support systems. The CritICIS system contains guidelines that are based on the Situation-Action Rules (SARS) as well as on PSMs that solve Event-Based Modular Tasks (EBMTs) such as selection tasks and monitoring tasks. The CritICIS system has undergone a validation, in which guidelines were tested on a large patient data set of previously admitted ICU patients. For this purpose, a development environment was designed that enabled guideline authors (intensivists) to develop, validate and update new guidelines as well as to customize existing guidelines that were used in similar situations. Among other things, the validation showed that 88% of all issued reminders, issued during the last two years (based on the existing patient data set) were classified as correct. A more detailed description of the development environment including the validation procedure can be found elsewhere [28]. The CritICIS system is now fully operational in the 20-bed ICU of the Catharina Hospital, Eindhoven, the Netherlands. With the help of data collected from the operational system, the SAR as well as the EBMT guidelines are currently being evaluated.

Another system that consists of SAR guidelines is the GRIF system, developed to change Family Physicians' (FP) test ordering behavior by focusing on the appropriateness of test requests. GRIF was validated by comparing comments of human experts with comments of the reminder system. The overall agreement in the final validation round was 69%, where the number of correct reactions of the reminder system was almost as high as the number of correct reactions of the human expert. Details are described elsewhere [35].

The Multidisciplinary Psychoactive Drug Selection –advisor system (M-PADS) is a decision support system, developed for selecting the most appropriate

psychoactive drug in order to treat psychiatric patients [36]. It contains guidelines that consist of a number of selection and monitoring PSMs, varying from the ones, described in the examples section to more complex ones that process more 'deep knowledge' (using a semantic network). Each PSM is modeled as a subguideline and branching and synchronization primitives are used to execute these PSMs in parallel. In contrast to the other examples that used Protégé for developing the domain ontology, the domain ontology for this system was developed with the help of the GALEN approach [37]. A first evaluation is currently ongoing.

Finally, the framework is currently also being used to develop guidelines that are based on primitives as well as PSMs, similar the to already mentioned hypertension guideline. Application domains include anesthesia (real-time weaning protocols), family practice (diabetic guidelines) and oncology (guidelines for the treatment of leukemia).

6 Discussion

6.1 *Characteristics of the ontological approach*

Over the last decade, a number of approaches for guideline representation have been proposed. These representations usually define guidelines in terms of primitives, such as actions and decisions. Although primitives are invaluable to describe non-decomposable steps in a guideline, more abstract descriptions such as PSMs facilitate the shareability and reusability of clinical guidelines. As the ontological representation is extensible and represents guidelines on the level of PSMs as well as on the level of primitives, it is scalable, flexible and expressive enough to define guidelines that differ in complexity and application domain, as illustrated by the examples.

A very important issue in the development of guidelines is the knowledge acquisition process. Graphical editors such as KA-Tools are increasingly used for acquiring guidelines. Representation formalisms such as GLIF and *Proforma* characterize a guideline by means of a limited set of primitives (e.g., decisions and actions). Therefore, the user interface of each primitive is limited to those basic primitives (e.g., a 'prescribe new drug' action is visualized in the same way as a 'send an e-mail to a practitioner' action). Being able to extend the method ontology by defining new primitives as well as being able to define specific visualization information improves the acceptance of the corresponding KA-Tool [28]. Furthermore, describing (sections of) guidelines by means of PSMs facilitates the authoring of guidelines by hiding the control structure and providing task-specific user

interfaces, in which domain experts are able to enter domain-specific knowledge depending on the guideline's task.

The examples and results have illustrated that the guideline's application domain dictates its representation. Guidelines that are more complex or domain-specific usually require a more low-level representation (e.g., a set of primitives) as these guidelines are usually too specific to be captured by PSMs. Guidelines that address more generic tasks (e.g., heuristic classification or selection tasks) are more suited to be represented by means of PSMs. However, when guideline authors become more familiar with the application domain, they may be able to recognize certain patterns, which can be translated into PSMs.

6.2 *Sharing models and guidelines among different institutions*

As mentioned earlier, an important issue is the shareability of guidelines as well as guideline representations among multiple institutions and organizations to improve the guidelines' effectiveness. The ontological guideline representation described in this paper facilitates this shareability, as earlier-developed domain and method ontologies can be reused among various application domains. Also, whenever guidelines are described in terms of PSMs, the guidelines itself are also shareable in the case that one PSM can be applied to various domains (e.g., classification or clinical trials). The two-phased process can be used to make guidelines more site-specific: the basic structure of the guideline is defined during the first phase, after which different institutions can modify domain knowledge to create a more customized guideline. However, problems may also arise when utilizing the ontological approach, of which the earlier-mentioned mapping problem is probably the most common one.

6.3 *Comparing other formalisms*

When comparing the methodology and representation described in this paper with other approaches such as the Arden Syntax, Prestige, GLIF, *Proforma*, Asbru and the recent EON model, a number of similarities as well as discrepancies are encountered. As the Arden Syntax models guidelines in terms of modular rules, this representation is not suitable for representing complex guidelines. In terms of the approach, described in this paper, the SAR model encompasses Arden Syntax guidelines.

Both GLIF and EON model flow control in a guideline by sequences of primitives, similar to the approach described in this paper. Asbru and *Proforma* use similar constructs, with some differences however. In Asbru, for example, the body of a (sub)guideline (referred to as a plan in both Asbru and

Proforma) consists only of actions. The decisions are modeled in the plan's preferences. The action sequence is modeled in the same manner as the approach in this paper. Similar to GLIF, *Proforma* models guidelines (plans) by means of a basic set of primitives, which are also graphically represented by means of a flowchart. However, sets of primitives are not modeled as sequences. Instead, all primitives are executed in parallel, where links between primitives are temporal constraints. *Prestige* does not model guidelines in terms of sequences of primitives, but as sequences of primitive states (e.g., an action is in a rejected state or completed state), using transition networks to describe guideline dynamics. Although well defined, this approach may result in a more opaque guideline authoring process.

Our methodology and the EON approach share the view that non-monolithic models are necessary in order to deal with the variety and complexity of guidelines. In addition, conceptualization of the guideline domain in terms of activities and scenarios allows the construction of complex temporal guidelines, such as the hypertension guideline

None of the above-mentioned approaches use the notion of PSMs to describe stereotypical tasks that a guideline may perform (although an older version of EON was based on a single PSM, called ESPR, which was used to represent clinical trials [38]). However, EON, *Proforma* and Asbru do contain constructs for describing (sub)guidelines on higher levels of abstraction such as eligibility criteria, intentions, goals, scenarios and plans. For example, *Proforma* uses the same two-layered approach for laying out plans as our approach.

Regarding knowledge acquisition, EON and GLIF both utilize Protégé to acquire knowledge from guideline authors, whereas *Proforma* relies on custom-made graphical tools for the knowledge acquisition process. Asbru uses Protégé as well as AsbruView [39] for knowledge acquisition. The KA-Tool, developed to support the approach, described in this paper, uses plug-ins to support the definition of very specific and flexible user interfaces for guideline acquisition. This was for example not possible in the previous Windows version of Protégé [18], where the user interface was not as flexible as the one, described here. This has been recognized by the developers of Protégé, as the recent Java version supports the use of plug-ins to create flexible task-specific user interfaces. However, this version was not available during the time our framework was developed.

In summary, the representation, described in this paper merges several approaches, used in knowledge modeling to define various classes of guidelines that differ in application domain and in complexity. The use of a

suite of tools that supports all stages in guideline development and execution is often underestimated but of crucial importance. The use of PSMs as well as ontologies facilitates guideline reusability as well as shareability as we have shown by means of the examples. Furthermore, primitive- and task-specific user interfaces can drive an interactive KA-Tool to assist in the often arduous process of guideline authoring. Finally, as ontologies can make the conceptualizations behind a model explicit, the ontological representation can be used to characterize the requirements of guidelines [40]. The granularity and complexity of a guideline can be expressed in terms of number of used instances, primitives, ontologies and (refined) PSMs. Examples include 1) the GRIF knowledge base, which consists of about 2000 instances, 2 primitives, 2 ontologies and 1 strategy), 2) the M-PADS knowledge base, which consists of about 150 instances, 9 primitives, 3 ontologies and 15 PSMs and 3) the hypertension guideline, which currently consists of 180 instances, 5 ontologies and 2 PSMs.

6.4 Conclusion

As illustrated by the examples and results, our approach meets all earlier-mentioned requirements, seems expressive enough to represent various classes of guidelines and will hopefully contribute to the development of standards for computer-based clinical guidelines.

Acknowledgements

The authors wish to thank Mark Musen, Jan Bergmans and two anonymous reviewers for comments on an earlier version of this paper.

References

1. Grimshaw JM, Russel IT. Effects of Clinical Guidelines on Medical Practice: A Systematic Review of Rigorous Evaluation. *Lancet* 1993;342:1317-22.
2. Effective Health Care. Implementing Clinical Practice Guidelines: Can guidelines be used to improve clinical practice? *Effective Health Care* 1994;8:1-12.
3. Van Der Lei J, Talmon JL. Clinical Decision-Support Systems. In: Van Bommel and Musen (eds). *Handbook of medical informatics*. Houten: Bohn Stafleu Van Loghum, 1997.
4. Fridsma DB, Gennari JH, Musen MA. Making Generic Guidelines Site-Specific. *Proc AMIA* 1996;:597-601.
5. Position statements from the Invitational Workshop: Towards Representations for Sharable Guidelines. Available at <http://www.glif.org/workshop/statement.htm>.
6. Ohno-Machado L, Gennari JH, Murphy SN, Jain NL, Tu SW, Oliver DE, Pattison-Gordon E, Greenes RA, Shortliffe EH, Barnett GO. The guideline interchange format: a model for representing guidelines. *JAMIA* 1998;5(4):357-72.
7. Tu SW, Musen MA. A flexible approach to guideline modeling. *Proc AMIA Symp* 1999;:420-4.
8. Hripcsak G. Rationale for the Arden Syntax. *Comput Biomed Res* 1994;27(4):291-324.
9. Fox J, Johns N, Rahmzadeh A. Disseminating medical knowledge: the PROforma approach. *Artif Intell Med* 1998;14:157-81.

10. Shahar Y, Miksch S, Johnson P. The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines. *Artif Intell Med* 1998;14:29-51.
11. Musen MA. Dimensions of Knowledge Sharing and Reuse. *Comput Biomed Res* 1992;25:435-67.
12. Gruber TR. A translation approach to portable ontologies. *Knowledge Acquisition* 1993;5(2):199-220.
13. Chandrashekar B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* 1986;1:23-30.
14. Clancey WJ. Heuristic classification. *Artificial Intelligence* 1985;27(3):289-350.
15. Musen MA. Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. *Proc AMIA Symp* 1998;:46-52.
16. Schreiber AT, Wielinga BJ, De Hoog R, Akkermans H, van der Velde W, Anjewierden A. CommanKADS: A Comprehensive Methodology for KBS Development. *IEEE Expert* 1994;:28-37.
17. Motta E. Reusable Components for Knowledge Modelling. Case Studies in Parametric Design Problem Solving. Amsterdam: IOS Press, 1999.
18. Musen MA, Gennari JH, Eriksson H, Tu SW, Puerta AR. PROTEGE II: Computer Support For Development Of Intelligent Systems From Libraries Of Components. *Medinfo* 1995;8(1):766-70.
19. Fensel D, Benjamins VR, Motta E, Wielinga B. UPML: A Framework for knowledge system reuse. *Proceedings of the International Joint Conference on AI (IJCAI '99)* 1999.
20. Marcus S. Automated knowledge acquisition for expert system. Norwell: Kluwer Academic Publishers, 1988.
21. Eriksson H, Musen MA. Metatools for knowledge acquisition. *IEEE Software* 1993;10(3):23-9.
22. Studer R, Eriksson H, Gennari J, Tu S, Fensel D, Musen M. Ontologies and the Configuration of Problem-solving Methods. *Proceedings of the 10th Knowledge Acquisition for Knowledgebased Systems Workshop, Banff*, 1996.
23. Chen PPS. The entity-relationship approach to logical data base design. The Q.E.D. monograph series on Data base management no. 6. Wellesley, Mass : Q.E.D. Information Sciences.
24. Gennari JH, Tu SW, Rothenfluh TE, Musen MA. Mapping Domains to Methods in Support of Reuse. *International Journal of Human-Computer Studies* 1994;41:399-424.
25. De Clercq PA, Blom JA, Hasman A, Korsten HHM. Design and implementation of a framework to support the development of clinical guidelines. *Int J Med Inf* 2001;64(2-3):285-318.
26. Hripcsak G, Clayton PD, Jenders RA, Cimino JJ, Johnson SB. Design of a clinical event monitor. *Comput Biomed Res* 1996;29(3):194-221.
27. McDonald CJ, Hui SL, Smith DM, Tierney WM, Cohen SJ, Weinberger M. Reminders to physicians from an introspective computer medical record. A two-year randomized trial. *Ann Intern Med* 1984;100:130-8.
28. De Clercq PA, Blom JA, Hasman A, Korsten HHM. A strategy for development of practice guidelines for the ICU using automated knowledge acquisition techniques. *Int J Clin Monit Comput* 1999;15:109-17.
29. Van der Lei J, Musen MA. A model for critiquing based on automated medical records. *Comput Biomed Res* 1991;24:344-78.
30. De Clercq PA, Blom JA, Hasman A, Korsten HHM. Gaston: An architecture for the acquisition and execution of clinical guideline-application tasks. *Med Inform Internet Med* 2000;25(4):247-63.

31. Miller PL. Expert Critiquing Systems, Practice-Based Medical Consultation by Computer. New York: Springer-Verlag, 1986.
32. Chandrasekaran B, Johnson TR, Smith JW. Task-Structure Analysis for Knowledge Modeling. *Communications of the ACM* 1992;35(9):124-37.
33. The Medical Guideline Technology project. INCO-COPERNICUS Project IC15 CT 98-0315. Homepage available at <http://frost.open.ac.uk/mgt/>.
34. National High Blood Pressure Education Program. The Sixth Report of the Joint National Committee on Detection, Evaluation, and Treatment of High Blood Pressure. Washington: NIH; 1998.
35. Bindels R, de Clercq PA, Winkens RAG, Hasman A. A test ordering system with automated reminders for primary care based on practice guidelines. *Int J Med Inf* 2000;58-59(1):219-33.
36. Van Hyfte DMH, De Clercq PA, Tjandra-Maga TB, Zitman FG, De Vries Robbé PF. Modelling the psychoactive drug selection application domain at the knowledge level. *Proc Belgium-Netherlands Conf on Artificial Intelligence* 1999;:187-8.
37. Rector A, Solomon W, Nowlan W, Rush T, Zanstra P, Claassen W. A Terminology Server for medical language and medical information systems. *Meth Inform Med* 1995;34:147-57.
38. Musen MA, Tu SW, Das A, Shahar Y. EON: A Component-Based Approach to Automation of Protocol-Directed Therapy. *JAMIA* 1996;3:367-88.
39. Miksch S, Kosara R, Shahar Y, Johnson PD. AsbruView: Visualization of Time-Oriented, Skeletal Plans. *The Fourth International Conference on Artificial Intelligence Planning Systems* 1998, Carnegie-Mellon University, Pittsburgh, PA, 11-8.
40. Pisanelli DM, Gangemi A, Steve G. Towards a Standard for Guideline Representation: an Ontological Approach. *JAMIA* 1999;6(4):906-10.

CHAPTER 4

DESIGN AND IMPLEMENTATION OF A FRAMEWORK TO SUPPORT THE DEVELOPMENT OF CLINICAL GUIDELINES

Published in:
The International Journal of Medical Informatics 2001;64(2-3):285-318

Paul A. de Clercq
Arie Hasman
Johannes A. Blom
Hendrikus H.M. Korsten

1 Introduction

Recently, studies have shown the benefits of using clinical guidelines in the practice of medicine [1]. Utilizing guidelines such as standard care plans, critical pathways and protocols in various clinical settings may lead to a reduction of practice variability and patient care costs, while improving patient care [2]. Use of decision support systems that incorporate such guidelines offer promising possibilities for guideline implementation. According to the Institute of Medicine (IOM), these decision support systems are in fact crucial elements in long-term strategies for promoting the use of guidelines [3].

There have been numerous efforts to develop systems that support guideline-based care in an automated fashion, covering a wide range of clinical settings and tasks [4]. Despite these efforts, only a few systems progressed beyond the prototype stage and the research laboratory. Building systems that are both effective in supporting clinicians and accepted by them has proven to be a difficult task. Yet, of the few systems that were evaluated by a controlled trial, the majority showed impact [5]. This paper describes and discusses Gaston: a framework that facilitates all stages in the guideline development process, ranging from the definition of models that represent guidelines to the implementation of run-time systems that provide decision support, using the guidelines that were developed during the previous stages. The Gaston framework consists of 1) a newly developed guideline representation formalism that uses the concepts of primitives, Problem-Solving Methods (PSMs) and ontologies to represent guidelines of various complexity and granularity and different application domains, 2) a guideline authoring environment that enables guideline authors to define guidelines, based on the newly developed guideline representation formalism, and 3) a guideline execution environment that translates defined guidelines into a more efficient symbol-level representation, which can be read in and processed by an execution-time engine.

Section 2 of this paper defines a number of design criteria that were formulated regarding the aspects of guideline representation, guideline authoring and guideline execution and also describes the methods and materials that were used to develop the Gaston framework, according to the formulated design criteria. Section 3 describes the Gaston framework by example in terms of the four stages that were identified in the guideline development process, along with the tools that were developed to support each stage. Section 4 presents a number of guidelines and decision support systems that were developed by means of the Gaston framework. Finally,

section 5 discusses various aspects of guideline-based decision support in general and the Gaston framework in particular.

2 Materials and Methods

2.1 *Design criteria*

2.1.1 Guideline representation formalisms

A very important aspect that has to be reckoned with when designing guideline-based decision support systems is the issue of guideline representation. During the last decade, various guideline representation languages have been developed, each with their own formalisms and specifications. By analyzing a number of these, criteria were formulated for a guideline representation language [6-8]. These requirements include the possibility to represent temporal logic, branching and sequencing, patient data elements, (eligibility) criteria, actions and decompositions of actions.

Depending on the guideline's application domain, the guideline representation formalism must also be able to represent in a consistent manner various kinds of guidelines that may differ considerably in complexity. Examples are relatively simple guidelines that model independent modular rules (e.g., alerts in reminder systems) [9, 10], but also complex guidelines that use notions such as temporal abstraction and scheduling in order to model complex treatment plans [11-13].

Another important issue is the shareability of both the guideline representation and the guidelines among multiple institutions and organizations to improve the guidelines' effectiveness [14]. Therefore, the guideline representation formalism must facilitate the reuse and sharing of similar guidelines among various domains.

2.1.2 Guideline acquisition

An important issue in the development of guidelines is the knowledge acquisition process. The traditional knowledge elicitation methodology that required an intense cooperation between knowledge engineer and domain expert created a severe bottleneck as the two experts had to reach a common understanding before progress could be made [15]. As a response to this problem, Knowledge Acquisition Tools (KA-Tools) are increasingly used to acquire knowledge directly from a domain expert. These tools may facilitate the knowledge acquisition process by helping domain experts formulate and structure domain knowledge for use in knowledge based systems [16]. Since the use of knowledge acquisition tools in practice is very limited because of

their inability to assist domain experts to enter knowledge (e.g., guidelines) that is relevant to the specific guideline-application task, a criterion is that (the user interface of) a knowledge acquisition tool facilitates the entry of guidelines, specific to the target guideline-application domain.

2.1.3 Guideline execution

A guideline execution engine must be able to process the guideline representation format directly. As execution speed is a very important requirement at this stage, the representation language must be described in an efficient format to be interpretable by an execution engine in real-time [17, 18]. Finally, guideline execution engines must also be able to exchange information with the outside world, such as external information systems.

2.2 *Applied materials and methods*

2.2.1 Guideline representation formalisms

The core of the Gaston framework consists of a newly developed guideline representation formalism. In order for this guideline representation formalism to satisfy all above-mentioned requirements, a number of approaches that are known in the area of knowledge representation are combined. These approaches are based on the concepts of primitives, Problem-Solving Methods (PSMs) and ontologies.

During the last decade, a common approach has been to model the control structure of guidelines in terms of explicit primitives, which characterize stereotypical tasks a guideline may perform, such as checking eligibility criteria, actions and decisions [6-8]. At the same time, various research groups developed representation formalisms that did not specifically focus on guideline-based care, but concentrated more on the abstract behavior of decision support systems in general [19-21]. These methodologies express the notion that the behavior of decision support systems can be described by means of two independent classes of reusable components: 1) domain ontologies that characterize concepts and relationships in an application area, providing a domain of discourse and 2) domain-independent algorithms that describe abstract methods for achieving solutions to common tasks, such as constraint satisfaction, classification, planning and critiquing. Although known under various names, this paper refers to these algorithms as Problem-Solving Methods (PSMs) [22]. PSMs such as heuristic classification [23] are role-limiting by nature, meaning that the PSMs impose specific problem-solving roles on domain knowledge. These problem-solving roles are referred to as knowledge roles, which give an abstract description of the function particular domain knowledge has to play. When refining a PSM to a certain

domain, the knowledge roles are mapped onto domain knowledge. PSMs can be reused to solve similar problems in different application domains by using different domain ontologies. PSMs are decomposable into subtasks, which can be executed by submethods. When no longer decomposable, a submethod is referred to as a primitive PSM or mechanism.

When comparing the primitive-based approach vs. the PSM-based approach, each approach has its strong points as well as shortcomings. On the one hand, separating domain-specific knowledge and PSMs may increase the reusability as well as shareability of developed guidelines, as earlier developed domain ontologies and PSMs (that are already tested and proved) can (partly) be reused in other developments. This is more difficult in the primitive-based approach where domain and procedural knowledge are often intertwined. Also, as PSMs usually describe knowledge on an abstract level, the global structure can be explicitly stated, in contrast with primitive-based approaches that often represent guidelines at a single level of detail. However, most of the guideline representations that are used in clinical practice today do not use the notions of PSMs and domain-specific knowledge, because of several reasons, the main reason being that certain types of protocols used in daily practice can not easily be expressed as structured, reusable stereotypical tasks. This is especially true for many clinical guidelines, which often address specific clinical problems. A more detailed discussion on the subject of primitives vs. PSMs is given elsewhere [24].

The guideline representation formalism developed in the Gaston framework combines the concepts of primitives and PSMs to represent guidelines in terms of 1) primitives to construct the guideline's control structure explicitly and 2) PSMs to model guidelines that perform stereotypical tasks. Also, the formalism supports the use of subguidelines in order to solve multiple tasks.

The representation uses ontologies as an underlying mechanism to represent guidelines in terms of PSMs and primitives in a consistent way. Two types of ontologies are defined: domain ontologies and method ontologies. As mentioned earlier, domain ontologies model domain-specific knowledge in terms of entities, attributes and relations [25]. Method ontologies [26] model concepts such as primitives, PSMs and guidelines similarly. Primitives are used 1) to describe single guideline steps, and 2) to describe the internal structure of PSMs. The guideline representation formalism is non-monolithic, meaning that ontologies can be extended to capture new guideline characteristics. The ontologies were defined by means of the ontology editor that was developed in the Protégé framework: a methodology and a set of tools to develop knowledge based systems [21]. With Protégé, knowledge

engineers are able to define concepts and relations to create domain as well as method ontologies. Protégé uses a slightly adapted version of Open Knowledge Base Connectivity (OKBC) [27, 28] as the underlying knowledge model and stores the ontologies using various representation languages such as CLIPS [29] and the Resource Description Format (RDF) [30].

2.2.2 Guideline acquisition

In order to create a flexible KA-Tool that can be used in various application domains, the KA-Tool has been implemented as a kernel that loads a number of plugins, in which each plugin defines a different functionality. The KA-Tool loads the necessary ontologies (in CLIPS format) to visually represent guidelines in terms of primitives and PSMs. The current version runs under the Microsoft Windows environment, in which all plugins are implemented as Dynamic-Link Libraries (DLLs). As DLLs are independent of the used programming language, different languages can be used to develop plugins.

2.2.3 Guideline execution

Similar to the implementation of the KA-Tool, the guideline execution engine is also implemented as a kernel that loads a number of plugins (DLLs). However, in contrast with the requirements of the KA-Tool, the execution engine must be optimized to meet the requirements of speed and compactness in order to execute guideline in real-time. Therefore, the engine was based on the format, developed in the SIMPLEXYS project that aimed at the development of real-time decision-support systems [31].

3 The Gaston framework

3.1 Overview

By using the techniques, described in the previous section, the Gaston guideline development methodology and supporting framework were developed. The framework consists of a suite of tools that support the various stages in guideline development. Figure 1 shows the process view of the framework.

The process consists of four stages, each of which is reusable in other guideline development processes:

- Develop, derive or reuse application-specific domain and method ontologies.
- Develop or reuse libraries of PSMs.
- Develop guidelines in terms of PSMs and primitives through a Knowledge Acquisition Tool (KA-Tool).

- Automatically translate these guidelines into a more efficient symbol-level representation, which can be read in and processed by an execution-time interpreter.

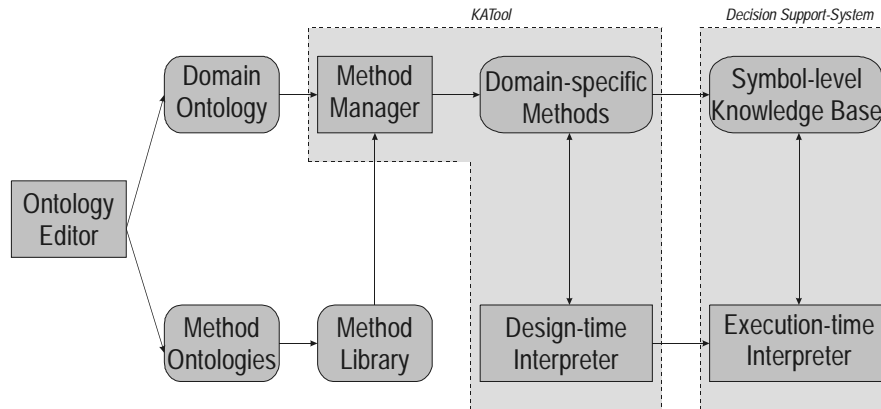


Figure 1: Process view of the Gaston framework. Rounded rectangles represent models (e.g., ontologies), straight rectangles represent modules (e.g., programs)

Several tools support each stage. Protégé was used to facilitate the development of domain and method ontologies. A separate KA-Tool was developed (not automatically generated by Protégé), consisting of a kernel of which the functionality is extended by loading additional plug-ins. Finally, a run-time environment was developed that executes guidelines, acquired through the KA-Tool.

The first stage involves developing or reusing domain and method ontologies by defining hierarchies of entities, attributes and relations. Depending on the requirements of the method ontology, existing domain ontologies can be extended with new attributes or relations.

A method library is a collection of PSMs and primitives created in the second stage that can be used in the third stage for the definition of guidelines to solve certain tasks. As PSMs usually describe rather abstract problem-solving behavior, a knowledge engineer uses the method manager to design application-specific PSMs by 1) refining the PSM by providing specific values of certain attributes of the PSM (e.g., the specific contents of a message that must be supplied to a user) and 2) specifying the PSM's knowledge roles by mapping concepts from the domain ontology onto corresponding concepts of the PSM. Three types of knowledge roles are defined: Input roles, Output roles and Intermediate roles. Input and Output roles refer to knowledge roles that are also used by other PSMs (e.g. another PSM that uses this PSM to solve a certain subtask), whereas Intermediate roles refer to knowledge roles that are used only by the PSM internally.

The third stage utilizes the KA-Tool, containing a design-time interpreter that loads the required primitives and domain-specific methods and creates a user interface that enables guideline authors to develop guidelines. Primitives are displayed in a flowchart representing the guideline, whereas PSMs are visualized by utilizing the visualization information of the PSM itself. The latter enables guideline authors to enter domain information regarding the corresponding task without the need to know the internal control structure of the PSM.

When instructed, the KA-Tool combines the control structure of each guideline and PSM and creates a network consisting solely of primitives. In this way, a symbol-level knowledge base is created that can be processed by an execution-time engine that executes the implementation modules, attached to each primitive.

The remaining part of this section describes the various stages and associated tools in more detail using as example a guideline-based decision support system that was developed in the area of hypertension [32]. This system was developed within the framework of the Medical Guideline Technology (MGT) project [33].

3.2 Stage 1: defining domain and method ontologies

3.2.1 Utilizing the OKBC model

The first stage involves developing or reusing domain and method ontologies by defining hierarchies of entities, attributes and relations. As mentioned earlier, a slightly modified version of the OKBC knowledge model is used to represent domain and method ontologies. OKBC is a frame-based language and is used in the Gaston framework to define an ontology in terms of classes, attributes and facets. Classes are concepts in the domain of discourse. Each class is explicitly described by means of a number of attributes, referred to as slots in Protégé. Facets describe the properties of an attribute, in the same way as attributes describe a class. For example, the concept of a drug can be represented by a `Drug` class that contains various attributes such as a `Dosage` attribute that defines the drug's dosage, a `Dosage_Unit` attribute that defines possible units of a dosage and a `Prescription_Date` attribute that defines the prescription date. Each attribute is defined by means of a number of facets, such as a `type` facet that holds the attribute's type (e.g., 'string' or 'number') or an `allowed-values` facet that holds the allowed values of an attribute. The `type` facet of the `Dosage` attribute, for example, holds the value 'number', whereas the `type` facet of the `Dosage_Unit` attribute holds the value 'symbol' to indicate that there only exist

a limited number of possible dosage units. In this case, the `allowed-values` facet holds the possible units such as mmol/l or mg/pill. Finally, the `type` facet of the `Prescription_Date` attribute holds the value 'date'. Furthermore, the knowledge base that contains the actual guidelines and PSMs are described by means of class instances in which the attributes have specific values. An instance of the `Drug` class, for example, may refer to an actual prescription of a drug, in which the attributes of the instance contain values related to the prescribed drug.

Besides 'traditional' classes, the OKBC knowledge model in Protégé also defines the concept of metaclasses. A metaclass is a class whose instances are themselves classes. In OKBC, every frame (e.g., classes, attributes and facets) is an instance of a class. Since classes are also frames, every class is an instance of another class. Therefore, every class has a dual identity: it is a subclass of a class in the class hierarchy—its superclass,—and it is an instance of another class—its metaclass. Therefore, a metaclass acts as a template for classes that are its instances. A metaclass describes how a class that instantiates this template will look: namely, which attributes it will have and what the attribute's facets are. Similarly, a 'traditional' class describes what instances of that class look like: which attributes the instances will have and what are the facets of these attributes. For example, in Protégé, every class by default is an instance of the metaclass `:STANDARD-CLASS`. This metaclass contains a number of attributes (referred to as slots in Protégé) that explicitly defines a class. Examples are the `:NAME` attribute that contains the class' name, the `:DIRECT-SUPERCLASSES` attribute that contains the class' parents and the `:DIRECT-TEMPLATE-SLOTS` attribute that contains the attributes of the class. However, if an ontology requires a different metaclass, such a metaclass can be defined. Once it is defined, instances of it (which are classes) can be acquired. A more detailed discussion on metaclasses and their use in Protégé can be found elsewhere [28]. Examples of the use of metaclasses are found in the next sections.

3.2.2 Domain ontologies

Domain ontologies model domain-specific knowledge in terms of entities, attributes and relations. Because ontologies are models, there are multiple ways of defining ontologies. A domain ontology generally defines the global concepts that are relevant to the application domain in terms of classes (e.g., drugs) and attributes (e.g., dosage). Instances of a drug are not regarded as part of the ontology but are acquired through knowledge acquisition [34]. However, inheritance is often used to define additional classes that characterize more specific concepts (e.g., an antibiotic is a member of the

class of drugs). For example, figure 2 shows an example of a domain ontology, used in the development of the hypertension guidelines.

Domain_Entity	
.....Treatment	
.....Drug	Drug
.....Antibiotic	Dosage -> Number
.....Anti_Depressive	Dosage_Unit -> Symbol (allowed-values: mmol/l,
.....Fluxetine_Hydrochloride	mg/kg, mg/pill)
.....Circulation	Prescription_Date -> Date
.....Acetylcysteine	Has_Interactions ->* Interaction_Relation
.....Beta-Blocker	
.....Cardio-surgical	
.....Disease	Interaction_Relation
.....Indication	Target -> Drug
.....Laboratory_Test	Significance -> symbol (allowed-values: normal,
Relation_Entity	severe, contraindicated)
.....Interaction_Relation	

*Figure 2: Part of a domain ontology. The left column shows a class hierarchy of entities that describe a particular domain. The right column presents a more detailed view of two classes and their attributes (not all attributes are shown). Each attribute has a type such as integer, string or symbol and is by default inherited by the subclasses (attributes that are inherited from other classes are shown in italic). In this example, the *Interaction_Relation* class models an interaction between two drugs. By means of the *Interaction_Relation*'s *Significance* attribute, each interaction can be characterized as normal, severe or contraindicated. Attributes may refer to one instance (e.g., each drug has only one dosage) or multiple instances (e.g., each drug may have various known interactions). If an attribute refers to multiple instances, an asterisk follows the arrow*

This particular domain ontology uses inheritance to define entities, relations and attributes such as drugs, diseases and treatments. For example, an entity that represents a disease has attributes that hold the name and date of occurrence of the disease, whereas an entity representing a drug has additional attributes that store the drug's dosage and prescription date. Furthermore, this domain ontology also contains relationships between entities such as the *Has_Interactions* relation, which models an interaction relation between two drugs.

Furthermore, metaclasses are used to define class-specific properties. For example, a drug may also be known by its brand name (e.g., Fluoxetine Hydrochloride is better known by its brand name Prozac). As this property remains the same for each drug instance (contrary to, for example, the dosage of a drug that may differ for various instances), it is defined as part of the *Drug* class definition. Therefore, in this example, a *:STANDARD-DRUG-CLASS* metaclass was derived from the *:STANDARD-CLASS* metaclass, which -besides the standard attributes of the *:STANDARD-CLASS* metaclass- also defines an additional *:BRAND-NAME* attribute of type string. As a result, it is possible to

include a brand name in the definition of a `Drug` class and subclasses (which are defined as instances of the `:STANDARD-DRUG-CLASS` metaclass). Similar to metaclasses, metaslots are used to add new facets to an attribute definition. For example, the `Dosage` attribute in the domain ontology is an instance of the `:STANDARD-DATASOURCE-SLOT` metaslot, which is derived from the default `:STANDARD-SLOT` metaslot. Again, besides the standard facets that are defined in the `:STANDARD-SLOT` metaslot such as the name and type facets, the `:STANDARD-DATASOURCE-SLOT` metaslot defines the `datasource` facet, which refers to an interface to an external database such as a Electronic Patient Record (EPR). This interface can be used by a guideline execution engine to get the required data. As the `datasource` facet may refer to multiple interfaces, a single domain ontology can be used in combination with various databases.

As mentioned earlier, instances (of non-metaclasses) are usually not included in an ontology, but are acquired through knowledge acquisition. It depends on the target domain and the developer of the domain ontology what is regarded as a class and what is regarded as an instance. For example, in the domain ontology that is shown in figure 2, all known drugs are predefined by means of classes. Other ontologies only define a single `Drug` class [34]. Individual drugs are then acquired as instances of the `Drug` class through knowledge acquisition. Another example is the `Has_Interactions` attribute of the `Drug` class, also shown in figure 2. As this attribute models an interaction between two drugs, it is also possible to define the actual interaction in the domain ontology (e.g., by means of a `:HAS-INTERACTION` attribute of the metaclass `:STANDARD-DRUG-CLASS`). However, in the MGT project, interactions were acquired through knowledge acquisition and are therefore not part of the domain ontology.

Ontologies are entered through Protégé, which stores ontologies in a slightly modified version of the CLIPS format. Figure 3 shows a part of the domain ontology of figure 2 in terms of class and instance definitions.

In figure 3, attributes are referred to as a single-slot attribute when it holds a single value (e.g., each drug only has a single dosage), whereas a multislot attribute may contain multiple values (e.g., each drug can have multiple interactions). Whenever the `type` facet holds the value 'instance', this attribute points to a class instance. The `:THING` class (superclass of the `Relation_Entity` class) is the root of an ontology. A more detailed description of the CLIPS format can be found elsewhere [29].

The upper section of figure 3 shows the definition of the `:STANDARD-DRUG-CLASS` metaclass and the `:STANDARD-DATASOURCE-SLOT` metaslot along with a

number of their attributes (not all attributes are shown here). This section also shows the definition of the `Drug` class, the `Antibiotic` class and the `Interaction_Relation` class. The `Interaction_Relation` class inherits the `Target` attribute from its parent, the `Relation_Entity` class. However, as the `Interaction_Relation` class specifically represents drug interactions, it overrides the `allowed-classes` facet by setting its value to the `Drug` class.

```

;+ *** Class Definitions ***
(defclass :STANDARD-DRUG-CLASS
  (is-a :STANDARD-CLASS)
  (multislot :BRAND-NAME
    (type string)))

(defclass :STANDARD-DATASOURCE-SLOT
  (is-a :STANDARD-SLOT)
  (single-slot datasource)
  (type instance)
  (allowed-classes DataSource_Definition)))

(defclass Drug
  (is-a Domain_Entity)
  (single-slot Dosage
    (type float))
  (single-slot Dosage_Unit
    (type symbol)
    (allowed-values mmol/l mg/kg mg/pill))
  (single-slot Prescription_Date
    (type date))
  (multislot Has_Interactions
    (type instance)
    (allowed-classes Interaction_Relation)))

(defclass Anti_Depressive
  (is-a Drug))

(defclass Fluxetine_Hydrochloride
  (is-a Anti_Depressive))

(defclass Relation_Entity
  (is-a :THING)
  (single-slot Target
    (allowed-classes Domain_Entity)))

(defclass Interaction_Relation
  (is-a Relation_Entity)
  (single-slot Target
    (allowed-classes Drug))
  (single-slot Significance
    (type symbol)
    (allowed-values normal severe contraindicated)))

;+ *** Instance definitions***
([Fluxetine_Hydrochloride] of :STANDARD-DRUG-CLASS
  (:BRAND-NAME "Prozac"))

([Dosage] of :STANDARD-DATASOURCE-SLOT
  (datasource DataSource_Definition_1))

```

Figure 3: Part of a domain ontology, stored in a modified CLIPS format

As classes are also instances of metaclasses, a class is defined by setting the attribute values of the class' metaclass [35]. The lower section shows the `Fluxetine_Hydrochloride` class as an instance of the `:STANDARD-DRUG-CLASS` metaclass. Similarly, it also shows the `Dosage` attribute as an instance of the

:STANDARD-DATASOURCE-SLOT metaslot. The `Datasource_Definition_1` instance refers to an external interface definition (e.g., for communication with an EPR) stored elsewhere.

3.2.3 Method ontologies

As mentioned earlier, guidelines are represented by a set of primitives or by means of a PSM. Analogous to domain ontologies that describe domain-specific knowledge, method ontologies specify primitives, PSMs and guidelines in terms of entities, attributes and relations. A core method ontology was developed [24] which contains classes that define primitives, PSMs, guidelines and related concepts such as knowledge roles. Figure 4 shows a part of the core method ontology that was developed to model various categories of guidelines.

	<pre> Primitive Name -> string Caption -> string Author -> string Explanation -> string </pre>
<pre> Guideline_EntityGuidelinePSMPrimitiveControl_PrimitiveDecisionBoolean_CriterionK_Of_N_CriteriaBranchingSynchronizationAction Refiner Knowledge_RoleInput_RoleOutput_RoleIntermediate_Role DefinitionProcedure_DefinitionVisualization_DefinitionDescription_Definition Control_Structure </pre>	<pre> PSM Name -> string Caption -> string Author -> string Explanation -> string Goal -> K_Of_N_Criteria Visualization -> Complex_Visualization_Definition Control -> Control_Structure Description -> Description_Definition Refiners ->* Refiner Mappings ->* Knowledge_Role </pre>
	<pre> Guideline Name -> string Caption -> string Author -> string Explanation -> string Goal -> K_Of_N_Criteria Visualization -> Complex_Visualization_Definition Control -> Control_Structure Task_Description -> string Validation -> symbol (allowed-values: test, production) Target_Users -> string Eligibility_Criteria -> K_Of_N_Criteria Abort_Criteria -> K_Of_N_Criteria </pre>

Figure 4: A section of the core method ontology that describes the guideline model.

The left column shows a hierarchy of classes that represent primitives, PSMs, guidelines and related concepts. It also presents a number of primitives that are used to describe single guideline steps such as decisions and actions. The right column shows the three main classes in detail (again, not all attributes are shown)

The `Primitive` class defines a primitive, whereas the `PSM` class defines Problem-Solving Methods. Finally, the `Guideline` class models a (sub)guideline. The `Refiner`, `Knowledge_Role`, `Definition` and `Control_Structure` classes are auxiliary classes. Depending on the

requirements of the guideline application domain, the core ontology can be extended. For example, in order to be able to define more specific actions (e.g., administer a drug), the core ontology is extended with primitives that represent these actions. This approach makes it possible to manage various categories of guidelines that differ in variability and complexity. Similar to domain ontologies, method ontologies are entered in Protégé and stored in the CLIPS format.

3.2.4 Specifying primitives

In the guideline model, primitives represent both non-decomposable parts in a guideline (e.g. decisions and actions) similar to earlier-mentioned representations, and non-decomposable parts in PSMs. These primitives are based on version 2.0 of GLIF [8]. This specification defines the following types of primitives that are commonly used to describe guidelines: 1) Action primitives that specify clinical actions (e.g., administer a drug), 2) Decision primitives that model decision points in a guideline (e.g., if this patient suffers from hypertension then perform an action), 3) Branching primitives that direct the guideline flow to multiple (parallel) paths and 4) Synchronization primitives that converge paths that previously diverged because of a Branching primitive.

As shown in figure 4, the pragmatics of a primitive is defined in the method ontology by a number of attributes such as the `Name`, `Caption` (which holds the primitive's title), `Author` and `Explanation` attributes. Similar to the use of metaclasses in domain ontologies, metaclasses are used in method ontologies to define class-specific properties. All classes derived from the `Primitive` class are also instances of the `:STANDARD-PRIMITIVE-CLASS`. This metaclass defines four additional attributes, named `:VISUALIZATION`, `:PROCEDURE`, `:REFINERS` and `:MAPPINGS`. Visualization information is used to define a primitive-specific user interface in a KA-Tool in terms of the primitive's parameters. The `:PROCEDURE` attribute contains execution-time information, used by the interpreter of a decision support system that incorporates the primitive. This attribute, combined with the primitive's parameters, defines a generic interface to an actual implementation procedure (executable code). The `:REFINERS` attribute specifies which (combination of) attributes of the PSM are used to further refine the primitive and the `:MAPPINGS` attribute specifies the primitive's roles and contains mappings to parameters from other primitives, to concepts from the domain ontology or to knowledge roles from a PSM.

An example of a commonly used primitive is the `k_of_N_Criteria` primitive, derived from the `Decision` primitive. This primitive is a logical statement that

directs the flow of the guideline depending on its evaluation (true or false). The statement contains a number of criteria and is evaluated as true if at least a certain number (κ) of all criteria (N) is also true. Figure 5 shows the representation of this primitive in terms of (meta)classes.

Besides the attributes, inherited from its parents, the `κ _of_ N _Criteria` class defines four additional attributes. The actual criteria are stored in the `Criteria` attribute, which refers to one or more instances of the `Criterion` class (each criterion is modeled by a single instance of the `Criterion` class). The κ attribute of the `κ _of_ N _Criteria` class defines the number of criteria that must be evaluated as true in order to evaluate the entire logical statement as true. The `satisfied` and `otherwise` attributes contain references to primitives that may follow the `κ _of_ N _Criteria` primitive, depending on the outcome of the logical statement. Figure 5 also defines the `:STANDARD-PRIMITIVE-CLASS` metaclass, of which the `κ _of_ N _Criteria` primitive is declared an instance of. The `Primitive_Visualization_Definition_1` instance (which is an instance of the `Primitive_Visualization_Definition` class, defined elsewhere) contains visualization information that is specific for the `κ _of_ N _Criteria` primitive. Similarly, the `Procedure_Definition_1` instance contains specific execution-time information. The value of the `:REFINERS` attribute is used to refine every created instance of the `κ _of_ N _Criteria` class. In this case, the value of the `:REFINERS` attribute contains references to two attributes, which indicate that each instance of the `κ _of_ N _Criteria` primitive is refined by specifying values for the `caption` and κ attributes. The one-to-one value of the `Refiners_1's Mapping_Type` attribute means that it is directly defined by filling in a value for the `caption` attribute in the refinement process. The content of the `Primitive_Visualization_Definition_1`, `Refiner_1` and `Refiner_2` instances are used during the guideline acquisition phase, whereas the contents of the `Procedure_Definition_1` instance is used during the guideline execution phase. The sections that describe the guideline acquisition (section 3.4) and guideline execution phases (section 3.5) present examples on the use of the `κ _of_ N _Criteria` primitive to acquire, represent and execute criteria in a guideline.

3.2.5 Specifying Problem-Solving Methods

PSMs model stereotypical processes that may occur in a guideline such as heuristic classification and risk-assessment. Although a PSM is partially defined by means of the same attributes as a primitive, there are also differences between them. PSMs contain a high-level description (stored in the `Description` attribute) that describes the used strategy (in the current version of the ontological model, this description is stated in an informal way). In contrast to primitives, PSMs have a control structure that describes the

internal structure of the PSM in terms of subcomponents. This structure may refer to subtasks (that are solved by other PSMs), but also to primitives. Similar to domain ontologies, instances of PSMs are not regarded as part of a method ontology. The actual control structure of a PSM (which consists of instances) is defined in the method library component. This component is explained in more detail in section 3.3, which also presents an example of a PSM and its control structure in terms of instances.

```

;+ *** Class Definitions ***
(defclass :STANDARD-PRIMITIVE-CLASS
  (is-a :STANDARD-CLASS)
  (single-slot :VISUALIZATION
    (type instance)
    (allowed-classes Primitive_Visualization_Definition))
  (single-slot :PROCEDURE
    (type instance)
    (allowed-classes Procedure_Definition))
  (multislot :REFINERS
    (type instance)
    (allowed-classes Refiner))
  (multislot :MAPPINGS
    (type instance)
    (allowed-classes Knowledge_Role)))

(defclass K_Of_N_Criteria
  (is-a Decision)
  (single-slot K
    (type integer))
  (multislot Criteria
    (type instance)
    (allowed-classes Criterion))
  (single-slot Satisfied
    (type instance)
    (allowed-classes Primitive))
  (single-slot Otherwise
    (type instance)
    (allowed-classes Primitive)))

(defclass Criterion
  (is-a :THING)
  (single-slot Target
    (type instance)
    (allowed-classes Domain_Entity))
  (single-slot Next_Criterion
    (type instance)
    (allowed-classes Criterion))
  (multislot Relation_Operators
    (type string)))

;+ *** Instance definitions***
([K_Of_N_Criteria] of :STANDARD-PRIMITIVE-CLASS
  (:VISUALIZATION Primitive_Visualization_Definition_1)
  (:PROCEDURE Procedure_Definition_1)
  (:REFINERS [Refiner_1], [Refiner_2]))

([Refiner_1] of Intermediate_Role
  (Mapping_Type one-to-one)
  (Target [Caption]))

([Refiner_2] of Intermediate_Role
  (Mapping_Type one-to-one)
  (Target [K]))

```

Figure 5: The $\kappa_of_N_criteria$ primitive and auxiliary classes, represented in CLIPS

Similar to primitives, PSMs also contain visualization information that defines a specific user interface for use in a KA-Tool. However, the visualization information of the PSM differs from the visualization information of a primitive: since a PSM has access to all the visualization information of the subcomponents (primitives or PSMs) in the control structure (in terms of knowledge roles), this information is used by the PSM to define a specific user interface in terms of its own knowledge roles, in combination with the knowledge roles of the subcomponents. The visualization information of a PSM is stored in the `visualization` attribute of the PSM class. An example is provided in section 3.4.

3.2.6 Specifying Guidelines

The `Guideline` class describes an entire (sub)guideline. A guideline is associated with a task it has to solve. This task can be solved explicitly by processing a set of primitives or by selecting an appropriate PSM. Similar to a PSM, a guideline contains a control structure that describes the internal structure of the guideline in terms of subcomponents. In contrast to the control structure of a PSM, however, the control structure of a guideline does not support subtask decomposition: it contains a set of primitives or a reference to a single PSM. In case of the PSM, the rationale behind this limitation is that each guideline must solve a task, which is executed by a single PSM (although the PSM can use subtask decomposition to solve the task). Guidelines can be combined however to form a ‘superguideline’.

Similar to a PSM, a guideline also contains visualization information to represent its control structure. From this visualization information, a flowchart will be created in case the control structure consists of a number of elements. In case the guideline consists of a single PSM, the visualization of the PSM will be used.

Furthermore, the `Guideline` class defines several guideline-specific attributes such as a `Task` attribute that (informally) describes the task that has to be solved, eligibility criteria that may evoke a guideline, abort criteria that may abandon it and temporal criteria (e.g., this guideline is to be executed 4 times a day). Other guideline-specific attributes are a `validation` attribute that indicates whether the guideline has been approved for routine use (production) or is still in the test phase (test), and a `Target_Users` attribute that denotes the intended users of the guideline (e.g., administrators, physicians or nurses). Finally, similar to a PSM, the `Guideline` class also contains a `Goal` attribute that formally defines the goal of the solved task.

3.3 Stage 2: developing method libraries

A method library is a collection of available PSMs that can be used by guidelines to solve certain tasks. The method library consists primarily of instances, as each PSM is represented by means of a collection of instances. As is the case with ontologies, method libraries are also defined using Protégé. This section uses the selection PSM as an example. This PSM (described in more detail elsewhere [24]), covers situations in which a physician's newly selected action or decision may not be the most appropriate one and if so, to report possible conflicting situations [11]. For example, this PSM is used by a subguideline of the hypertension guideline for the detection of drug interactions and the detection of inappropriate drug dosages.

In order to solve these tasks, the selection PSM generally executes the following steps:

- Determine all possible conflicting situations regarding the newly selected action or decision.
- Determine whether one or more of these possible conflicting situations actually occur.
- Report all found occurring conflicting situations to the user.

The first step requires access to a domain ontology to specify possible conflicting situations, whereas the second step requires access to an external data source such as an Electronic Patient Record (EPR) to obtain previously selected actions or decisions. Finally, the third step generates advice to the user. To describe the control structure of the selection and similar PSMs, the method guideline ontology was extended with new primitives [24]. This part of the ontology defines various primitives that represent operations on sets of domain entities, such as `Add_Entities_By_External`, which is used to obtain actions and decisions from an external source (e.g., EPR), `Add_Entities_By_Relation`, used to determine all possible conflicting situations, and `Logical_Operator`, used to determine the conjunction of two sets. Furthermore, the ontology also defines the `Generate_Advice_From_Entities` primitive that is used to report occurring conflicting situations and the `set` class, which is a data structure that is used to store a number of domain entities (e.g., drugs).

```

;+ *** selection PSM definition***
([PSM_1] of PSM
  (Name "Selection PSM")
  (Description [Description_1])
  (Goal [K_Of_N_Criteria_1])
  (Refiners [Refiner_1], [Refiner_2])
  (Mappings [Intermediate_Role_1], [Intermediate_Role_2],
    [Intermediate_Role_3])
  (Visualization [Complex_Visualization_Information_1])
  (Control [Control_Structure_1]))

([Control_Structure_1] of Control_Structure
  (Root_Element [Eligibility_Criteria_1])
  (Elements [Eligibility_Criteria_1], [Branching_1],
    [Add_Entities_By_Relations_1],
    [Add_Entities_By_External_1],
    [Synchronization_1], [Logical_Operator_1],
    [Generate_Advice_From_Entities_1]))

([Eligibility_Criteria_1] of Eligibility_Criteria
  (Criteria <Empty>)
  (Satisfied [Branching_1])
  (Otherwise <Empty>)
  (Output_Set [Set_1]))

([Branching_1] of Branching
  (Branches [Add_Entities_By_Relations_1], [Add_Entities_By_External_1])
  (Selection_Method all_off)
  (Order_Constraint any_order))

([Add_Entities_By_Relations_1] of Add_Entities_By_Relations
  (Input_Set [Set_1])
  (Source_Relation <Empty>)
  (Output_Set [Set_2])
  (Successor [Synchronization_1]))

([Add_Entities_By_External_1] of Add_Entities_By_External
  (Context_Class <Empty>)
  (Output_Set [Set_3])
  (Successor [Synchronization_1]))

([Synchronization_1] of Synchronization
  (Continuation wait_for_all)
  (Successor [Logical_Operator_1]))

([Logical_Operator_1] of Logical_Operator
  (Input_Sets [Set_2], [Set_3])
  (Operation <Empty>)
  (Output_Set [Set_4]))

([Generate_Advice_From_Entities_1] of Generate_Advice_From_Entities
  (Entity_Sets [Set_1], [Set_4])
  (Message_Template <Empty>))

([Refiner_1] of Refiner
  (Mapping_Type one-to-one)
  (Target [Logical_Operator_1.Operation]))

([Refiner_2] of Refiner
  (Mapping_Type one-to-one)
  (Target [Generate_Advice_From_Entities_1.Message_Template]))

([Intermediate_Role_1] of Intermediate_Role
  (Mapping_Type one-to-one)
  (Target [Eligibility_Criteria_1.Criteria]))

([Intermediate_Role_2] of Intermediate_Role
  (Mapping_Type one-to-one)
  (Target [Add_Entities_By_Relations_1.Source_Relation]))

([Intermediate_Role_3] of Intermediate_Role
  (Mapping_Type one-to-one)
  (Target [Add_Entities_By_External_1.Context_Class]))

```

Figure 6: A part of the selection PSM in terms of instances. The values of the *Goal*, *Description* and *visualization* attributes are not shown here

The `control` attribute of the `PSM` class stores the PSM's control structure in terms of instances. However, as the method library only contains abstract PSMs that are not yet further specified, the refiners and knowledge roles are also not yet specified. The specification process that results in application-specific PSMs is carried out during the knowledge acquisition phase. Figure 6 presents a part of the unrefined selection PSM in terms of primitives from the method ontology.

The core of the selection PSM consists of the `PSM_1` instance. Besides attributes that describe the PSM's pragmatics (e.g., `Name`, `Author` and `Description`), the `PSM_1` instance also contains attributes that define the capabilities of the PSM such as the `Goal`, `Parameters`, `Visualization` and `Control` attributes. Similar to the `:REFINERS` and `:MAPPINGS` attributes of the `:STANDARD-PRIMITIVE-CLASS` metaclass, the `Refiners` and `Mappings` attributes of the PSM are used to define an application-specific PSM. The selection PSM in figure 6 defines two refiners and three mappings, which contain references to specific attributes of instances that make up the PSM's control structure. As all knowledge roles of the selection PSM are used only internally and not by other PSMs, these knowledge roles are defined as intermediate. The control structure of the selection PSM consists of seven primitives. Each primitive is characterized by means of a number of attributes, which may refer to global structures that contain information that is used throughout the control structure (e.g., `set_1`). The first step is an instance of the `Eligibility_Criteria` class. This primitive (derived from `K_Of_N_Criteria`) specifies whether the physician's newly selected decision or action applies to this PSM. The `Criteria` attribute formally defines the actual action or decision. However, as the actual action itself is specified in the refinement process, it has no specific value yet in the abstract PSM's control structure. The newly selected action or decision is also stored in `set_1` for later use. Whenever the eligibility criteria hold, the `Branching` and `Synchronization` primitives are used to determine all information relevant for detecting possible conflicting situations, related to the application of the selection PSM (details on the attributes of the `Branching` and `Synchronization` classes are described elsewhere [8]). The `Add_Entities_By_Relations_1` instance retrieves all possible conflicting situations, based on the newly selected action or decision and the value of the `Source_Relation` attribute. The `Add_Entities_By_External_1` instance retrieves all relevant previously carried out actions and decisions from an external source such as an EPR (the `Context_Class` attribute specifies which type of actions and decisions must be acquired from the external source). The `Logical_Operator_1` instance determines present conflicting situations by comparing all possible conflicting

situations (stored in `set_2`) with the actual actions and decisions (stored in `set_3`). Present conflicting situations are reported by the `Generate_Advice_From_Entities_1` instance (the message is generated from the `Message_Template` attribute). `set_1`, `set_2`, `set_3` and `set_4` are instances of the `set` class. The next section presents an example of a refined selection PSM that reports drug interactions, as well as an example of the use of the PSM's `visualization` attribute.

3.4 Stage 3: authoring guidelines

3.4.1 Overview

The KA-Tool, used to facilitate the guideline authoring process, consists of a collection of modular components. An overview of these components is shown in figure 7.

The core of the KA-Tool is a design-time interpreter, which provides a means of communication between the user and various knowledge managers such as a domain manager, a guideline manager and a method manager. The design-time interpreter combines information from all available managers and creates a user interface that enables guideline authors to develop guidelines in terms of PSMs and primitives.

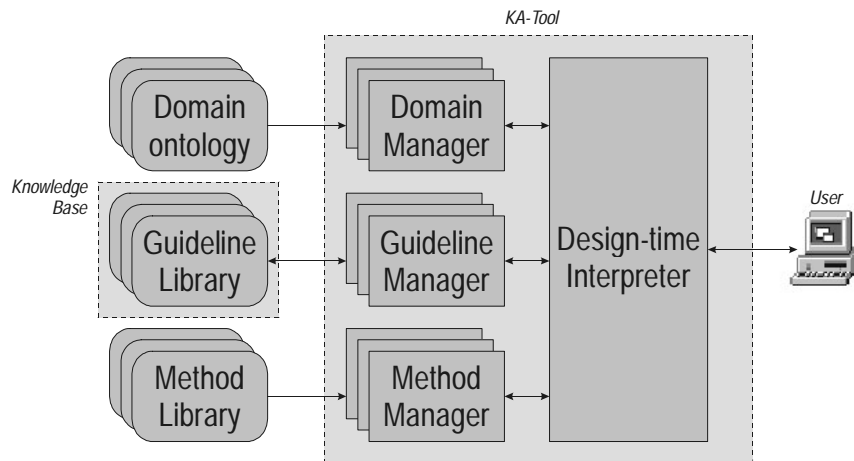


Figure 7: System overview of the components in the KA-Tool

3.4.2 Authoring primitives

The KA-Tool can be used to represent guidelines by means of primitives as well as a PSM. When a guideline consists of primitives, guideline authoring becomes a two-phased process.

The first phase consists of describing the guideline's structure in terms of primitives and subguidelines (flow control). In this phase, all primitives and

subguidelines are treated as black boxes with no domain-specific content. As mentioned earlier, the control structure of guidelines that do not consist of a single PSM are visualized by means of a flowchart by default. Figure 8 shows a part of the hypertension guideline in the KA-Tool, visualized in terms of a flowchart.

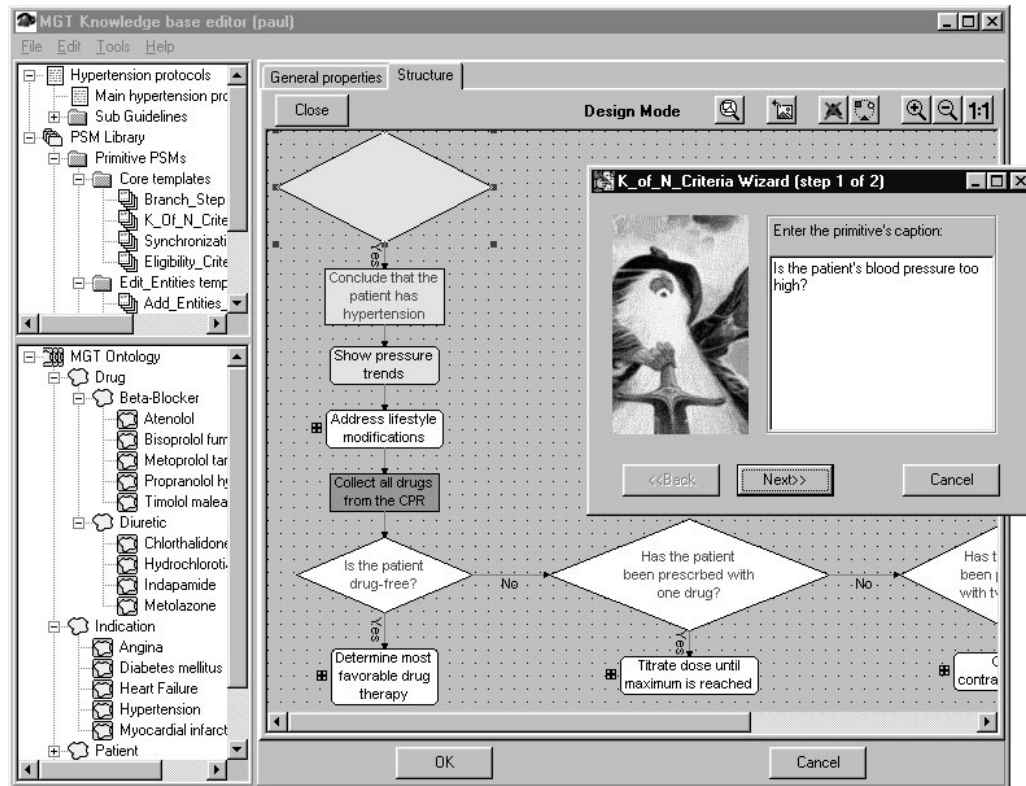


Figure 8: The user interface of the Gaston KA-Tool, used for defining the control structure of a hypertension guideline in terms of primitives. Primitives with a + sign (e.g., 'Address lifestyle modifications') are references to subguidelines

The KA-Tool generally consists of three panes. The lower left pane shows all concepts stored in an application-specific domain ontology, visualized through the domain manager, which loads a domain ontology (this particular domain ontology was developed for use in the domain of hypertension) and passes it to the design-time interpreter. The upper left pane presents an overview of all guidelines that are present in the guideline library. Similar to the domain manager, the guideline manager loads and visualizes the contents of the guideline library. Whenever a guideline is selected in the upper left pane, the guideline manager combines the guideline's control structure (stored in the `control` attribute) with the guideline's visualization information (stored into the `visualization` attribute) and shows a detailed description of the guideline through the design-time interpreter in the right pane.

The method library contains all unrefined primitives and PSMs, which are also shown in the upper left pane. When a guideline author wants to add a new primitive to the guideline's control structure, this primitive is selected in the upper left pane and dragged onto the right pane. For example, the topmost element in figure 8 is an instance of the `κ_of_N_Criteria` primitive, which was selected from the upper left pane. As a result, the method manager activates a primitive-specific 'wizard', also shown in figure 8. This wizard enables the author to refine the selected primitive by specifying the primitive's intermediate roles, based on the contents of the `:REFINERS` and `:PARAMETERS` attributes. The method manager has access to the domain ontology through the design-time interpreter for mapping concepts from the domain ontology onto the intermediate roles. Regarding instances of the `κ_of_N_Criteria` class for example, the wizard enables guideline authors to specify values for the `caption` and `κ` attributes (see also figure 5). The actual criteria itself are not defined in this phase.

Every primitive is represented by a separate step in the flowchart. The shape and colors of each primitive are defined in the `:VISUALIZATION` attribute of the primitive's metaclass. As shown in figure 5, the value of this attribute refers to an instance of the `Primitive_Visualization_Information`. Figure 9 shows the contents of this instance that describes the visualization information regarding the `κ_of_N_Criteria` primitive.

```
/* *** K_Of_N_Criteria Visualization definition ***
([Primitive_Visualization_Definition_1] of Primitive_Visualization_Definition
  (Shape [Shape_Definition_1])
  (User_Interface_Procedure "coreLib.K_Of_N_Criteria_CreateInterface"))

([Shape_Definition_1 of Shape_Definition
  (Shape_Type diamond)
  (Shape_Background_Color yellow)
  (Shape_Foreground_Color black))
```

Figure 9: An instance of the `Primitive_Visualization_Information` class that describes the visualization information regarding the `κ_of_N_Criteria` class

The appearance of each primitive is defined in the `shape` attribute, which describes the primitive's shape in the flowchart (e.g., each `κ_of_N_Criteria` instance is represented by a yellow diamond containing black characters).

Besides the appearance of each primitive in a flowchart, the `Primitive_Visualization_Definition` class also defines another type of visualization information that is used in the second phase of the guideline acquisition process. This phase consists of specifying domain knowledge that is required by the various primitives such as specifying the actual criteria in the 'Is the patient's blood pressure too high?' element (figure 10).

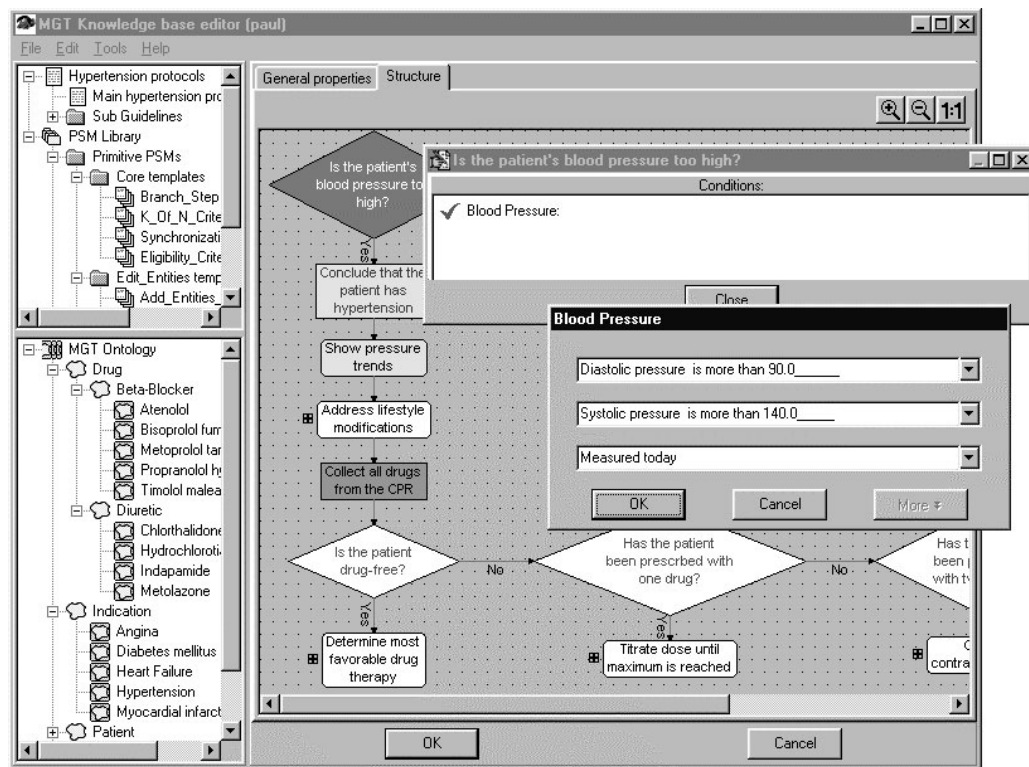


Figure 10: The user interface of the Gaston KA-Tool, used for specifying primitive-specific domain knowledge to 'flesh-out' primitives with domain knowledge

In this phase, the value of the `User_Interface_procedure` attribute is used, which refers to a procedure name in a function library that is able to create a primitive-specific user interface. In this case, the `K_Of_N_Criteria_CreateInterface` procedure in the `coreLib` function library is executed with the selected instance as a parameter. As a result, a user interface is created that enables a guideline author to define a number of criteria.

These criteria are formed by instantiated entities from the domain ontology, which are selected from the lower left pane and dragged to the 'conditions' window in the right pane. Regarding the 'Is the patient's blood pressure too high?' element in figure 10 only a single criterion is entered, which states that the patient's blood pressure is too high when the systolic blood pressure exceeds 140 mm Hg and the diastolic blood pressure exceeds 90 mm Hg. For each criterion, it is possible to edit the underlying domain entity's attributes. For example, `Blood_Pressure` is a class that is defined in the hypertension domain ontology. This concept has three attributes: `Diastolic_pressure` and `Systolic_pressure` that are of type number, and `Measured` that is of type date. Based on the attribute's type, relations are displayed (e.g., less than, equals)

that enable an author to specify certain conditions such as ‘more than 90’, ‘less than 90’, ‘equals 90’, ‘today’ and ‘more than 2 days ago’. Figure 11 shows the actual representation of the criterion of the ‘Is the patient’s blood pressure too high?’ element in terms of instances as stored in the guideline library.

```

;+ *** Criterion as instance definition ***
([K_Of_N_Criteria_1] of K_Of_N_Criteria
  (Caption "Is the patient's blood pressure too high?")
  (K <empty>)
  (Criteria [Criterion_1])
  (Satisfied [Start_Activity_1])
  (Otherwise <empty>))

([Criterion_1] of Criterion
  (Relation_Operators "Diastolic_pressure=More",
    "Systolic_pressure=More",
    "Measured=Equals")
  (Target [Blood_Pressure_1])
  (Next_Criterion <empty>))

([Blood_Pressure_1] of Blood_Pressure
  (Diastolic_pressure 90)
  (Systolic_pressure 140)
  (Measured "today"))

```

Figure 11: A criterion as a number of instances in the guideline library

Each criterion is modeled by means of an instance of the `criterion` class, which contains relations operators (e.g., ‘more’, ‘less’, ‘increases’, ‘decreases’, ‘equals’, ‘last’) as well as a reference to an instance (stored in the `Target` attribute) of a domain ontology class (e.g., `Blood_Pressure`) of which its attributes may contain the criterion’s values. The `Next_Criterion` attribute, empty in this case, may hold a reference to another instance of the criterion in case it consists of multiple parts (this implements the OR-operator). The `k` attribute in the `k_of_N_Criteria_1` instance is left undefined, meaning that `K` is equal to `N` (all criteria must be evaluated to true). The `otherwise` attribute contains no value, indicating that the guideline ends whenever this criterion does not evaluate to true. The `satisfied` attribute refers to an instance of a `start_Activity` primitive, which models an action that starts a new activity [36] such as establishing a new diagnosis (e.g., the patient is diagnosed as having hypertension) or starting a new treatment.

In order to model guidelines that contain activities, the core method ontology again was extended with primitives that represent activities. The `start_Activity` class for example, models an action that starts a new activity (e.g., start a new treatment). The `Treatment` class is an example of an activity that models a treatment such as prescribing a new drug. The `Activity_start` and `Activity_End` attributes of this class specify the start and endpoints of the treatment. Other attributes of this class are the `Activity_Class` and the `Activity_Attributes` attributes that are intermediate knowledge roles that

refer to the domain entities, specified in the activity (e.g., the dosage of a drug). This extended ontology is described in more detail elsewhere [24].

```

;+ *** Start Beta-Blocker definition ***
([Start_Activity_1] of Start_Activity
  (Current_Activity [Treatment_1])
  (Starting_Value minimum))

([Treatment_1] of Treatment
  (Activity_Class [Beta-Blocker])
  (Activity_Attributes [Dose])
  (Activity_Start "now")
  (Activity_End "now+24h"))

```

Figure 12: The prescription of a Beta-Blocker in terms of primitive instances. The instances denote that the prescription starts immediately with a minimum dosage and ends after 24 hours (the uncertainty part of the time annotation is not specified here)

Activities are used throughout the hypertension guideline. For example, another section of this guideline states that a patient with high blood pressure must be prescribed a standard anti-hypertensive (usually a Beta-blocker or Diuretic), unless there are compelling indications that favor the prescription of another drug. Also, a low dosage of the prescribed drug should be used, slowly titrated upward using a patient-specific schedule. Figure 12 shows the prescription of a Beta-Blocker in terms of the above-mentioned primitives.

3.4.3 Authoring PSMs

In contrast to defining the control structure of a guideline in terms of primitives (phase 1), the control structure of a guideline that is executed by means of a PSM is not explicitly described by a guideline author as this structure is already defined in the PSM internally. When a guideline author creates a guideline in terms of a PSM, the PSM is selected in the upper right pane, similar to the selection of a primitive. The method manager then reads and copies the control structure of the abstract PSM in the method library, after which the guideline author is able to create an application-specific PSM by filling in specific values for the `Refiners` and `Mappings` attributes through a wizard, again similar to the use of primitive-specific wizards. For example, an application-specific selection PSM was included in the hypertension guideline in order to report drug interactions. For this purpose, two refiners and three intermediate roles were specified through the wizard by filling in the values of the corresponding attributes. Figure 13 shows the control structure of the refined selection PSM after the intermediate knowledge roles were specified.

```

;+ *** refined selection PSM control structure ***
([Eligibility_Criteria_1] of Eligibility_Criteria
  (Criteria [Criterion_1])
  (Satisfied [Branching_1])
  (Otherwise <Empty>)
  (Output_Set [Set_1]))

[Criterion_1] of Criterion
  (Target [Drug_1])

([Drug_1] of Drug)

([Branching_1] of Branching
  (Branches [Add_Entities_By_Relations_1], [Add_Entities_By_External_1])
  (Selection_Method all_off)
  (Order_Constraint any_order))

([Add_Entities_By_Relations_1] of Add_Entities_By_Relations
  (Input_Set [Set_1])
  (Source_Relation [Has_Interactions])
  (Output_Set [Set_2])
  (Successor [Synchronization_1]))

([Add_Entities_By_External_1] of Add_Entities_By_External
  (Context_Class [Drug])
  (Output_Set [Set_3])
  (Successor [Synchronization_1]))

([Synchronization_1] of Synchronization
  (Continuation wait_for_all)
  (Successor [Logical_Operator_1]))

([Logical_Operator_1] of Logical_Operator
  (Input_Sets [Set_2], [Set_3])
  (Operation AND)
  (Output_Set [Set_4]))

([Generate_Advice_From_Entities_1] of Generate_Advice_From_Entities
  (Entity_Sets [Set_1], [Set_4])
  (Message_Template "You have prescribed %Set_1[0].name%. However, the
    patient has also been prescribed with
    %Set_4[*].name%, which is a known interaction of
    %Set_1[0].name%. This interaction is known as
    %Set_4[*].attrs[0].value%"))

```

Figure 13: The control structure of an application-specific selection PSM that reports drug interactions

In this case, the two refiner attributes have been specified by filling in certain values such as the `operation` attribute which has been ascribed the value `AND` to obtain the conjunction of two sets, and the `Message_Template` attribute which now contains the advice, shown to a user. Similarly, the three knowledge roles now contain explicit mappings to concepts from the domain ontology. For example, the value of the `criteria` attribute of the `Eligibility_Criteria_1` instance now refers to a drug to indicate that this PSM is triggered whenever a new drug is being prescribed. Also, the `source_relation` attribute is mapped onto to the `Has_Interactions` attribute of the `Drug` class to specify that the `Has_Interactions` attribute models drug interactions and the value of the `context_class` attribute now refers to the `Drug` class to acquire all prescribed drugs from an external source (e.g., EPR).

By applying different refinements and mappings, the selection PSM can be used to solve various selection tasks. Besides selection tasks, other tasks, such as monitoring tasks, preparation tasks and responding tasks [11] can also be solved by PSMs, similar to the one that was used to solve the selection tasks. The application-specific selection PSM, shown in figure 13, is further specified using simple straightforward mappings (although even this example has been somewhat simplified for the convenience of the reader). In many situations however, mapping terms from one (domain or method) ontology to another is not very straightforward. Problems arise if mappings are not one-to-one or, even worse, when there exist semantic differences between the various ontologies [37].

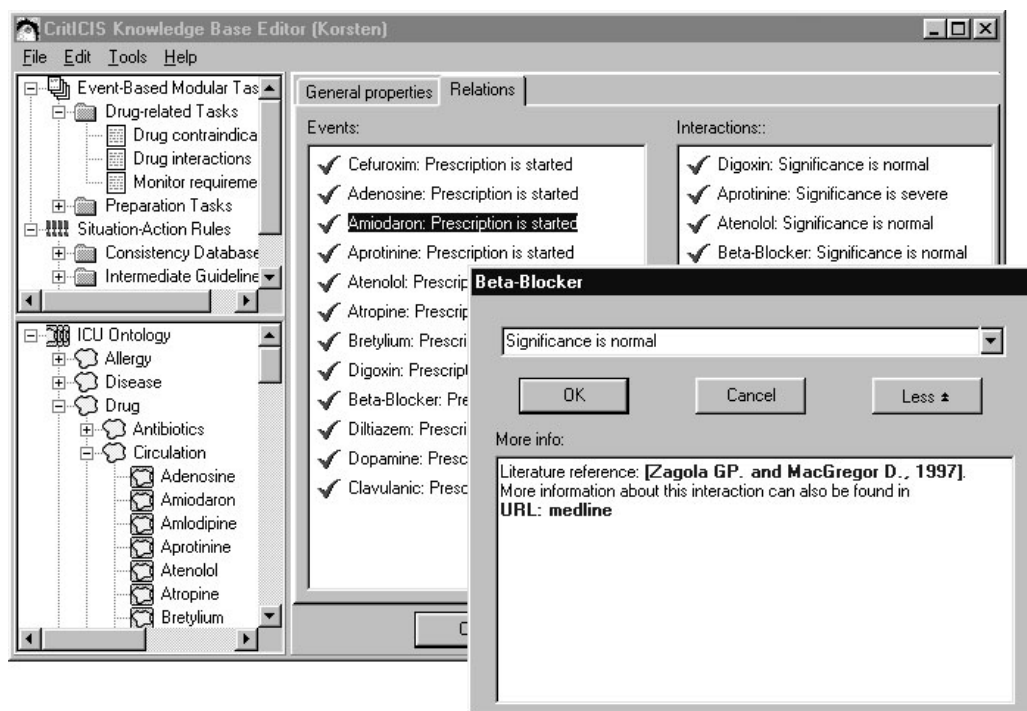


Figure 14: The user interface generated by the selection PSM to represent the drug interaction task

As shown earlier, each primitive contains information to visually represent itself in the KA-Tool. Therefore, a user interface can be automatically constructed, as described in the previous section. In case of a PSM, however, each PSM performs a single task from the viewpoint of the guideline's author. Based on the control structure and visualization information of the PSM, the guideline manager constructs a user interface through the design-time interpreter that reflects this viewpoint. For example, figure 14 shows the user interface of the KA-Tool used to acquire drug interactions. It uses the visualization information from the PSM to override the default flowchart user

interface and provides a means for entering domain-specific knowledge in terms of knowledge roles such as drugs and their interactions.

As shown in figure 6, the visualization information of the selection PSM is stored in the `Complex_Visualization_Information_1` instance of the `Complex_Visualization_Information` class. Figure 15 shows the contents of the `Complex_Visualization_Information_1` instance (although somewhat simplified).

```
;+ *** selection PSM Visualization definition ***
([Complex_Visualization_Definition_1] of Complex_Visualization_Definition
  (Elements [Visualization_Element_1])
  (Root_Element [Visualization_Element_1])
  (Visualization_Control static))

([Visualization_Element_1] of Visualization_Element
  (Target [Add_Entities_By_Relations_1])
  (Visual_Mapping_Type tab_page)
  (Visual_Mapping_ID "Relations_Tab")
  (Visual_Mapping_Alignment client))
  (Next_Visualization_Elements <empty>))
```

Figure 15: The contents of the `Complex_Visualization_Information_1` instance, used to visualize the selection PSM

The user interface of a PSM is built from a number of visual elements, which are normally created by executing (a combination of) primitive-specific visualization procedures such as the `K_of_N_Criteria_CreateInterface` procedure, although it is also possible to develop new visualization procedures that are PSM-specific. Regarding the selection PSM that reports drug interactions for example, guideline authors only have to define new relations that describe drug interactions. As this part of the PSM is handled by the `Add_Entities_By_Relation` primitive, the PSM's user interface is constructed by executing the visualization procedure, defined in the `:VISUALIZATION` property of the `Add_Entities_By_Relation` primitive. The format of each visual element is defined in an instance of the `Visualization_Element` class. For example, the visual element that allows for the definition of drug interactions in figure 14 is defined in the `Visualization_Element_1` instance in figure 15. The applied visualization procedure is determined through the `Target` attribute, which refers to an instance in the PSM's control structure (e.g., the `Add_Entities_By_Relations_1` instance in figure 14). The `Visual_Mapping_Type`, `Visual_Mapping_ID` and `Visual_Mapping_Alignment` attributes determine the visual element's appearance in the KA-Tool. In this case, the values `tab_page`, `relations_tab` and `client` imply that the user interface element, created by the visualization procedure of the `Add_Entities_By_Relation` primitive, is mapped onto a (new) tab page and that the interface element must cover the entire area of the tab page. The

`Visualization_Control` attribute of the `Complex_Visualization_Information_1` instance defines in what way and order the group of visual elements must be visualized. In straightforward PSMs such as the selection PSM, all visual elements are shown instantaneously, i.e., there is no dynamic control structure that defines a particular order among multiple visual elements. Regarding more complex PSMs such as a Cover-and-Differentiate [34], domain-specific knowledge that is acquired through one visual element may dynamically determine its successor. This however, is not the case in the selection PSM.

As a result of the contents of the `Complex_Visualization_Information_1` instance, the refined selection PSM is represented by means of an *Events* pane and an *Interactions* pane (shown in figure 14), where each entity in the *Event* pane denotes a newly prescribed drug. The content of the *Interactions* pane depends on the selected event in the *Events* pane and lists all known interactions of the newly prescribed drug. Drugs are selected from the domain ontology in the lower left pane and dragged onto the *Events* or *Interactions* pane. Every drug that is linked to an event by dragging it to the *Interactions* pane creates a `Has_Interactions` relation in the guideline library between the event and the dragged drug (and also the other way around, as this relation is bilateral). It is also possible to edit the relation's attributes (e.g., the `significance` attribute of the `Interaction_Relation` class) and provide more information on each relation such as literature references or hyperlinks.

3.4.4 Implementation of the KA-Tool's component architecture

The component architecture of the KA-Tool is implemented as a kernel (the design-time interpreter), which loads a number of plugins. Each plugin contains a manager such as a guideline manager, a method manager or a domain manager (see also figure 7). As a result, the functionality of the KA-Tool is entirely defined by means of the loaded plugins. For example, when the method manager plugin (or a plugin with a similar functionality) is not loaded, it is not possible to construct new guidelines in terms of primitives or PSMs, as the primitive- and PSM-specific wizards are defined in the method manager plugin. However, when a guideline manager plugin as well as a domain manager plugin is loaded, it is possible to define primitive- or PSM-specific domain knowledge such as the drug interactions, shown in figure 14 or the criteria of the 'Is the patient's blood pressure too high?' element, shown in figure 10.

Similar to ontologies, plugins in the KA-Tool are also derived from core plugins to introduce additional functionalities. For example, Gaston was used in a number of projects [38-40] that aimed at the development of reminder

systems, based on rule-based guidelines [41]. For this purpose, a guideline manager was developed with access to a guideline library as well as a method library. In this case, the method library contained a single rule-based strategy that modeled the control structure of rule-based guidelines in terms of primitives. By means of the visualization information of this strategy, authors were able to create and implement new rule-based guidelines through the plugin, without any knowledge on the underlying guideline's control structure [24].

The design-time interpreter communicates with all loaded plugins through a standard interface: it does not differentiate between different types of plugins such as domain manager plugins, guideline manager plugins and method manager plugins. As there is also no limit on the number of loaded plugins, multiple plugins of similar functionalities can be loaded. Therefore, the KA-Tool may contain multiple domain ontologies, method libraries or guideline libraries. An example of the latter can be found in the KA-Tool used to develop guidelines for the CritICIS system, a real-time reminder system for use in Intensive Care Units [38]. By means of the CritICIS KA-Tool, guideline authors were able to define guidelines through multiple guideline managers.

3.5 Stage 4: executing guidelines

When instructed, the KA-Tool retrieves the control structure of each guideline and PSM and creates a structure that consists solely of primitives. By combining this structure with the implementation procedures that are attached to each primitive, it is automatically compiled into a more efficient description. Just as the `:VISUALIZATION` property of each primitive refers to a primitive-specific visualization procedure used by the KA-Tool, the `:PROCEDURE` property of a primitive contains the name of a primitive-specific implementation procedure. The requirements of this description differ from the requirements of the ontological representation that is used during the knowledge acquisition process. The latter representation is sufficiently abstract and clear to be comprehended by guideline authors, whereas the execution-time representation is optimized to meet execution-time requirements such as compactness and execution speed. During execution time, the compiled representation forms a symbol-level knowledge base that is processed by a decision support system. Similar to the architecture of the KA-Tool, the decision support system consists of a collection of modular components, implemented as a kernel that loads a number of plugins. Again, the kernel (execution-time scheduler) communicates with all loaded plugins through a standard interface that does not differentiate between different types of plugins. Therefore, the decision support system may contain various types of plugins, each with a different functionality. In order for a guideline-based

decision support system to be flexible and reusable in various application domains, a number of plugins were developed in the Gaston project that define typical functionalities such as plugins that traverse the knowledge base and plugins that define communication interfaces with the outside world (e.g., EPRs or EPR users). An overview of the execution-time scheduler with a number of typical plugins is shown in figure 16.

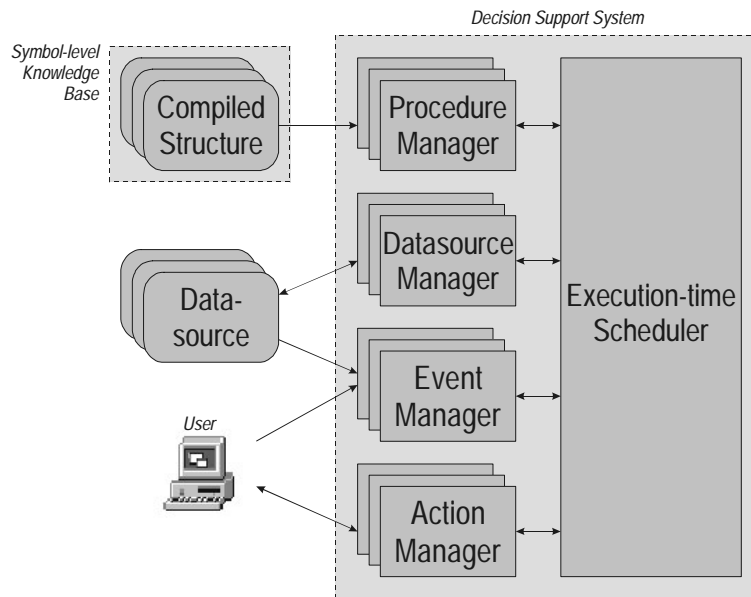


Figure 16: System overview of the components in the Decision Support System

Four different types of plugins are shown, which all communicate through the execution-time scheduler.

The Procedure Manager plugin traverses the compiled structure in the compiled structure and informs the execution-time scheduler which procedures are to be executed and in what order.

The Datasource Manager makes use of standard communication protocols to exchange clinical data with data sources. Examples of data sources are EPRs that contain clinical data (e.g., prescribed drugs or established diagnoses) or patient monitors that contain physiological data (e.g., heart rate or ECG). The defined protocols allow for a two-way communication, enabling data acquisition as well as data storage. An example of the latter is the hypertension guideline, which uses an instance of the `start_Activity` class (the second step in figure 8) to automatically diagnose the patient as having hypertension if the patient's blood pressure is too high. This diagnosis is then stored into the patient's EPR through the Datasource Manager. However, this feature is normally used with caution in decision support systems.

The Event Manager defines protocols for the specification of pertinent events that trigger the execution of certain guidelines. These events may originate from other systems such as an EPR (e.g., the prescription of a new drug) or from a user that requests the execution of a guideline manually.

Finally, the Action Manager plugin is developed to establish a means of communication with the users of the decision support system such as the way in which advice is being presented to a user when necessary (e.g., reminders). Also, in case the decision support system is used as a consultation system [42], this plugin facilitates a dialog between the system and the user.

The screenshot shows a Microsoft Access window titled "Microsoft Access - [Medication]". The patient information is "Patient: 13043255510 Smith". The "Administration time:" is displayed as a grid of days from 1 to 11, with columns for Day, 4, 6, 8, 0, 2, 4, 2, 4, 6, 8, 0, 2. The medication list includes:

Medicine	Freq.	Unit + Pharm. form	Admini- stration	Begin	End	Day	1	1	2	2	2	1	1
Bretylium		μG/KG/Min	5 mg/kg	i.v.	5/28/98	KTN	0						
Ceftazidim	3	X DGS	1000 mg	i.v.	5/28/98	KTN	0						
Allopurinol	1	μG/KG/Min	300 ml	i.v.	5/28/98	KTN	0						
Angiotensin 2			2.5 mg	i.v.	5/28/98	KTN	0						
Atropine		BONUS	0.5 mg/ml	i.v.	5/28/98	KTN	0						
Labetalol		VLG AFSPH	1 mg/ml	i.v.	5/28/98	KTN	0						
Digoxin	1	μG/KG/Min	25 mg	i.v.	5/28/98	KTN	0						
Allopurinol	1												

A recommendation dialog box is overlaid on the medication list, titled "Recommendation regarding patient Smith". It contains the following text:

The following advice is given:

- You have prescribed Amiodaron. However, the patient has also been prescribed with Digoxin, which is a known interaction of Amiodaron. This interaction is known as normal. Is the advice correct? ☒ yes ☐ no (explanation)

Literature reference: [Zagola GP, and MacGregor D., 1997]. More information about this interaction can also be found in [medline](#)

The dialog box has "OK" and "Cancel" buttons. At the bottom of the main window, there are buttons for "Add Medication", "Stop Medication", "Pumpdose", "OK", "Change Medication", and "Info".

Figure 17: Advice generated by the selection PSM. The advice is shown, overlaying the user interface of a specific EPR system

Figure 17 shows an example of the use of the decision support system in daily care. In this case, a physician has prescribed a new drug for a given patient by means of entering it into an EPR system. As a result, the EPR system activates the decision support system with a *Prescribe_New_Drug* event, which is processed by the Event Manager plugin. The EPR system also supplies additional parameters such as the patient's ID and the name of the started drug. Among other tasks, this event causes the refined drug

interaction PSM to be executed by the Procedure Manager plugin, which retrieves all known drugs that have an interaction relation with the started drug and queries the EPR through the Datasource Manager plugin to determine whether one of them is present. Whenever this is the case, the system reports these interactions to the user through the Action Manager plugin by means of pop-up windows. As this picture was taken from a test environment [38], users are able to validate the advice to improve the guideline's correctness and helpfulness.

Although each plugin in the decision-support system defines a different functionality, their functions are not mutually exclusive. For example, the tasks the Action Manager plugin performs can be viewed as comparable to the tasks the Datasource Manager plugin performs in case the user is regarded as a data source. Similarly, the Datasource Manager plugin as well as the Event Manager plugin define a communication protocol with external data sources such as an EPR. However, the plugins, shown in figure 16, were developed independently of each other to facilitate the reusability of the framework among different application domains. An example is the implementation of the CritICIS system. During this project, an ICU replaced their old EPR system with a modern one. In order for the CritICIS system to function, a new Datasource Manager plugin was designed to acquire data from and store data in the new EPR system. The other plugins did not have to be changed, which shortened the time span necessary to migrate from the old EPR system to the new one. Also, by adding new functionality to the Action Manager plugin, the system could be reused to execute guidelines such as the hypertension guideline over the internet [43]. An example is shown in figure 18.

Plugins not only communicate with the outside world, but also exchange required information among each other through the execution-time scheduler. For example, whenever a primitive's procedure requires certain data (e.g., to evaluate a criterion), the Procedure Manager plugin sends a request to the Datasource Manager plugin to obtain this data. The other way around, in order to acquire this data, the Datasource Manager plugin requires the location of the data in the data source (e.g., EPR). As mentioned earlier, this information is stored into the `:DATASOURCE` attribute of each domain concept. To acquire this information, the Datasource Manager plugin requests this information from the Procedure Manager plugin that has access to the compiled structure where this information resides.

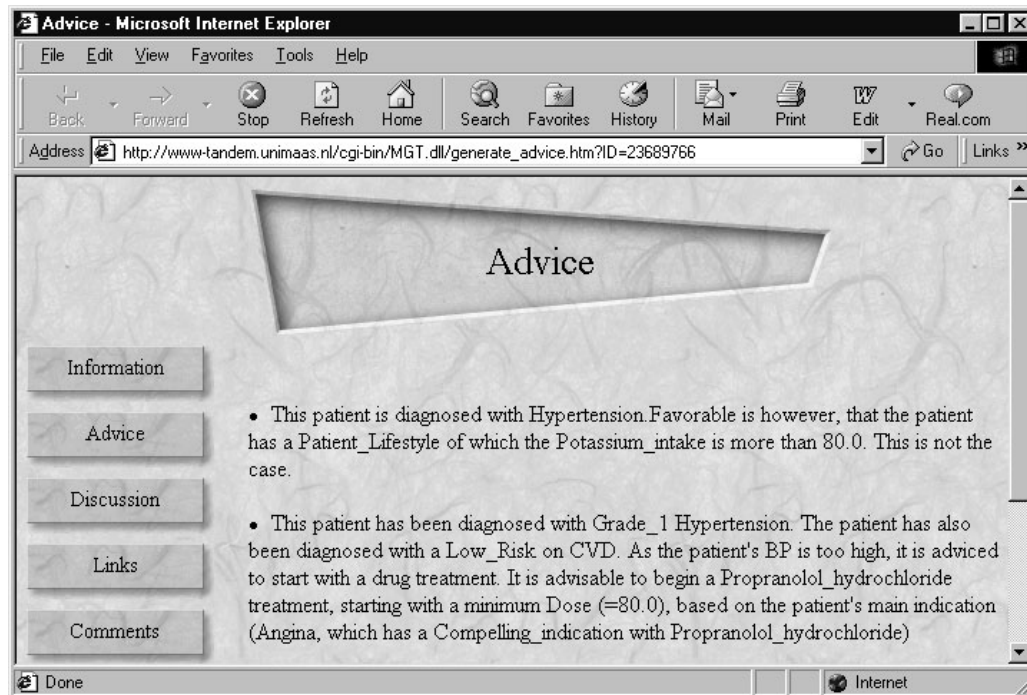


Figure 18: Advice, generated by the Gaston guideline execution system that is incorporated in an internet-based health record system

4 Results

The methodology described in this paper was used to develop a number of guidelines and decision support systems that differ in granularity, complexity and application domain.

The CritICIS system is a real-time reminder system used in critical care environments such as Intensive Care Units. The domain ontology of this system is based on the IMPACT Minimal Standard Data Set, a set of medical terms describing the state of a patient in an Intensive Care Unit [44]. At present, the ontology consists of about 2000 divided into about 100 categories. After a short training period, care providers found the user interface of the KA-Tool as well as the decision-support system useful and sufficiently 'intuitive'. In order to determine the validity of entered knowledge, the CritICIS system has undergone a first validation, in which guidelines were tested on a large patient data set of previously admitted ICU patients. This validation, described in detail elsewhere [38], showed that 88% of all issued reminders were classified as correct. The CritICIS system is now fully operational in the 20-bed ICU of the Catharina Hospital, Eindhoven, the Netherlands.

Another system that has been developed by means of the Gaston framework is the GRIF system, developed to change Family Physicians' (FP) test

ordering behavior by focusing on the appropriateness of test requests [39]. Using a retrospective random selection of 253 request forms the comments of human experts to the comments of the reminder system were compared. A panel of three expert physicians judged the requested tests independently based on interpretations of the practice guidelines. The majority assessment of the physicians was compared to the assessment of the reminder system. In case the system's output differed from the majority assessment the written practice guidelines were consulted. On average 4.8 reminders were produced per form. In total 32 of the 442 given reminders (7%) were given incorrectly. The amount of information and the level of detail (the specificity of the terms) in which the FP describes the patients' medical status are crucial for the reminder system to react correctly. Details are described elsewhere [45].

The Multidisciplinary Psychoactive Drug Selection –advisor system (M-PADS) is a decision support system developed for selecting the most appropriate psychoactive drug in order to treat psychiatric patients [46]. It contains guidelines and PSMs that solve a number of tasks, varying from tasks that are similar to the earlier-mentioned drug interaction tasks, to more complex ones that process 'deep knowledge' (using a semantic network). In contrast to the other examples that used Protégé for developing the domain ontology, the domain ontology for this system was developed with the help of the GALEN approach [47]. A first evaluation is currently ongoing.

Finally, decision-support systems were developed that provide advice through the Internet. These systems were integrated in a web-based consumer health record system, which can be used both by care providers and patients to enter and share medical and patient information [43]. Two pilot projects concerning this topic are currently in progress: 1) the TANDEM project [40], which focuses on the treatment of Diabetes and 2) the earlier-mentioned Medical Guideline Technology (MGT) project [33] that focuses on the treatment of Hypertension. The TANDEM system has now been in operation for several months. A pilot study is in progress with 10 patients and two care providers. Patients as well as care providers appear to have accepted the system very well. The pilot study will continue for another three months. The pilot study of the MGT project is currently in its initial phase.

5 Discussion

Although the number of guideline-based decision-support systems increased rapidly during the last years, the number of systems that are actually used in daily practice is still very small. The use of a framework as described in this paper may increase the number of systems that are used in practice as it covers all stages in the guideline development process, from the guideline

acquisition phase to the guideline execution phase. The use of primitives, PSMs ontologies and plugins facilitates guideline reusability as well as shareability. As the ontological representation is extendible and represents a guideline on different levels, it is scalable, flexible and expressive enough to define guidelines that differ in complexity and application domain, as is illustrated by the number of systems that were developed by means of the framework.

In contrast with other approaches that model PSMs, this framework differs slightly as it models the control structure in terms of explicit primitives that each describe a different aspect of the PSM. In other approaches, the control structure of a PSM often primarily consists of other PSMs (in case a PSM is no longer decomposable, such a PSM is referred to as a primitive PSM or mechanism [26]), where the behavior of each (primitive) PSM is defined by means of a formal language such as predicate logic (which automatically also requires the development of a general interpreter that is able to execute the logical statements efficiently). This approach can be implemented by the Gaston framework, for example by means of creating a `Primitive_PSM` class that contains an attribute (e.g., `Operational_Definition`), which holds a formal description of the PSM's behavior [48, 49]. In this case, the implementation procedure stored in the `:PROCEDURE` attribute refers to a general interpreter which executes the description (e.g., a CLIPS or a PROLOG interpreter). The approach, described in this paper however, defines the control structure of a PSM in terms of different elements (e.g., primitives or other PSMs) in order to 1) use the same representation for defining guidelines in terms of PSM as well as primitives, 2) attach visualization information to each class to facilitate the knowledge acquisition phase and 3) attach execution procedures to each class to automatically create a run-time decision-support system. The number of decision support systems, used in daily practice that are based on PSMs is very limited, mainly as a result of the abstractness of a PSM. This project has started with implementing rather simple PSMs such as the selection PSM, presented in this paper (and even this example has been simplified somewhat). The examples and results illustrate that the guideline's application domain dictates its representation. Guidelines that are more complex or domain-specific usually require a more low-level representation (e.g., a set of primitives) as these guidelines are usually too specific to be captured by PSMs. Guidelines that address more generic tasks (e.g., heuristic classification or selection tasks) are more suited to be represented by means of PSMs. When guideline authors become more familiar with the application domain, they may be able to recognize certain patterns, which can be embedded into PSMs. Besides further development of

the models, we are also currently implementing more complex PSMs such as Cover-and-Differentiate and Episodic Skeletal-Plan Refinement (ESPR).

Creating a non-monolithic ontological representation supports the development of extended ontologies and new primitives. However, one must be careful not to create a large number of primitives to address very specific tasks. Instead, creating more general primitives that can be refined to represent specific behavior seems a more favorable option. As mentioned earlier, ontologies primarily consist of classes and do not contain instances. The latter are acquired during the knowledge acquisition phase. The examples show that for knowledge acquisition, various types of instances exist. For example, an instance that defines a drug interaction implies that this interaction exists for all occurrences of that drug, although it is not defined at the level of the `Drug` class. This, in contrast to an instance of the `Drug` class that is used to describe a single activity. In this case, the instance represents a particular prescription that is started on a certain date with a certain dosage. Although both situations are represented by means of an instance, their purposes are different. Therefore, they must be defined by means of different relations, instead using a standard 'instance-of' relation [50]. For example, all instances that represent drug interactions are defined as a 'interaction_instance-of' `Drug`, in contrast to an instance that represents an activity that is defined as an 'instance-of' `Drug`. The framework supports this feature through the metaclass feature of the OBKC model. As a result of applying this model, an ontology in the Gaston does not primarily consists of classes anymore, but also contains instances. For example, each primitive in the method ontology is an instance of the `:STANDARD-PRIMITIVE-CLASS` metaclass. As mentioned earlier, the `:VISUALIZATION` attribute of this class refers to an instance in another ontology that holds information to define a primitive-specific user interface in the KA-Tool. Although this information is not usually included in an ontology, this has been a deliberate design-criterion in order to facilitate the development of systems that are used in practice.

The use of a kernel and plugins allows for the creation of various application-specific systems during the guideline acquisition and execution phases. The KA-Tool for example uses plug-ins to support the definition of very specific and flexible user interfaces for guideline acquisition. This was for example not possible in the previous Windows version of Protégé [21], where the user interface was not as flexible as the one, described here. This has been recognized by the developers of Protégé, as the recent Java version supports the use of plug-ins to create flexible task-specific user interfaces [51]. However, this version was not available during the time this framework was

developed (a re-implementation of the Gaston framework in Protégé will require a complete reprogramming of the framework in Java).

In the literature, the process of guideline development is often focused on the issues of guideline representation and acquisition (stages 1-3 in the Gaston framework). However, an execution-time decision-support system that can be automatically created from acquired guidelines (stage 4) is often underestimated but of crucial importance. Developing the execution-time system with the requirements of execution speed and compactness in mind [31] resulted in a fast real-time system.

Concerning future developments, the model still needs improvements in a number of areas. First, the description of a PSM as well as a guideline on a high level is still informal. Consequently, there is no automatic mapping from a task description to the control structure of the PSM. We are currently investigating the integration of formal PSM-languages [48] within our framework to make such mappings explicit. Also, the primitives developed in our model do not represent uncertainty that is sometimes required by guidelines. Finally, the representation of temporal information is also still under development. Currently, our system representation is for example not as expressive in this area as for example the representation, described in the Asbru [13] language.

References

1. Grimshaw JM, Russel IT. Effects of Clinical Guidelines on Medical Practice: A Systematic Review of Rigorous Evaluation. *Lancet* 1993;342:1317-22.
2. Effective Health Care. Implementing Clinical Practice Guidelines: Can guidelines be used to improve clinical practice? *Effective Health Care* 1994;8:1-12.
3. Field MJ, Lohr KN. Guidelines for Clinical Practice: From Development to Use. Washington, DC.: National Academy Press, 1992.
4. Van Der Lei J, Talmon JL. Clinical Decision-Support Systems. In: Van Bommel and Musen (eds). *Handbook of medical informatics*. Houten: Bohn Stafleu Van Loghum, 1997.
5. Johnston ME, Langton KB, Haynes RB, Mathieu A. Effects of computer-based clinical decision support systems on clinician performance and patient outcome. A critical appraisal of research. *Ann Intern Med* 1994;120(2):135-42.
6. Hripcsak G. Rationale for the Arden Syntax. *Comput Biomed Res* 1994;27(4):291-324.
7. Fox J, Johns N, Rahmzadeh A. Disseminating medical knowledge: the *PROforma* approach. *Artif Intell Med* 1998;14:157-81.
8. Ohno-Machado L, Gennari JH, Murphy SN, Jain NL, Tu SW, Oliver DE, Pattison-Gordon E, Greenes RA, Shortliffe EH, Barnett GO. The guideline interchange format: a model for representing guidelines. *JAMIA* 1998;5(4):357-72.
9. Miller PL. *Expert Critiquing Systems, Practice-Based Medical Consultation by Computer*. New York: Springer-Verlag, 1986.
10. Hripcsak G, Clayton PD, Jenders RA, Cimino JJ, Johnson SB. Design of a clinical event monitor. *Comput Biomed Res* 1996;29(3):194-221.

11. Van der Lei J, Musen MA. A model for critiquing based on automated medical records. *Comput Biomed Res* 1991;24:344-78.
12. Musen MA, Tu SW, Das A, Shahar Y. EON: A Component-Based Approach to Automation of Protocol-Directed Therapy. *JAMIA* 1996;3:367-88.
13. Shahar Y, Miksch S, Johnson P. The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines. *Artif Intell Med* 1998;14:29-51.
14. Fridsma DB, Gennari JH, Musen MA. Making Generic Guidelines Site-Specific. *Proc AMIA Symp* 1996;:597-601.
15. Miller RA. Strategies for Medical Knowledge Acquisition. In: Van Bommel and Musen (eds). *Handbook of medical informatics*. Houten: Bohn Stafleu Van Loghum, 1997.
16. Eriksson H, Musen MA. Metatools for knowledge acquisition. *IEEE Software* 1993;10(3):23-9.
17. Gill H, Ludwigs U, Matell G, Rudowski R, Shahsavari N, Ström C, Wigertz O. Integrating knowledge-based technology into computer aided ventilation systems. *Int J Clin Monit Comput* 1990;7(1):1-6.
18. Laffey TJ, Cox PA, Schmidt JL, Kao SM, Read JY. Real-Time Knowledge-Based Systems. *AI Magazine* 1988;9;1:27-45.
19. Schreiber AT, Wielinga BJ, De Hoog R, Akkermans H, van der Velde W, Anjewierden A. *CommankADS: A Comprehensive Methodology for KBS Development*. *IEEE Expert* 1994;:28-37.
20. Motta E. *Reusable Components for Knowledge Modelling. Case Studies in Parametric Design Problem Solving*. Amsterdam: IOS Press, 1999.
21. Musen MA, Gennari JH, Eriksson H, Tu SW, Puerta AR. *PROTEGE II: Computer Support For Development Of Intelligent Systems From Libraries Of Components*. *Medinfo* 1995;8(1):766-70.
22. Musen MA. Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. *Proc AMIA Symp* 1998;46-52.
23. Clancey WJ. Heuristic classification. *Artificial Intelligence* 1985;27(3):289-350.
24. De Clercq PA, Blom JA, Hasman A, Korsten HHM. The Application of Ontologies and Problem Solving Methods for the Development of Shareable Guidelines. *Artif Intell Med* 2001;22(1):1-22.
25. Gruber TR. A translation approach to portable ontologies. *Knowledge Acquisition* 1993;5(2):199-220.
26. Studer R, Eriksson H, Gennari J, Tu SW, Fensel D, Musen MA. Ontologies and the Configuration of Problem-solving Methods. *Proceedings of the 10th Knowledge Acquisition for Knowledgebased Systems Workshop, Banff, 1996*.
27. Chaudhri V, Farquhar A, Fikes R, Karp P, Rice J. The Generic Frame Protocol 2.0. Available at <http://www.ai.sri.com/~gfp/spec.html>.
28. Noy NF, Ferguson RW, Musen MA. The knowledge model of Protégé-2000: combining interoperability and flexibility. *2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, . 2000.
29. Giarratano J, Riley G. *Expert Systems: Principles and Programming*. Boston: PSW Publishing Company, 1994.
30. World Wide Web Consortium, *Resource Description Framework (RDF)*. Available at <http://www.w3.org/RDF/>
31. Blom JA. Temporal logics and real time expert systems. *Comput Methods Programs Biomed* 1996;51:35-49.

32. National High Blood Pressure Education Program. The Sixth Report of the Joint National Committee on Detection, Evaluation, and Treatment of High Blood Pressure. Washington: NIH; 1998.
33. The Medical Guideline Technology project. INCO-COPERNICUS Project IC15 CT 98-0315. Homepage available at <http://frost.open.ac.uk/mgt/>.
34. Musen MA. Decision-support methods. In: Van Bommel and Musen (eds). Handbook of medical informatics. Houten: Bohn Stafleu Van Loghum, 1997.
35. Kiczales G, des Rivieres J, Bobrow DG. The Art of the Metaobject Protocol. Cambridge, MA: The MIT Press, 1991.
36. Tu SW, Musen MA. A flexible approach to guideline modeling. Proc AMIA Symp 1999;:420-4.
37. Chandrasekaran B, Johnson TR, Smith JW. Task-Structure Analysis for Knowledge Modeling. Communications of the ACM 1992;35(9):124-37.
38. De Clercq PA, Blom JA, Hasman A, Korsten HHM, A strategy for development of practice guidelines for the ICU using automated knowledge acquisition techniques. Int J Clin Monit Comput 1999;15:109-17.
39. Bindels R, De Clercq PA, Winkens RAG, Hasman A. A test ordering system with automated reminders for primary care based on practice guidelines. Int J Med Inf 2000;58-59(1):219-33.
40. The TANDEM project. Leonardo da Vinci Program DK/97/2/00376/PI/II.1.1.c/FPC. Homepage available at <http://www.tandem.v-chi.dk/>.
41. McDonald CJ, Hui SL, Smith DM, Tierney WM, Cohen SJ, Weinberger M. Reminders to physicians from an introspective computer medical record. A two-year randomized trial. Ann Intern Med 1984;100:130-8.
42. Miller RA, Masarie FE Jr. The demise of the "Greek Oracle" model for medical diagnostic systems. Meth inform Med 1990;29:1-2.
43. De Clercq PA, Hasman A. Design of a Consumer Health Record for Supporting the Patient-centered Management of Chronic Diseases. Medinfo 2001;10(2):1445-9.
44. Project IMPACT Page. SSCM Web Page. Available at http://www.sccm.org/impact/impact_home_set.html.
45. Bindels R, Hasman A, Winkens RAG, Pop P, Van Wersch JWW. Validation of a knowledge based reminder system for diagnostic test ordering in family medicine. Int J Med Inf 2001;64(2-3):341-54.
46. Van Hyfte DMH, De Clercq PA, Tjandra-Maga TB, Zitman FG, De Vries Robbé PF. Modelling the psychoactive drug selection application domain at the knowledge level 1999. Proc Belgium-Netherlands Conf on Artificial Intelligence 1999;:187-8.
47. Rector A, Solomon W, Nowlan W, Rush T, Zanstra P, Claassen W. A Terminology Server for medical language and medical information systems. Meth Inform Med 1995;34:147-57.
48. Fensel D, Benjamins VR, Motta E, Wielinga B. UPML: A Framework for knowledge system reuse 1999. Proceedings of the International Joint Conference on AI (IJCAI '99) 1999.
49. Gil Y, Melz E. Explicit representations of problem-solving strategies to support knowledge acquisition. Proc Thirteenth National Conference on Artificial Intelligence 1996.
50. Woods W. What's in a link: foundations for semantic networks. In: Bobrow and Collins (eds). Representation and understanding: studies in cognitive science. London: Academic Press, 1975.
51. Grosso WE, Eriksson H, Fergerson RW, Gennari JH, Tu SW, Musen MA. Knowledge Modeling at the Millennium (The Design and Evolution of Protégé-2000). Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), Banff, Canada, October 1999.

CHAPTER 5

A STRATEGY FOR DEVELOPMENT OF PRACTICE GUIDELINES FOR THE ICU USING AUTOMATED KNOWLEDGE ACQUISITION TECHNIQUES

Published in:
The International Journal of Clinical Monitoring and Computing 1999;15:109-117

Paul A. de Clercq
Johannes A. Blom
Arie Hasman
Hendrikus H.M. Korsten

1 Introduction

Practice guidelines are increasingly used in health care, especially in clinical care and emergency care environments such as the Intensive Care Unit (ICU), to improve the quality of care [1]. Reminder systems may use these guidelines to provide decision support to the ICU health care workers [2]. Traditionally, knowledge engineers acquired the guidelines from a domain expert (physician) by means of various knowledge elicitation techniques, such as interview methods or card sorting [3]. This process usually creates a severe bottleneck, however, because the domain expert and the knowledge engineer have to reach a common understanding before progress can be made [4] and even then the production rate of knowledge is very low. Therefore, automated knowledge acquisition tools such as knowledge base editors are increasingly used to acquire knowledge directly from domain experts [5].

This paper demonstrates how the development of practice guidelines with automated knowledge acquisition techniques can improve the quality of computer support in an ICU. To validate our ideas, we created and installed a knowledge acquisition environment in the ICU of the Catharina Hospital, Eindhoven, the Netherlands. In this environment, physicians are able to formulate, update and evaluate guidelines without the assistance of a knowledge engineer.

2 Materials and methods

2.1 *The knowledge acquisition environment*

The knowledge acquisition environment consists of a collection of tools to implement and evaluate practice guidelines:

- A graphical knowledge acquisition tool, called the CritICIS knowledge base editor, is used to formulate and update practice guidelines. The knowledge base editor is implemented by means of the Borland Delphi graphical authoring environment and runs under the Microsoft Windows operating system.
- A Patient Data Management System (PDMS) collects, stores and manages clinical data as well as physiological data in an ICU [6]. The PDMS in our ICU, called the Intensive Care Information System or ICIS¹, has been in operation for more than two years now and has made the use

¹ICIS is manufactured by INAD Medical Systems BV, POB 178, 5600 AD, Eindhoven, the Netherlands

of paper records superfluous. ICIS was developed with Microsoft Access and runs under Microsoft Windows.

- An expert system, called CritICIS, reports inconsistencies between the clinical data stored in the PDMS and the implemented guidelines that were entered using the knowledge base editor. CritICIS is implemented as a reminder system. It monitors the actions and observations of physicians and nurses and generates a reminder whenever a guideline is not followed [7]. Decision support systems that operate in dynamic and unstable clinical domains such as the ICU should be real-time systems, because, even though the volume of data in such an environment can be overwhelming [8, 9], the system's response must be timely. We developed the reminder system by means of the Gaston architecture, a collection of tools that assist in the design of real-time expert systems [10]. The Gaston architecture contains the SIMPLEXYS verification toolbox: a variety of tools to perform logical and semantic tests on the implemented guidelines [11].

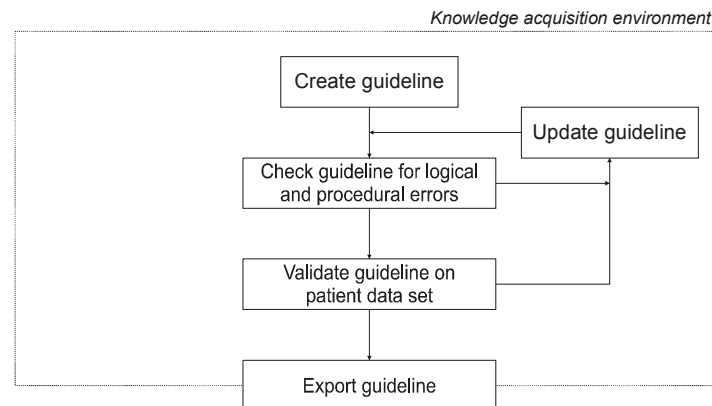


Figure 1: Formulating and evaluating practice guidelines in the knowledge acquisition environment

Practice guidelines are formulated, updated and transferred to the knowledge base of the reminder system with the help of the knowledge base editor. Subsequently, the guidelines are validated by testing the guidelines on a data set of previously admitted patients. A panel of ICU physicians then evaluates the output of the reminder system. Based on this evaluation, the guidelines may be updated (or discarded). When the ICU physicians approve a guideline, it is exported to the system that is used in daily practice (figure 1).

2.2 Creating and updating guidelines

The CritICIS knowledge base editor is a graphical knowledge acquisition tool that enables physicians to formulate guidelines and transfer them to the

reminder system's knowledge base. This process of representing knowledge has several distinct aspects [12], shown in the next sections.

2.2.1 Lexicon / Ontology

In order to be able to reason about a medical domain, its terms or objects (e.g., diseases, drugs and diagnoses) must be established, classes of objects must be defined (e.g., classes of diseases, classes of drugs, etc.) as well as relationships between the various terms and classes (e.g., pneumonia is a member of the class of lung diseases; the class of lung diseases in turn is part of the diseases class). This structure is referred to as an ontology: a formal description of objects in the world, those objects' properties, and the relations among them [13].

We have used the IMPACT Minimal Standard Data Set, a set of medical terms, describing the state of a patient in an Intensive Care Unit [14] as an ontology for our ICU. It defines a number of classes, each describing a distinct medical category such as drugs, treatments and diagnoses. Every category may contain subcategories. In the knowledge base editor the ontology is implemented as a class tree and inheritance is used to create hierarchical relationships. Every object in the tree is defined by a number of properties, some of which are mandatory (e.g., the object's name and code ID) whereas others depend on the object's representation in the real world. For example, an object that represents a treatment has an additional property that holds the treatment's duration, whereas an object that represents a drug has properties that store the drug's dose, frequency and unit. At present, the ontology consists of more than 1900 objects divided in about 100 classes.

2.2.2 Inference Syntax

Guidelines are also represented as objects. Every guideline object encapsulates a production rule (using the syntax '*IF expression THEN GiveReminder*') and is defined by the following properties:

- *Name*: the name of the guideline.
- *Author*: the name of the guideline's author.
- *Type*: the guideline's type. The knowledge base editor defines two types: Reminder and Intermediate. Guidelines of type Reminder issue a reminder when a guideline is not followed, whereas guidelines of type Intermediate are parts of other guidelines; they do not lead to a reminder but store intermediate observations or conclusions.
- *Validation*: this binary property indicates whether a guideline is approved for routine clinical care (Production) or is still in the test phase (Test).

- *Explanation*: the *explanation* property holds an explanation of the guideline, for example used for documentation purposes. It may also contain literature references.
- *Message*: this property holds the message that is to be given by the reminder system when the guideline is not followed (only required for guidelines of type Reminder).
- *Expression*: the expression part of the encapsulated production rule. If the reminder system evaluates an expression as true, a reminder will be issued. The property is stored as references to instantiated objects of the ontology class tree, combined with Boolean operators (AND, OR and NOT).

The user interface of the knowledge base editor presents the ontology as well as the guidelines in the knowledge base to the user. It enables physicians to define a guideline by entering its property values (figures 2 and 3).

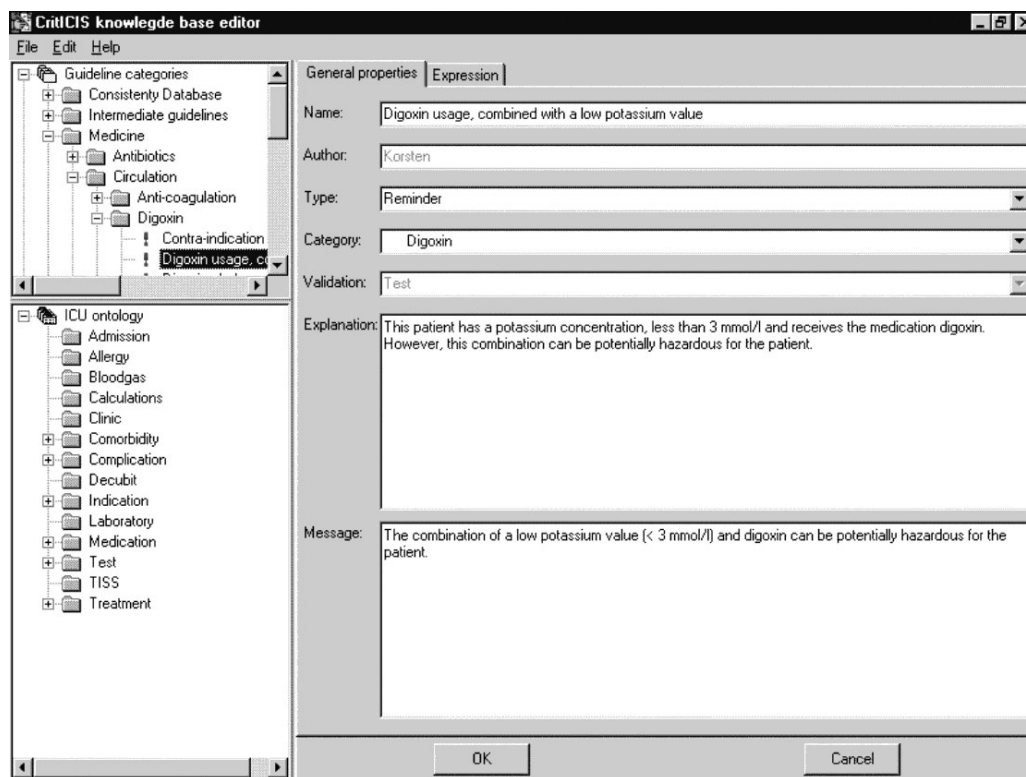


Figure 2: The user interface of the CritICIS knowledge base editor. The upper left window presents the knowledge base with all currently implemented guidelines. The lower left window shows the ontology class tree. The right window presents a detailed description of a single guideline by means of its property values

The user interface of the knowledge base editor is divided into three sections or windows. The upper left window presents the knowledge base with all

currently implemented guidelines. The lower left window shows the ontology class tree. The window on the right presents a detailed description of a single guideline by means of its property values.

Guidelines are grouped into categories according to the guideline's purpose. Guidelines about drug contraindications, for example, are stored in the contraindications category. Physicians may store guidelines into existing categories or create a new category to hold the guideline. Whenever a physician creates or selects a guideline, the right window presents the guideline's properties divided into two pages, called the 'general properties' page and the 'expression' page. The '*Digoxin usage, combined with a low potassium value*' guideline is an example; it indicates that a patient with a potassium blood concentration less than 3 mmol/l should not be prescribed the drug Digoxin. The right window in figure 2 shows the general properties of this guideline such as name, author, type, etc.

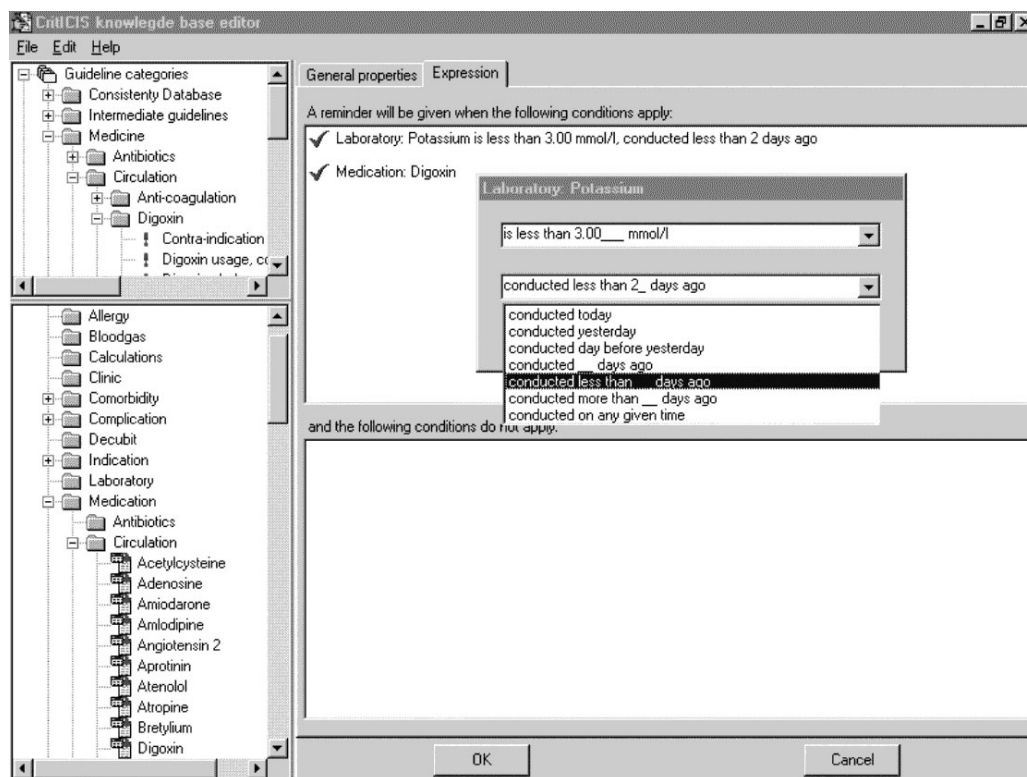


Figure 3: A guideline's expression. The Potassium pop-up window shows the properties of the potassium laboratory test

The 'expression' page presents the content of the *expression* property. For example, figure 3 shows the expression of the '*Digoxin usage, combined with a low potassium value*' guideline. This particular expression evaluates to true if a laboratory test, carried out less than 2 days ago, returns a potassium

concentration lower than 3 mmol/l and if the medication Digoxin has been prescribed as well. As a result, a reminder will be issued. The expression consists of instantiated objects from the ontology class tree, which are selected from the tree and dragged to the right window (this operation implements the AND-operator). It is also possible to edit a term's properties, which depend on its representation in the real world. Finally, propositions that consist of multiple terms can be created by dragging a term from the ontology tree on top of an existing proposition in the right window (this implements the OR-operator).

2.3 Validating implemented guidelines

After their creation or update, guidelines are tested in various ways. First, the guidelines are translated into a semantic net, after which the correctness of the net is verified. These tests include the detection of various types of logical errors such as incompleteness, inconsistencies, conflicts and (partial) tautologies, as well as the detection of procedural errors, such as infinite loops (e.g., self-references) [11].

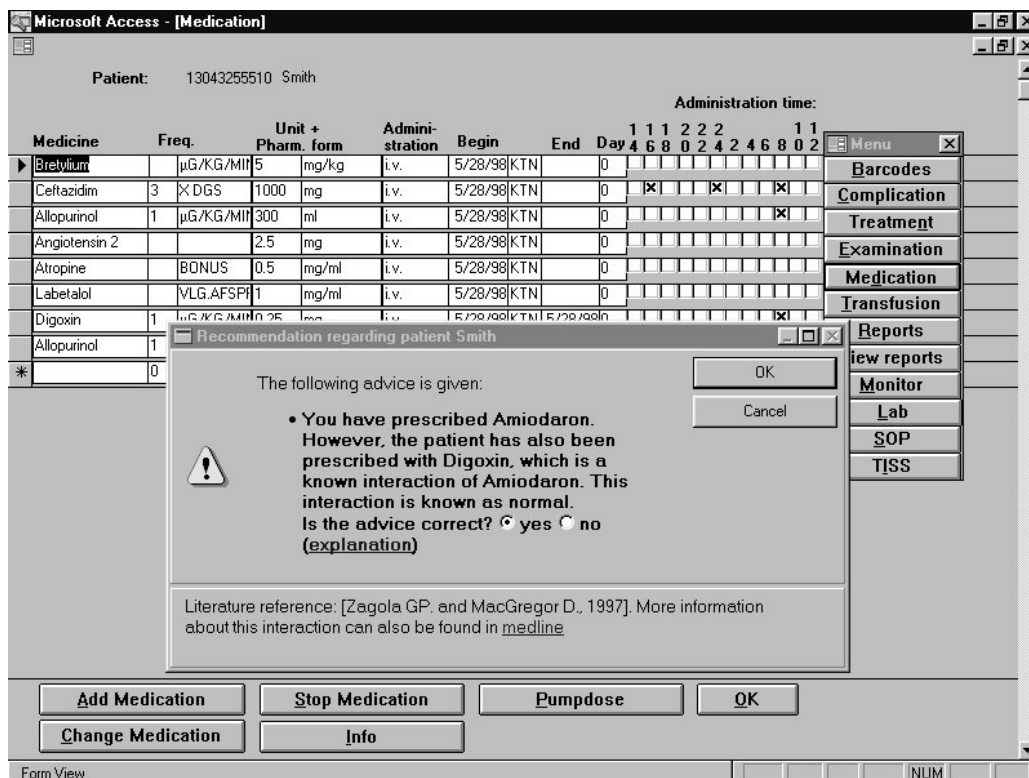


Figure 4: If a guideline is not followed, CritICIS generates and displays a reminder that overlays the normal user interface of the PDMS

Next, the new guidelines are tested on a large data set of previously admitted ICU patients. The physician activates the reminder system through the PDMS,

now pointing to the database of previously admitted ICU patients. The reminder system checks for inconsistencies between the implemented guidelines and these patient data. If an inconsistency is found, the corresponding reminder is generated and displayed by means of a pop-up window which overlays the normal user interface of the PDMS (figure 4).

The pop-up window asks the physician, who now may need to examine the corresponding patient data, to classify the reminder as 'correct' (in case the reminder was appropriate) or 'incorrect' (in case it was inappropriate). If it is not possible to classify a reminder, it is labeled 'inconclusive'. The physician may additionally provide a rationale to explain his or her evaluation of the reminder. All evaluations are stored into a database and presented as histograms (figure 5), in which each column depicts the quality of a single guideline by showing the number of times the corresponding reminder was evaluated as correct, incorrect or inconclusive.

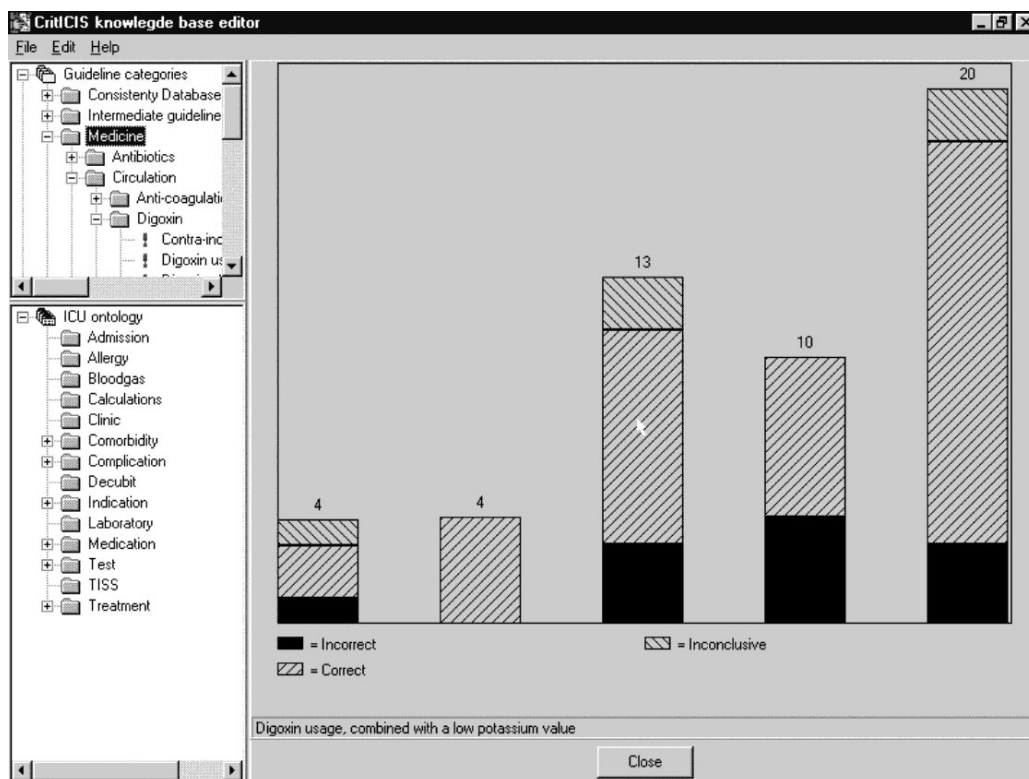


Figure 5: Each histogram shows the performance of one guideline. The title of every guideline is shown in the bottom part of the screen, whenever the user points to the corresponding histogram. An example is the column in the middle, which shows the quality of the 'Digoxin usage, combined with a low potassium value' guideline. The corresponding reminder has been generated 13 times, of which three were classified as incorrect, eight as correct and two as inconclusive

2.4 Exporting the guidelines

Guidelines are stored into a knowledge base, which is maintained by the knowledge base editor. When instructed, the editor is able to export the content of the knowledge base in a number of formats. Normally, the knowledge base is exported into the format that is used by the reminder system. It is also possible to export the guidelines in different formats for other systems. The knowledge base editor can be used for writing Medical Logic Modules (MLMs) in the Arden Syntax, for example, a language designed as an open standard for the procedural representation and sharing of medical knowledge [15].

In parallel to the development of the knowledge base editor, the reminder system is being integrated into the standard PDMS that operates in the ICU. When all the ICU's physicians have approved a guideline, it is transferred to the reminder system and integrated into the PDMS. From then on, it provides decision support to the health care workers of the ICU. In this environment, the operation of the reminder system is similar to the operation of the system in the knowledge acquisition environment, with the difference that now the reminder system contains only validated guidelines and operates in real-time.

3 Results

Physicians as well as the nursing staff have accepted the knowledge editor and the reminder system very well. After a short training period, physicians find the editor's interface useful and sufficiently 'intuitive'. The content of the guidelines varies from relatively simple, such as 'a patient, admitted outside normal working hours, is usually marked as an emergency' to more complex, such as the detection of drug contraindications and side effects or warnings for the absence of certain monitoring requirements. Initially, physicians on an ad-hoc basis implemented mainly local evidence-based guidelines. However, in the course of time, physicians developed guidelines in a more structured manner, based on literature and common consensus. Also, newly entered guidelines were often variants of and inspired by previously entered guidelines, e.g., a new contraindication or a new check for data entry completeness.

At present, the physicians have entered 58 different guidelines into the reminder system's knowledge base by means of the knowledge base editor. In order to determine the validity of these initially entered guidelines, we tested the reminder system with the 58 guidelines on a patient data set of 803 previously admitted patients. As a result, 27 guidelines fired at least once, generating 406 reminders in total. The 31 guidelines that never fired were not

further analyzed; a majority, however, is about exceptional situations that may be life threatening but that occur only very infrequently. In order to estimate the proportion of false-negatives (usually guidelines that are too specific), we are currently developing an annotated and validated patient database. Using this database, physicians will be able to estimate the number of false-negatives by selecting patients in this database and checking whether a reminder should have been issued.

Table 1 shows the validation results of the 27 guidelines that fired. We have classified the corresponding reminders in 5 categories, based on their causes of firing:

- *Incorrect guideline.* Reminders of this type are false alarms, given as a result of an incorrectly implemented guideline. The corresponding guideline is then removed from the knowledge base. Reminders of this category were not found during the validation process.
- *Too generic guideline.* Reminders in this category also represent false alarms, given as a result of a guideline that was not specific enough. The corresponding guideline needs to be updated.
- *Inconsistency database.* Reminders of this type represent situations in which data in the database of the PDMS is incomplete or inconsistent. These data need to be corrected.
- *Inappropriate action.* Reminders in this category are issued whenever actions or decisions of the ICU staff may not be the most appropriate ones.
- *Potential risk.* Reminders of this type represent situations that involve a potential risk.

As shown in table 1, from the 406 issued reminders, 356 (88%) were classified as correct and 50 (12%) were false alarms. The 50 false alarms were issued by five guidelines that were too generic. An example is the top guideline, stating that a patient with a subarachnoid bleeding must be treated with a laxative. It was issued 8 times, of which 7 were false alarms and 1 was correct. The false alarms were due to the fact that the guideline did not include certain exceptions (e.g., a patient with a traumatic head injury is not treated with a laxative). Another example is the 'The complication oliguria not diagnosed' guideline that embodies the notion that patients with a low urine output suffer from oliguria. This guideline generated 60 reminders, of which 49 were given correctly and 11 were false alarms. The 11 false alarms all concerned patients that had already been diagnosed with oliguria before their admission to the ICU (diagnoses entered before admission to the ICU are stored into another database table that erroneously was not specified for inspection).

Name guideline	Too generic guideline	Inconsistency database	Inappropriate action	Potential risk
A subarachnoid bleeding must be treated with a laxative	7		1	
No anti-coagulation prescribed	9	38	17	
Defibrillation not recorded		6		
Digoxin may increase rhythm disturbances				1
The complication oliguria not diagnosed	11	49		
The complication anuria not diagnosed		29		
A patient, admitted outside normal working hours, is usually an emergency		37		
A patient treated with renal-substitution therapy must be weighted daily	5	1	15	
Serum-levels of the antibiotic must be checked	18	1	20	
Digoxine may impair renal function				15
Diuretics increase digoxin-toxicity				25
Serum levels of digoxin increase by using verapamil or amiodaron				10
Due to hypotension, adjust the starting-dose of ACE-inhibition				46
Due to blunt abdominal trauma, check serum amylase			7	
Due to blunt thorax-trauma, make an Echocardiogram		1	7	
This patient should be treated with coumarin		1	2	
(Pre-)ecclampsia: Mg-supplement with a low Ca++ increases its toxicity.				1
Ca++ must be supplemented				1
NSAID's counteract the anti-hypertensive medication				7
You may consider giving magnesium in the case of a (pre-)ecclampsia				2
Half the dose of digoxin B.W. < 70kg, age >70 or creatine clearance <70 ml/min				8
Digoxin usage, combined with a low potassium value				2
Start digoxin according to protocol for pneumonectomy-patients			1	
Toxicity of Digoxin increased by the use of amphi-B				2
Recently bradycardia: relative contraindication for beta-blockade				1
75% standard dose EXTRA after dialysis				1
75% standard dose EXTRA after dialysis				1

Table 1: Performance of the 27 guidelines that fired

In all these five cases, the guidelines were updated to incorporate the exceptions, resulting in the elimination of all false alarms.

4 Discussion

Various studies, covering a wide range of clinical settings and tasks, conclude that the use of practice guidelines significantly improves the quality of care [16], especially when used in combination with computer-stored medical record systems such as a PDMS [17]. However, the process of guideline development is usually very time and resource consuming. There is a wide variety of variables and the rules are often physician- or ICU-specific [18]. We believe that automated knowledge acquisition tools such as knowledge base editors are able to facilitate the guideline development process -- as long as the physicians are willing to use them. Our knowledge base editor was therefore developed in close collaboration with and as specified by the physicians and nursing staff of our ICU. In order to enable sharing of the guidelines with other ICU departments, the guidelines are exportable, e.g. using the Arden syntax format [19].

An important issue is the attitude of the physicians and the nursing staff towards the use of decision support systems [20]. During the last two decades, the majority of expert consultation systems utilized the 'Greek Oracle' approach. These systems, such as INTERNIST-I [21], typically expect a health care worker to enter information about a patient, after which the system produces a number of conclusions and recommendations such as a list of possible diseases, a set of suggested tests or a treatment plan. Experience revealed, however, that the system's users could become annoyed by this approach, because the user's role often is diminished to that of a passive observer or even a 'slave' of the computer. Reminder systems, on the other hand, utilize the so-called critiquing approach. Critiquing systems are silent whenever the computer judges the user's (planned) behavior to be satisfactory given the case data, but offer a critique of the behavior should the user (propose to) take an action that is not consistent with the system's knowledge base [4]. As a result, a critiquing system structures its advice around the physician's own thinking and style of practice instead of 'trying to tell a physician what to do' [22]. Also, practice guidelines are well suited for implementation in a critiquing system, because a critiquing system notifies a user whenever there are inconsistencies between implemented guidelines and the treatment plans that were proposed by that same user or by the group in which he or she participates. During the last 15 years there have been several attempts to integrate critiquing systems with existing data-management systems, usually applied to a narrow problem domain such as

cancer treatment, hospital-acquired infections and antibiotic use, blood ordering and hypertension [23-26].

Computer-based critiquing may also have its limitations [27]. A critiquing system that only relies on medical records for its input may produce irrelevant critiques because a medical record entered by a physician usually only contains the *actions* of that physician; the underlying reasoning or rationale has to be reconstructed. For example, a physician may be well aware of violating a guideline but, since no guideline can cover *all* cases, other circumstances may have led the physician to decide otherwise. Also, the data in the patient database may be incomplete. In order to improve the performance of computer-based critiquing, the format of computer-based medical records must be further developed. In particular, medical devices (e.g., monitors, ventilators and infusion pumps) that automatically store their information and/or actions into the PDMS can lead to a far more complete patient database; in addition, this approach can greatly alleviate the current need for consistent and complete data entry by the ICU staff.

Another method of improving the performance of computer-based critiquing may be the utilization of the critiquing process model. According to Van der Lei and Musen [26], every computer system requires two distinct types of knowledge to review automated medical records: critiquing knowledge (knowledge about the process of critiquing itself) and medical knowledge (specific medical knowledge, required by the critiquing process). Utilizing the critiquing process model is thought to simplify system maintenance as well as the knowledge acquisition process. The knowledge base of the reminder system is currently implemented as a set of independent modules (comparable to Arden Syntax modules). However, we have started to analyze the existing guidelines in order to find a limited, useful classification of critiquing tasks. This bottom-up approach contrasts with the more usual top-down approach, which may lead to systems that are too 'theoretical' for the daily practice of the ICU with its established norms and procedures. Further research will have to determine whether (and if so, how) current and future guidelines are indeed 'bottom-up classifiable' into more abstract critiquing task classes.

5 Conclusions

This method enables physicians to define guidelines and transfer them to a decision support system that is used in daily practice. Using the validation of the guidelines on the stored patient database, physicians readily find previously not thought of exceptions, and equally readily improve the guidelines accordingly. These first results and findings convince us that this

bottom-up strategy, combined with appropriate automated knowledge acquisition tools, enables the medical specialists themselves to improve the quality of the knowledge base and, hopefully, ICU patient care without the assistance of a knowledge engineer.

References

1. Uckun S. Instantiating and monitoring skeletal treatment plans. *Methods Inf Med* 1996;35:324-33.
2. Lau F, Vincent DD. A Knowledge-Based Care Protocol System for ICU. *Medinfo* 1995;:979-83.
3. Welbank M. An overview of knowledge acquisition methods. *Interacting with computers* 1990;2(1):83-91.
4. Van Der Lei J, Talmon JL. Clinical Decision-Support Systems. In: Van Bommel and Musen (eds). *Handbook of medical informatics*. Houten: Bohn Stafleu Van Loghum, 1997.
5. Musen MA, Fagan L, Combs DM, Shortliffe EH. Use of a domain model to drive an interactive knowledge-editing tool. *Int J Man-Mach Stud* 1987;26:105-21.
6. Metnitz PGH, Lenz K. Patient data management systems in intensive care - the situation in Europe. *Intensive Care Med* 1995;21:703-15.
7. McDonald CJ, Hui SL, Smith DM, Tierney WM, Cohen SJ, Weinberger M. Reminders to physicians from an introspective computer medical record. A two-year randomized trial. *Ann Intern Med* 1984;100:130-8.
8. Gill H, Ludwigs U, Matell G, Rudowski R, Shahsavari N, Ström C, Wigertz O. Integrating knowledge-based technology into computer aided ventilation systems. *Int J Clin Monit Comput* 1990;7(1):1-6.
9. Laffey TJ, Cox PA, Schmidt JL, Kao SM, Read JY. Real-Time Knowledge-Based Systems. *AI Magazine* 1988;9(1):27-45.
10. De Clercq PA, Blom JA, Hasman A, Korsten HHM. Design and implementation of a framework to support the development of clinical guidelines. *Int J Med Inf* 2001;64(2-3):285-318.
11. Blom JA. Temporal logics and real time expert systems. *Comput Methods Programs Biomed* 1996;51:35-49.
12. Musen MA. Dimensions of Knowledge Sharing and Reuse. *Comput Biomed Res* 1992;25:435-67.
13. Gruber TR. A translation approach to portable ontologies. *Knowledge Acquisition* 1993;5(2):199-220.
14. Project IMPACT Page. SSCM Web Page. Available at http://www.sccm.org/impact/impact_home_set.html.
15. Jenders RA, Dasgupta B. Assessment of a knowledge-acquisition tool for writing Medical Logic Modules in the Arden Syntax. *Proc AMIA Symp* 1996;:567-71.
16. Effective Health Care. Implementing Clinical Practice Guidelines: Can guidelines be used to improve clinical practice? *Effective Health Care* 1994;8:1-12.
17. East TD, Morris AH, Wallace CJ, Clemmer TP, Orme JF Jr, Weaver LK, Henderson S, Sittig DF. A strategy for development of computerized critical care decision support systems. *Int J Clin Monit Comput* 1991;8(4):263-9.
18. East TD, Henderson S, Pace NL, Morris AH, Brunner JX. Knowledge engineering using retrospective review of data: a useful technique or merely data dredging? *Int J Clin Monit Comput* 1991;8(4):259-62.
19. Hripcsak G. Rationale for the Arden Syntax. *Comput Biomed Res* 1994;27(4):291-324.

20. Shortliffe EH. Testing Reality: The Introduction of Decision-Support Technologies for Physicians. *Meth Inform Med* 1989;28:1-5.
21. Miller RA, Masarie FE Jr. The demise of the "Greek Oracle" model for medical diagnostic systems. *Meth inform Med* 1990;29:1-2.
22. Miller PL. *Expert Critiquing Systems, Practice-Based Medical Consultation by Computer*. New York: Springer-Verlag, 1986.
23. Langlotz CP, Shortliffe EH. Adapting a consultation system to critique user plans. *Int J Man-Mach Stud* 1983;19:479-96.
24. Evans RS, Larsen RA, Burke JP, Gardner RM, Meier FA, Jacobson JA, Conti MT, Jacobson JT, Hulse RK. Computer surveillance of hospital-acquired infections and antibiotic use. *JAMA* 1986;1007-11.
25. Lepage EF, Gardner RM, Laub RM, Jacobson JT. Assessing the effectiveness of a computerized blood order "consultation" system. *Proc Annu Symp Comput Appl Med Care* 1991;:33-7.
26. Van der Lei J, Musen MA. A model for critiquing based on automated medical records. *Comput Biomed Res* 1991;24:344-78.
27. Van der Lei J, Van der Heijden P, Boon WM. Critiquing expert critiques. Issues for the development of computer-based monitoring in primary care. *Medinfo* 1989;:106-10.

CHAPTER 6

EXPERIENCES WITH THE DEVELOPMENT, IMPLEMENTATION AND EVALUATION OF AUTOMATED DECISION SUPPORT SYSTEMS

Based on papers published in:
The International Journal of Medical Informatics 2000;58-59(1):219-233
Medical Informatics and the Internet in Medicine 2000;25(4):247-263
The Journal of the European College of Neuropsychopharmacology 2000;10(3):390
Medical Informatics and the Internet in Medicine. Accepted for publication

Paul A. de Clercq
Rianne Bindels
Dirk M.H. van Hyfte
Hendrikus H.M. Korsten
Johannes A. Blom
Bruce H.R. Wolffenbuttel
Arie Hasman

1 Introduction

The number of guideline-based decision support systems that are aimed at improving the quality of care is rapidly increasing. There have been numerous efforts to develop systems that support guideline-based care in an automated fashion, covering a wide range of clinical settings and tasks [1]. However, building systems that are both effective in supporting clinicians and accepted by them has proven to be a difficult task. Of the systems that were evaluated by a controlled trial, the majority showed impact [2]. In order to be successful, attention must be paid to various areas that are important in the guideline development process such as guideline representation, acquisition, verification and execution [3].

This paper reports experiences concerning the development, implementation and evaluation of guideline-based systems that were created with the Gaston approach: a methodology and framework that facilitates all stages in the guideline development process, ranging from the definition of models that represent guidelines to the implementation of run-time systems that provide decision support, based on the guidelines that were developed during the previous stages [4]. The framework consists of 1) a guideline representation formalism that uses the concepts of primitives, Problem-Solving Methods (PSMs) and ontologies [5] to represent guidelines of various complexity and granularity and different application domains, 2) a guideline authoring environment that enables guideline authors to define guidelines, based on the newly developed guideline representation formalism, and 3) a guideline execution environment that translates defined guidelines into a more efficient symbol-level representation, which can be read in and processed by an execution-time engine that forms the Decision Support System (DSS).

The aim of this paper is to examine the possibilities to develop and implement guideline-based decision support systems for use in different domains through the Gaston approach. Related to this are questions that are commonly addressed when developing and implementing computer-based guidelines such as ‘how to represent different sorts of guidelines in a straightforward manner’, ‘how to facilitate guideline authors during the acquisition process’, ‘how to map concepts from a guideline to corresponding concepts in the real world’ and ‘how to support care providers in daily practice using guideline-based decision support systems’.

The remaining part of this paper addresses these questions by describing and discussing four systems that were developed for use in the specialties of family practice, critical care, psychiatry and chronic disease management: 1)

GRIF: a reminder system that provides automated feedback on test ordering in general practice [6], 2) CritICIS: a real-time critiquing system used in critical care environments such as Intensive Care Units [7], 3) M-PADS: a psychopharmacological advisory system that provides decision support on the process of selecting the most suited psychoactive drug [8], and 4) a consumer health record system for managing chronic diseases [9]. Each system has its own focus points. The GRIF system for example focuses on the evaluation of the system in daily practice and the behavior of its users (family practitioners). The M-PADS system focuses mainly on the guideline representation part and not so much on guideline verification or execution. The CritICIS system, however, does focus on these aspects although it also focuses on the other two (guideline representation and acquisition). Finally, the system for chronic disease management focuses on how such a system can improve the communication between patients and care providers.

The next four sections describe each system in more detail. The paper ends with a general discussion on the use of these systems and their relation with the Gaston framework.

2 Automated feedback on test ordering in general practice

2.1 Introduction

The consumption of diagnostic tests has increased over the past 20 years and there is a growing awareness that a relatively large percentage of test requests in health care are inappropriate.

To manage test consumption in the Maastricht region, the Transmural & Diagnostic Center has given personal feedback in the form of written reports to Family Practitioners (FPs) in the Maastricht region since 1985. Twice a year, each FP in the Maastricht region (± 85 FPs) receives a structured feedback report with critical comments on his/her test requests in a previous month. The individual biannual written feedback is based on a comparison of request forms (including provided administrative patient data, clinical patient data and requested diagnostic tests) with accepted national or regional guidelines.

Although the individual written feedback provided by the diagnostic center in Maastricht improved the quality of the test-ordering behavior of the FPs [10] and was appreciated by FPs in the Maastricht region, a more direct (related to each test order) and less laborious method of feedback was desired.

Therefore, it was suggested to develop an automated feedback system that would directly assist in the test ordering process.

To manage test consumption and to improve appropriateness of requests an automated feedback system was developed to change the FP test ordering behavior. The main aim of this study is to develop and evaluate an accurate and reliable automated feedback system to produce immediate advice about diagnostic test ordering of FPs.

2.2 Requirements

In daily practice the FP has about 8 to 10 minutes for the consultation of one patient. For this reason the automated feedback system's most important requirement is to work fast. The system must be easy to use and should not force FPs to enter data twice. Another important requirement is that the FP receives the feedback at the time the tests are requested. Finally, we wanted to limit the entry of free text (when entering the working hypotheses, existing problems and complaints) because an automated feedback system is better able to interpret standardized medical data.

The structured manual feedback reports contain several items: number of tests requested (compared to the previous analysis and compared to the average number of requests of their colleagues), rationality of the requests based on provided patient information, discussion of incorrect or redundant requests, a number of questions concerning the policy of the FP after receiving the results of the tests and a request to answer questions posed in the letter and to comment on the feedback. Our automated feedback system covers only a part of these items. It provides feedback concerning incorrect or redundant requests and indicates when insufficient patient information is given.

2.3 Methods

2.3.1 Environment

The GRIF¹ automated feedback system consists of five parts: a knowledge base, an order entry system, a module that provides reactive support (i.e. the advice), a module that provides passive support and a database [6]. Figure 1 shows an overview of all parts.

The order entry system

The FP will have to enter the data that (s)he thinks are sufficient to support the test request. The order entry system is not yet fully integrated with the Electronic Patient Record (EPR) of the FPs but has a real-time connection with the EPR via an intermediate database. At the moment the FP wants to

¹ GRIF is the Dutch acronym for "Geautomatiseerde Reminders als Interactieve Feedback" (Automated Reminders as Interactive Feedback).

request diagnostic tests, (s) he can switch from the EPR to the automated feedback system and the necessary patient related data (name, address, date of birth, gender, medication and existing problems) will be transferred from the EPR database to an intermediate database. At the time the FP starts the order entry system the data are transferred automatically from the intermediate database into the corresponding fields of the order entry system. Request specific data are entered directly into the system. These data consist of 1) medical data: working hypotheses, signs and complaints and physical examination results and 2) reasons for test request.

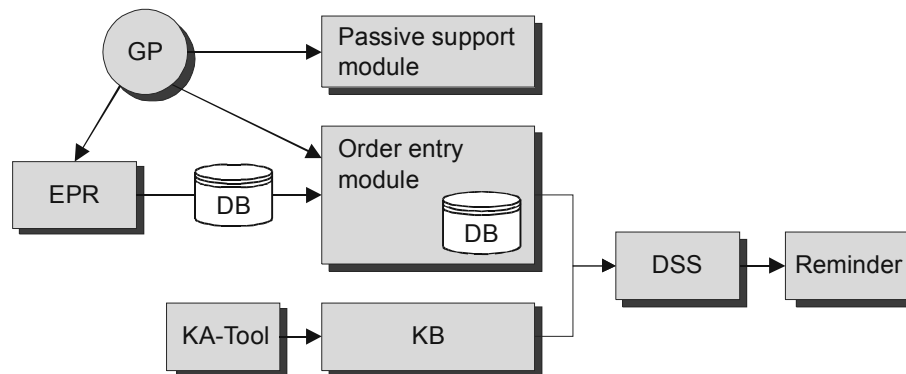


Figure 1: General structure of the automated feedback system. EPR is the Electronic Patient Record of the FP, DB is a database and DSS is the decision support system

Finally we added a facility that allows the FP to use his own terminology. The facility uses a list containing the medical terms for complaints and diagnoses from a list, containing all International Classification of Primary Care (ICPC) terms and their synonyms. After the FP has entered a search-term, a list of corresponding ICPC-terms [11] is presented and the FP has to choose the term that matches his/her description. The chosen term is translated into the corresponding ICPC-code. In this way we standardize the medical data the FP enters and therefore our guidelines in the knowledge base can be based on ICPC codes only.

2.3.2 Development and implementation

Acquisition

The GRIF guideline knowledge base consists of 134 rules that were extracted from known national and regional guidelines. These rules were represented as Situation Action Rules (SARs) [5]. The situation description (*'if-part'*) of each rule consists of logical propositions combined with Boolean operators, such as AND, OR or NOT. In this way situations are described for which feedback should be generated. The action part therefore only contains feedback such as giving a reminder to the physician. The guidelines are

entered into the knowledge base using the GRIF Knowledge Acquisition Tool (KA-Tool), which is part of the Gaston framework.

To allow reasoning about a medical domain, a domain ontology was built, which provides a domain of discourse by modeling entities and relationships for a particular domain of interest [5]. The ontology is read by the KA-Tool and implemented as a class tree, consisting of classes, subclasses and class members. The ontology consists of 153 objects divided into 11 classes, including diagnostic tests, patient information (age and gender), medical information (working hypotheses, existing problems and complaints) and the reasons for request. Based on concepts from the GRIF domain ontology, guideline authors were able to build the guidelines in the KA-Tool (examples of the workings and the user interface of the GRIF KA-Tool are shown elsewhere [6]).

Verification and testing

Verification refers to an internal static check on the system, which can be performed without test cases, and validation refers to tests performed to check the accuracy of the results given by the system, i.e. the performance of the system itself [12]. Two clinicians and one of the researchers carried out a logical verification (detection of contradictions and conflicts). A structure verification (detection of duplication of rules, circular rules and redundancies) of the knowledge base [13] was also carried out using a rule compiler [14].

Recommendations of the GRIF system were compared with comments of human experts using a retrospective random selection of 253 request forms, containing 1200 test requests. A panel of three expert physicians judged the requested tests independently, based on their interpretations of the practice guidelines.

The majority assessment of the physicians was compared with the assessment of the GRIF system. In case the system's output differed from the majority assessment the written practice guidelines were consulted. On average 1.75 recommendations were produced per form. In total 32 (7%) of the 442 given recommendations were given incorrectly. The amount of information provided and the level of detail (the specificity of the terms) with which the FP describes the patients' medical status are crucial for the GRIF system to react correctly [15].

Implementation

The KA-Tool transfers acquired rules to the Gaston Decision Support System (DSS) that provides the active support (e.g., generating the actual

recommendations). The DSS consists of a number of components that each performs a separate task such as processing the guidelines, interfacing with the order entry system and providing decision support to the GP [4]. In this case, an interface component was developed that interfaces with the order entry system. The DSS (see also figure 1) reads the patient data and checks whether any of the rules will fire and which feedback has to be provided. An example of feedback that is given by the DSS is shown in figure 2.

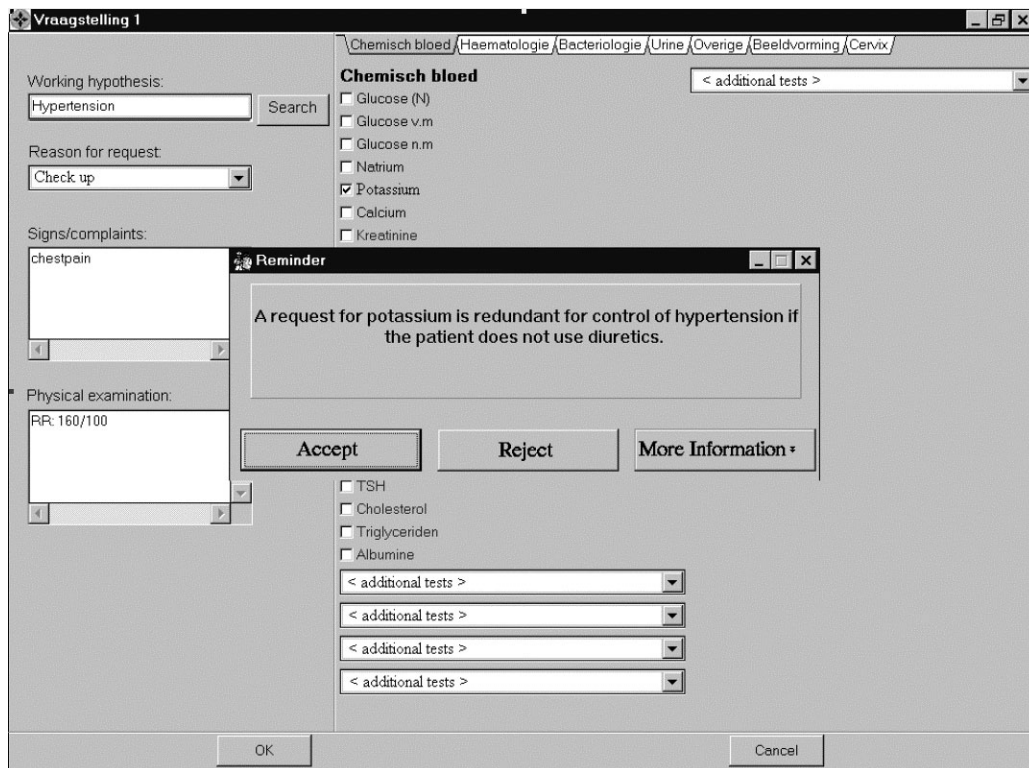


Figure 2: If a request is not according the guidelines, the reminder system generates and displays a reminder that overlays the normal user interface of the order entry form. Only the relevant words in the figure were translated from Dutch into English

In addition to the feedback, the pop-up window contains three buttons. The FP presses 'accept' to indicate that (s)he accepts the recommendation and 'reject' to indicate that (s)he does not agree. In both cases it is still possible for the FP to make changes in the request form after (s)he has seen the recommendation. Moreover the recommendation window contains a button to request more information. If this button is chosen the recommendation window will expand and may contain hyperlinks to the appropriate guidelines and/or references to the literature.

The GRIF system is in operation for more than two years in 15 Family Practices in the Netherlands and is still being used in daily practice.

2.4 Results

2.4.1 Efficacy evaluation in a laboratory setting

The efficacy of the GRIF system was evaluated in a laboratory setting. A randomized controlled trial (RCT) with balanced block design was used to study the potential effect of the GRIF system. The FPs reviewed a random sample of 30 request forms they filled in earlier that year. If deemed necessary, they could make changes in the tests requested. Next, the system displayed critical comments about their non-adherence to the guidelines as apparent from the (updated) request forms. Twenty-four randomly selected FPs participated. The number of requested diagnostic tests decreased with 17% (95% CI: 12-22%) due to the comments of the GRIF system. In addition, the fraction of tests ordered not in accordance with the practice guidelines decreased with 39% (95% CI: 28-51%). The FPs accepted 362 (50%) of the 729 recommendations. Although our experiment cannot predict the size of the actual effect of the GRIF system in daily practice, it was concluded that the observed effect might be the maximum achievable.

2.4.2 Evaluation in a daily practice

Eleven FPs in two regions of the Netherlands were monitored from August 2000 to July 2001. The GRIF system was implemented on the workstations at the offices of the participating FPs. The FPs were asked to use GRIF during patient consultation instead of filling in the paper request form. An analysis of usage behavior, the quality of provided information and the fraction of recommendations that was followed were analyzed.

During the intervention period, the FPs produced 2498 request forms using the GRIF system with 10139 tests on it. Of the 2780 recommendations, the percentage of followed recommendations varied between 3.4 and 8.3 percent dependent on the type of recommendation that was given. Advice that suggests removing a test because another - more appropriate or efficient - test was also requested and comments that suggest to request an alternative test were followed most frequently. The median time to generate, read and act on the presented feedback comments was 13 seconds. Entering (coded) medical patient data costs FPs a relatively large part of their patient consultation time.

2.4.3 Experiences with GRIF

FPs user-satisfaction with GRIF was measured using a questionnaire and group discussions (in the laboratory trial) and in-depth interviews (in the field trial) were conducted to elicit the opinions about and experiences with the system. The results show that the FPs in the laboratory trial had more positive

attitudes towards the system compared with the participants of the field trial. All discussion groups and most of the FPs in the field trial regarded receiving the feedback during the test ordering process an important advantage.

3 A real-time reminder system in Critical Care environments

3.1 Introduction

The CritICIS system is a real-time reminder system that reminds ICU health care workers of inconsistencies between a treatment plan and implemented guidelines. In the first version of CritICIS, all guidelines were implemented as Situation Action Rules (SARs), similar to the guidelines in the GRIF system. Physicians and nursing staff enter the rules using the CritICIS KA-Tool, after which a number of consistency and correctness tests are performed on the rules. The rules are then transferred to the knowledge base of the reminder system and validated by applying them to a large stored data set of previous patients. If the new rules are approved, they are exported to the reminder system that is used in daily practice. A detailed description of the process of representing, acquiring, verifying and executing rule-based guidelines in CritICIS has been published elsewhere [7]. This section describes how the CritICIS system was extended with guidelines that are no longer rule-based. In addition, this section also describes experiences of the use of the CritICIS system in daily practice in an Intensive Care Unit (ICU), where it is still in use today.

The objectives of the project were to provide decision support to health care workers in clinical care and emergency care environments and to design a knowledge acquisition environment that enables ICU care providers to formulate, update and verify guidelines without the assistance of a knowledge engineer. Also, decision support systems that operate in dynamic and unstable clinical domains such as the ICU should be real-time systems, because, even though the volume of data in such an environment can be overwhelming, the system's response must be timely.

3.2 Methods

3.2.1 Environment

The CritICIS environment consists of a collection of tools and modules to implement and evaluate guidelines in the ICU. An overview of all components is shown in figure 3.

The guidelines are entered by means of the CritICIS KA-Tool. This tool contains 1) Problem-Solving Methods (PSMs), which are generic strategies to

solve domain-independent stereotypical tasks (see also next section) and 2) primitives, which are small building blocks that are used to represent the steps in a guideline. In addition, the KA-Tool also contains the IMPACT domain ontology [16], which defines a set of medical terms, describing the state of a patient in an Intensive Care Unit. The IMPACT ontology defines a number of classes, each describing a distinct medical category such as drugs, treatments and diagnoses. More information about the use of PSMs, primitives and ontologies to acquire guidelines can be found elsewhere [5].

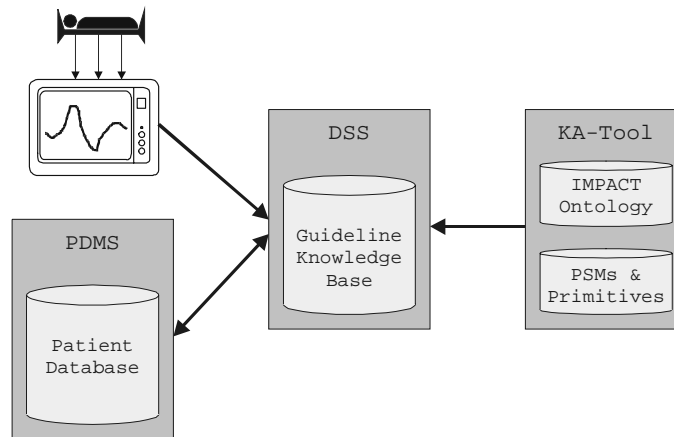


Figure 3: An overview of the main components of the CritICIS environment

When the guidelines have been entered, they are transferred to the CritICIS Decision Support System (DSS), where they form a guideline knowledge base. These guidelines are then executed by the DSS, which reads in the necessary patient data and compares the data with the guidelines. Whenever a guideline is not followed, the DSS will send a warning to the ICU care providers. The CritICIS DSS has access to two sources of data: 1) a Patient Data Management System (PDMS) that holds clinical data such as prescribed drugs and established diagnoses, and 2) a patient monitoring system that broadcasts physiological data such as a patient's blood pressure or heart rate. Examples of the KA-Tool, the DSS and the PDMS are shown elsewhere [7]. The CritICIS system is in operation since 2001 at the ICU of the Catharina Hospital, Eindhoven, the Netherlands.

3.2.2 Development and implementation

Acquisition

As mentioned earlier, the first version of the CritICIS system contained only rule-based guidelines. An example of such as rule is shown in figure 4.

```

IF Clavulanic acid is present
AND Cefuroxime is present
THEN give warning: "Prescribing Clavulanic acid and Cefuroxime at the same
                    time could be potentially hazardous for the patient."

```

Figure 4: A rule-based guideline in CritICIS that describes a drug interaction. This rule generates a reminder whenever the drugs Clavulanic acid and Cefuroxime are used at the same time

It is possible to describe certain classes of guidelines such as reminders by means of rules. However, describing guidelines in terms of the task that must be performed and the actions required executing such a task is usually a more natural way of representing knowledge. Besides representing less complex modular guidelines, rules can also be viewed as instances of a task. As a result of separating the domain concepts from the rule's syntax in the rule representation model, independent rules can be classified by identifying similar characteristics. For example, different rules that handle drug interactions and drug contraindications may all share the same syntax (figure 5).

```

Name: Undesirable combination of antibiotics.
Author: Korsten
Type: Reminder
Category: Undesirable medicine combinations
Explanation: Literature reference: [Zagola GP. and MacGregor D, The Critical
                    Care Drug Handbook 2nd Ed. pp. 165, 1997]
Message: Undesirable combination of antibiotics.
Rule premise: <drug:Clavulanic acid> and <drug:Cefuroxime>

Name: A Combination of a Beta-Blocker and Amiodarone is undesirable.
Author: Korsten
Type: Reminder
Category: Undesirable drug combinations
Explanation: See literature reference: [Zagola GP. and MacGregor D, The
                    Critical Care Drug Handbook 2nd Ed. pp. 195, 1997]
Message: A Combination of a Beta-Blocker and Amiodarone is undesirable.
Rule premise: <drug:Beta-Blocker> and <drug:Amiodarone>

Name: HOCM and pericarditis are contraindications for Digoxin
Author: Roos
Type: Reminder
Category: Digoxin
Explanation: See: [Drug Therapy in Cardiothoracic Surgery, v Zwieten en
                    Eijnsman, pp 40, 1997]
Message: HOCM and pericarditis are contraindications for Digoxin
Rule premise: (<disease:HOCM> or <disease:Pericarditis>) and <drug:Digoxine>

```

Figure 5: Two drug interaction guidelines and one drug contraindication guideline, taken from the CritICIS guideline knowledge base

Although these rules describe different drug interactions and contraindications, they share a general syntax. These generalized rules are referred to as rule templates (figure 6).

```

If <drug:A> is started;
And <drug:B> is present;
Then report that <drug:A> and <drug:B> are interactions

Drug contraindications:
If <drug:A> is started;
And <disease:B> is present;
Then report that <drug:A> and <disease:B> are contraindications

```

Figure 6: Examples of rule templates, acquired from analyzing similar rules

Sets of rules can be generalized to templates, on the basis of similar characteristics. Each template can be characterized by means of a pertinent event that executes the template (e.g., drug A is started) and actions that must be performed (e.g., report an interaction). A template also implicitly contains a relation between concepts. For example, regarding the drug interactions template shown above, drug A and drug B have an *interaction* relation.

Each rule template can be viewed as a guideline that exists of a sequence of steps that are carried out whenever a pertinent event occurs (e.g., starting a new drug). These types of guidelines are referred to as Event-Based Modular Tasks (EBMTs) and can be solved by the selection PSM that was especially developed to represent and solve EBMTs. More information on the representation and use of EBMTs and the selection PSM can be found elsewhere [5, 4]. In the CritICIS system, EBMTs that address drug interaction and contraindications were obtained by abstracting sets of rules to rule templates. However, it is also possible to acquire EBMTs directly from other sources such as the literature.

The CritICIS KA-Tool contains a number of PSMs and primitives such as SARs, EBMTs and flowcharts in order to capture various types of guidelines. As a result, the KA-Tool contains a number of different user interfaces that visualize these different types. Examples of the KA-Tool that visualizes SARs and EBMTs can be found elsewhere [5, 7]. Figure 7 shows an example of a more complex weaning guideline, acquired through the CritICIS KA-Tool.

Implementation

The first version of the CritICIS DSS was able to execute rule-based guidelines by reading data from the PDMS as well as the patient monitors and warning physicians and nursing staff when necessary by means of displaying pop-up windows [7]. In order to execute more complex guidelines such as

EBMTs and the above-mentioned weaning guideline, the DSS component that takes care of processing the guidelines was updated in order to be able to process these types of guidelines. It was not necessary to update the other components.

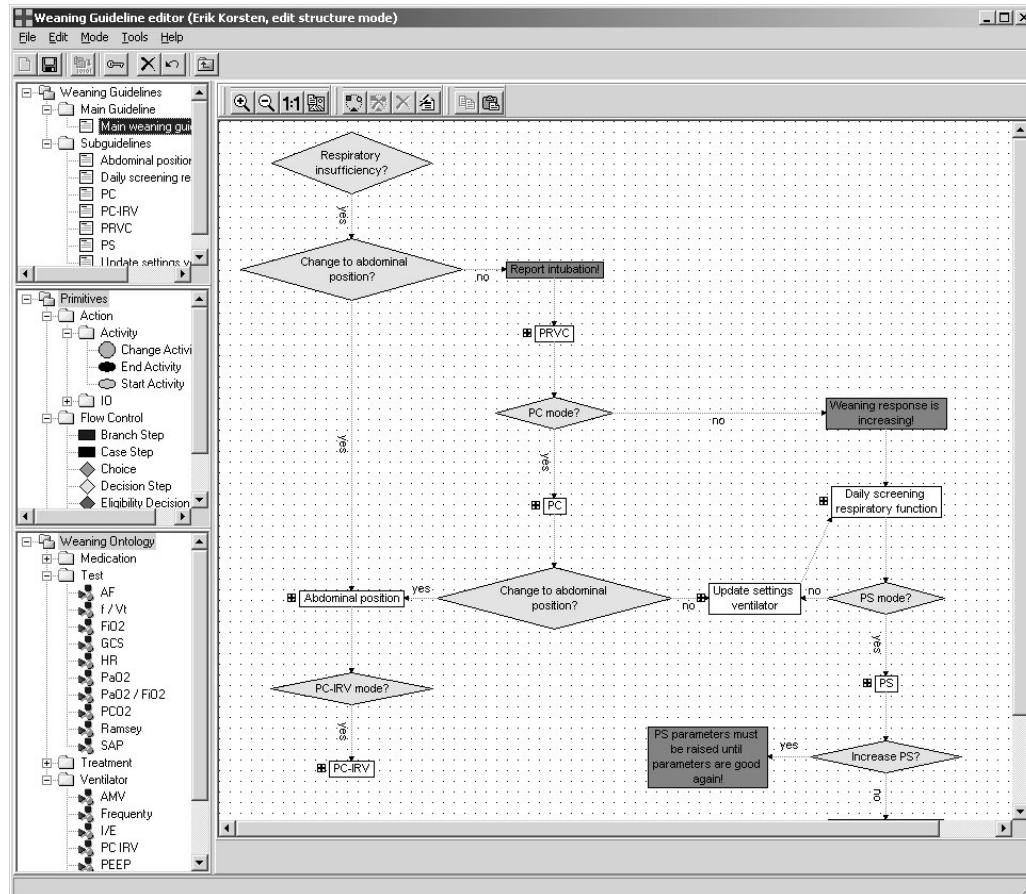


Figure 7: Part of a weaning guideline, entered in the CritICIS KA-Tool

An example is the execution of EBMTs. Whenever a certain event triggers a task (e.g., the prescription of a new drug), the EBMT procedure acquires the necessary knowledge from the domain ontology (e.g., all known contraindications of the started drug) and relevant patient data (e.g., all established diseases from the PDMS) and, if necessary, generates a critiquing statement: a recommendation involving one or more suggestions for possible modifications of the care provider's actions. For example, whenever an ICU care provider prescribes a new drug for a given patient, the PDMS system activates CritICIS with a 'starting a new drug' event. The PDMS also supplies additional parameters to CritICIS such as the patient's ID and the name of the started drug. This event causes the drug contraindications task to be executed. The drug contraindications task procedure then retrieves from the domain ontology all known diseases that have a contraindication relation with

the started drug and queries the PDMS to determine whether one of them is present. Whenever this is the case, the system generates a critiquing statement. Similar to the generation of reminders by rule-based guidelines, CritICIS generates critiquing statements by means of pop-up windows [5].

Currently, a closed-loop weaning decision support system is being developed at the ICU of the Catharina hospital. This real-time DSS continuously analyzes and monitors respiratory and lung mechanics, respiratory drive, gas exchange, blood gases, and hemodynamics in order to detect the patient's optimum flow requirements and ventilatory support, and instantly adapts the ventilator settings to the patient needs as required.

3.3 Results

3.3.1 Evaluation in daily practice

Evaluation of nursing guidelines for incomplete data

Besides the above-mentioned medical guidelines that provide decision support to ICU physicians, the CritICIS system also contains guidelines that are aimed to support the nursing staff when entering data in the PDMS. For a national study, in which the ICU participated, it was required that the results of certain laboratory tests had to be entered in the PDMS for a particular group of patients. Whenever an applicable patient was discharged and some of these test results were not entered at this time, CritICIS warned the nursing staff and provided a means for entering the missing data at that time. These data were then sent back to the PDMS. The system was implemented in daily practice at the ICU of the Catharina hospital, which consists of 21 beds and a nursing staff of 100 people. For one year, the number of daily discharged patients was measured as well as the number of reminders that were given to the nursing staff whenever one or more relevant data items were missing in the PDMS database. Figure 8 shows the number of reminders divided by the number of discharged patients for each day.

In 51% of all discharges patients, a reminder was given. It also shows that the number of reminders per discharged patient is not decreasing in the course of time, as one would expect. After conducting interviews with the nursing staff, it was clear that they were not using CritICIS as a reminder system, but as an intelligent order entry form. Part of the staff deliberately did not check whether the data were complete as they knew that CritICIS would check which data was missing and would provide a means for entering the required data during the patient's discharge process. The fact that the developers of CritICIS intended to increase the acceptance of CritICIS by not only reminding the

nursing staff but also giving them the means to enter the missing data on the spot resulted in this behavior. The advantage of this approach is that it does increase the system's acceptance. The drawback however, is that the users start depending on the system, which increases the possibility of errors whenever the system is not functioning or has an incomplete knowledge base.

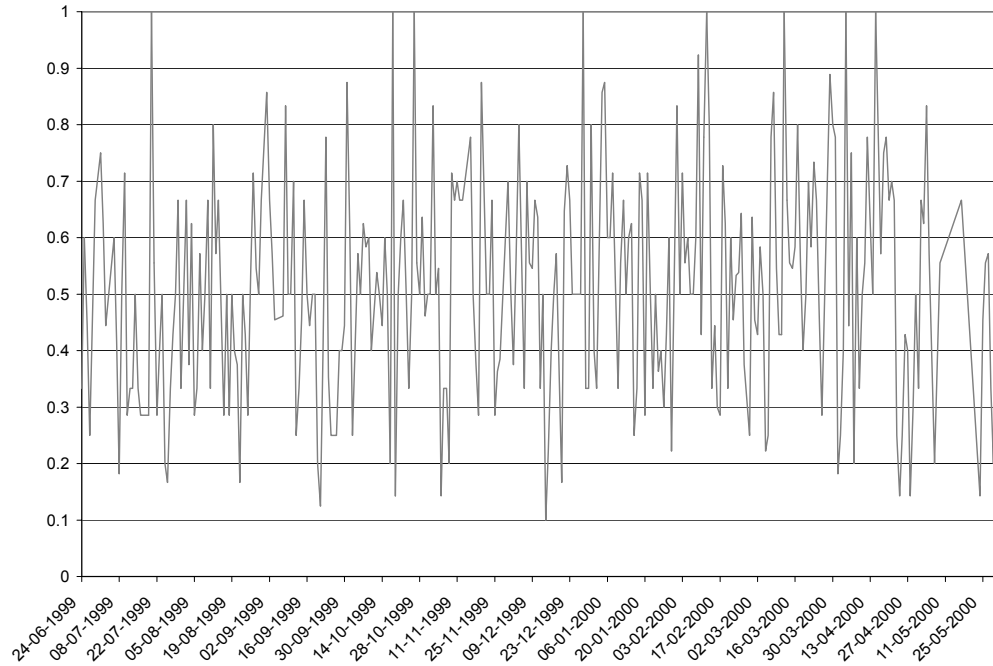


Figure 8: The number of reminders divided by the number of discharged patients, measured for nearly one year. The average number of discharged patients is 7.9 per day and the average number of reminders is 4.1 per day

Evaluation of medical guidelines

The CritICIS system contains a number of medical guidelines for example to detect drug interactions, contraindications and side effects or the absence of certain monitoring requirements [7]. The system, containing 67 rule-based guidelines and three EBMTs, was implemented for use in daily practice. For each guideline, only one reminder per day per patient was given. Whenever a reminder was issued more than one time for a certain patient, it was marked as 'hidden' and not shown to the physician. When a reminder was given, the physician was able to classify the reminder as 'correct' or 'incorrect' by examining the corresponding patient data in the PDMS. The classifications 'correct' and 'incorrect' correspond to the question whether a reminder was issued correctly in the given situation. Optionally, the physician could supply a rationale why (s)he classified a reminder as 'correct' or 'incorrect'. Whenever a physician was not able to judge a reminder (for example, due to lack of time), (s)he could ignore the reminder. The rule-based guidelines were

executed whenever a physician changed an item in the PDMS database. The EBMTs were executed whenever an event occurred that was relevant for a specific EBMT. For example, the drug interaction EBMT was executed whenever a new drug was being prescribed.

Over a period of 6 months, the DSS was executed 16,340 times. Of those 16,340 runs, there were 2,928 times that the DSS issued one or more reminders. The total number of reminders was 3,753 of which 2,731 were not shown to the physician (marked as 'hidden'). Of the 1,022 reminders that were given, 583 were ignored, 224 were classified as 'correct' and 215 were classified as 'incorrect'. Table 1 shows an overview of the 19 guidelines that led to non-hidden reminders and their classifications.

Name guideline	Correct	Incorrect	Ignore
A subarachnoid bleeding must be treated with a laxative	13	18	42
No anti-coagulant prescribed	77	95	208
Defibrillation not recorded	0	1	0
Drug interaction: beta-blockers and amiodaron	0	0	7
The complication oliguria not diagnosed	29	45	119
The complication anuria not diagnosed	15	26	56
A patient treated with renal-substitution therapy must be weighted daily	25	2	28
Serum-levels of the antibiotic must be checked	2	2	2
Check presence of imipenem	3	0	1
Check gentamicin Top/down	21	3	18
Due to hypotension, adjust the starting-dose of ACE-inhibition	0	0	18
Due to blunt abdominal trauma, check serum amylase	11	6	31
Due to blunt thorax-trauma, make an echocardiogram	5	4	9
This patient should be treated with coumarin	7	0	2
No treatment for decubit stage 1 or 2	2	0	4
No treatment for decubit stage 3	0	0	3
Digoxin usage, combined with a low potassium value	4	0	3
Start digoxin according to protocol for pneumonectomy-patients	9	13	31
Recently bradycardia: relative contraindication for beta-blockade	1	0	1

Table 1: Classification of the 19 guidelines that led to one or more reminders

215 reminders were classified as 'incorrect'. For the incorrect judgments, 64 times a rationale was given by an intensivist, explaining why (s)he found that the reminder was incorrectly given. All the reminders that were classified as incorrect concerned guidelines that were not specific enough or guidelines that had to be updated due to changes in the treatment plans. For example, the guideline 'no anti-coagulant prescribed' was classified as 'incorrect' 95 times. The users' rationales of why this guideline was not correct, repeatedly mentioned the fact that fragmin was prescribed, which is an anti-coagulant. The ICU staff had added fragmin to the list of used anti-coagulants, but this information was not mirrored in the guideline. After updating the guideline, all erroneous reminders disappeared.

Other incorrect reminders were a result of guidelines that were not specific enough. For example, the guideline 'Start digoxin according to protocol for pneumonectomy-patients', which states that a physician should prescribe digoxin to a patient who underwent pneumonectomy, was classified 31 times as incorrect. The rationale showed that only patients that *recently* underwent pneumonectomy should be prescribed with digoxin.

All the guidelines that led to the incorrect reminder, shown in figure 1, could be updated in order to decrease the number of false reminders. Some of these guidelines were already examined during a retrospective study using data of earlier admitted patients [7]. However, this study showed that it is still possible to get incorrect reminders as a result of new patient data and changing policies or guidelines.

3.3.2 User satisfaction with CritICIS

In order to measure the user satisfaction with CritICIS, a questionnaire was developed, based on the IBM computer usability satisfaction questionnaire [17]. The questionnaire consisted of 34 items, of which 14 were related to usability, 7 to training and support, 4 to user satisfaction, 5 to behavioral changes and 4 to usefulness (the questionnaire can be found in the appendix of this paper). All items could be scored on a 5-point Likert scale where 1 is 'strongly agree' and 5 is 'strongly disagree'. For all categories, the mean score of each intensivist was calculated and classified as 'positive' (mean < 2.5), 'neutral' (2.5 ≤ mean ≤ 3.5) and 'negative' (mean > 3.5). Furthermore, the questionnaire contained a number of open questions where opinions and experiences could be given about the system. The questionnaire was given to three intensivists that have been working with the CritICIS system in the ICU of the Catharina hospital. All three have more than 15 years experience working in the ICU and consider themselves expert on working with

computers in general and working with the PDMS in particular. Afterwards, the outcome of the questionnaires was discussed with the intensivists.

All intensivists scored 'positive' in the usability category (means: 2.2, 1.2, 2.4), meaning that they found the system workable. The user interface was generally regarded as 'intuitive' and easy-to-use. The intensivists disagreed somewhat on the issues of productivity and effectivity, as some stated that it 'slowed down the process of entering patient data'.

In the training and support category, one intensivist scored 'positive' and two scored 'neutral' (means: 2.7, 1, 2.6). The comments and discussion showed that they found support and training sufficient.

In the user satisfaction category, all scores were classified as 'positive' (means: 1.5, 2, 1.3), meaning that they were satisfied with the system. This opinion was confirmed in the comments and the discussion afterwards.

Regarding behavioral change, two intensivists scored 'neutral' and one scored 'positive' (means 2.8, 3, 2.4). In general, the intensivists did not believe that the use of critiquing systems such as CritICIS would automatically change their behavior, especially concerning the amount of entered data. They did state that they would be willing to encode more information in the PDMS for the purpose of decision support. Also, a combination of critiquing and proactive decision support would be favorable for them.

All intensivists strongly agreed that systems such as CritICIS are useful (means: 1, 1.5, 1) and that similar systems must be implemented in other departments.

Other comments of the intensivists concerned issues related to completeness, local adaptation and interfacing. They stated that, in order to improve the acceptance of the system in daily practice, it was necessary that the guideline knowledge base must at least contain those guidelines that cover the daily routine of the ICU. They stated further that a systematic procedure is mandatory that facilitates entering new guidelines or updating existing ones. Hospital organizational bodies must support this procedure. Also, they want to use (inter)national guidelines as a basis, from which they must be able to make local adaptations that fit their own institution. Furthermore, the guideline knowledge base should contain more treatment guidelines that suggest best practices and more nursing guidelines to improve the system's acceptance for the entire ICU.

Regarding decision support, they stated that, besides the PDMS and patient monitors, the CritICIS system should be interfaced to even more ICU-related equipment such as ventilators, pumps and laboratory systems. In addition to the current critiquing approach, they also suggested a more pro-active approach. Currently, the CritICIS system is implemented as a critiquing system that warns physicians whenever a guideline is not followed. A pro-active approach would enable them to ask the system for advice regarding certain complications, treatments or possible differential diagnosis. Finally, they suggested that it must be possible for CritICIS to send reminder-related data back to the PDMS if the intensivist agrees to a reminder. For example, whenever a reminder states that the prescription of a certain medication is not advisable, it must be possible to inform the PDMS that this medication must be stopped immediately.

4 A Multidisciplinary Psychoactive Drug Selection Advisory System

4.1 Introduction

Irrational and inconsistent use of psychoactive drugs is common in clinical practice due to the complex knowledge and data intensive nature of the psychoactive drug selection and prescription process. Sub optimal psychoactive drug therapy leads to hospital admissions, extended length of hospital stay, ineffective therapy, increased mortality and last but not least to increased costs [18]. The psychoactive drug selection process requires expertise from clinical, pathophysiological and pharmacotherapeutic knowledge [19]. Due to the information load, the lack of appropriate up to date information at the point of clinical care and the problem of integrating and weighing all information, it is questionable whether any clinician can manage such a complex situation effectively. As shown in a number of experiments, clinicians can benefit from knowledge-based systems to improve the psychoactive drug prescription [20].

A psychopharmacological advisor must meet a range of functional requirements, for the clinician (user), the domain expert and the knowledge engineer:

- The knowledge base should incorporate a multidisciplinary view on psychoactive drug selection, because different clinicians use different knowledge domains.
- The psychopharmacological advisory system should be able to explain its advice to make the reasoning transparent to the clinician.

- The psychopharmacological advisory system should be integrated with a patient record system.
- It must be easy for the domain expert to create and to maintain the knowledge base and to interact with the knowledge base in his own terminology in order to incorporate easily the continuously evolving neuropsychopharmacological knowledge.
- Minimizing the difficulty of updating, both when new neuropsychopharmacotherapeutic knowledge comes available or in the situation of new reasoning strategies.
- To share domain specific knowledge by different reasoning strategies to solve completely different tasks such as drug selection or drug administration tasks.
- To reuse reasoning components across divergent application domains (such as reasoning strategies to manage depression and bipolar disorder). This can save development effort in building new decision aids for new application domains.

4.2 Methods

4.2.1 Environment and System Overview

Instead of building a rule-based system in a straightforward way by the transcription of rules elicited from a domain expert, a psychopharmacological advisory system was developed in terms of the construction of a series of explicit models related to the psychoactive drug selection task. A general overview of the models and the software components to compose such a system is illustrated in figure 9.

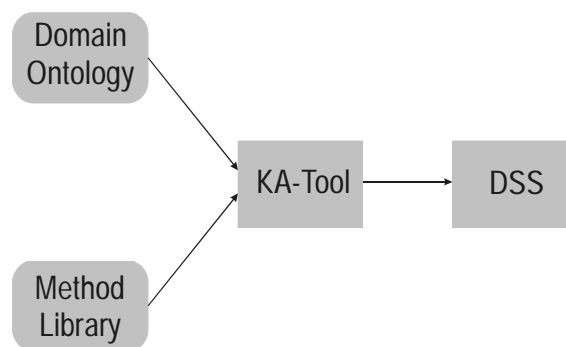


Figure 9: Overview of the components (rectangles) and models (rounded rectangles) of a psychopharmacological advisory system

The development and implementation process of a psychopharmacological advisory system as shown in figure 9 consists of four steps:

1. Development of a domain ontology
2. Construction of a Problem Solving Method (PSM) in the method library
3. Develop guidelines by refining PSMs with domain entities in the KA-Tool
4. Development of the Decision Support System.

Acquisition

In earlier research [19], a neuropsychopharmacological domain ontology relevant to rational psychoactive drug selection was defined, which combines all involved knowledge domains, required to support the psychoactive drug selection task. This ontology explicitly and formally represents domain classes such as *Drug_Therapy*, *Therapeutic* and concepts such as *Anafranil_Therapy*, *Anafranil*, *Clomipramine*, *Antagonism-of-Alpha-1-receptor* and relations such as *acts_On* and *has_Location*.

The domain ontology was defined through the Galen Representation and Integration Language (Grail), which is part of the Galen framework: a technology that facilitates the development of medical terminology and coding schemes [21]. Grail supports the composition of complex concepts from elementary concepts such as '*Dopamine-2-receptor at the postsynaptic membrane*'. The concepts from the Grail domain ontology were translated into a frame-based representation, in which classes have attributes of defined cardinality and data type (e.g., integer, float, string or Boolean). This representation is then used during the guideline acquisition process to acquire the guidelines that describe the rational psychoactive drug selection task (described in the next sections).

Furthermore, a knowledge analysis was performed to describe the psychoactive drug selection task by means of a clinical algorithm [8] and to model in a semiformal way the specification of this task [22]. The psychoactive drug selection task can be viewed as a modular task, which consists of the execution of different subtasks. Each (sub)task can be furthermore divided into more subtasks. As illustrated in figure 10, the psychoactive drug selection task can be divided into seven subtasks: 1) generating the candidate drug-therapy options for treating a specific psychiatric condition, 2) filtering out those options that are in conflict with one or more aspects of the patient's condition, 3) determining the level of contraindication, 4) determining the risks of the relative contraindications, 5) determining the required monitoring activities, 6) rank ordering the therapy options according to different neuropsychopharmacological-economical parameters and 7) determining the patient preferences.

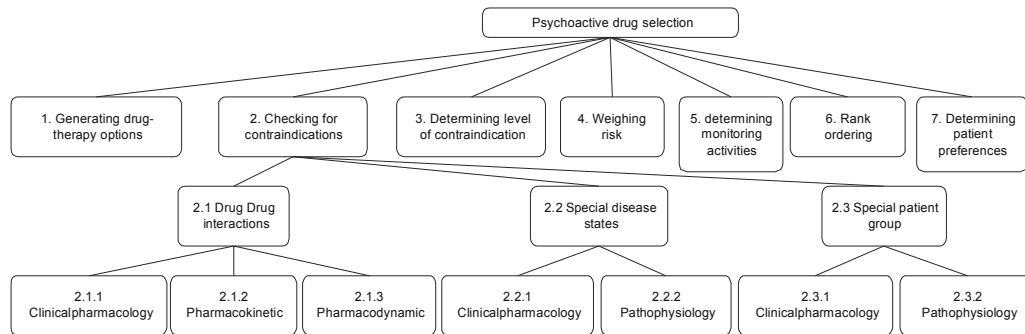


Figure 10: The hierarchical representation of the psychoactive drug selection task

These seven main (sub)tasks may be further subdivided in other subtasks. For example as represented in a hierarchical way in figure 10 the task that filters out possible contraindications (task 2) consists of three subtasks which refer to three possible clinical situations where a contraindication may occur, namely (task 2.1) contraindications based on drug-drug interactions, (task 2.2) contraindications on special disease states and (task 2.3) contraindications on special patient groups. Each of these subtasks can be further subdivided referring to different reasoning strategies. These contraindication tasks can be solved by reasoning based on clinical pharmacological knowledge (empirical) or derived by deep level pharmacokinetic, pharmacodynamic, and pathophysiological knowledge. All tasks are implemented by means of the earlier-mentioned Event-Based Modular Tasks (EBMTs) and corresponding PSMs, which are stored in the method library [5].

The KA-Tool loads the domain ontology and method library and creates a user interface that enables guideline authors to define guidelines that describe the psychoactive drug selection task in terms of primitives, PSMs and domain entities. Figure 11 shows the user interface of the KA-Tool that visualizes the *generate drug-therapy option* task.

The guideline acquisition process consists of refining each PSM with the appropriate domain entities. The KA-Tool loads all instances which enables the domain expert to fill in the knowledge roles of the selected PSM by means of domain specific knowledge. In this way, domain experts are only able to enter domain specific knowledge, while the knowledge that describes the problem solving method (which is usually a far more complex structure) remains unaltered. The domain expert can select an instance from the domain ontology class tree (e.g. *Major_Depression*) and drag it into the *DSM_IV Diagnosis* pane on the left side. Next he can drag a specific drug therapy (e.g. *Anafranil_Therapy*) to the *Drug_Therapy* pane. By dragging an *Anafranil_Therapy* he creates an *is_indicated_by* relation between the

Major_Depression and *Anafranil_Therapy*. By dragging the instance *Anafranil* to the Therapeutic pane he creates the *has_therapeutic* relation between *Anafranil_Therapy* and *Anafranil*. The number and the format of the panes on the right side of the user interface depend on the knowledge roles of the PSMs. In this way, the domain expert creates and maintains in his own terminology the knowledge base of the psychopharmacological advisory system.

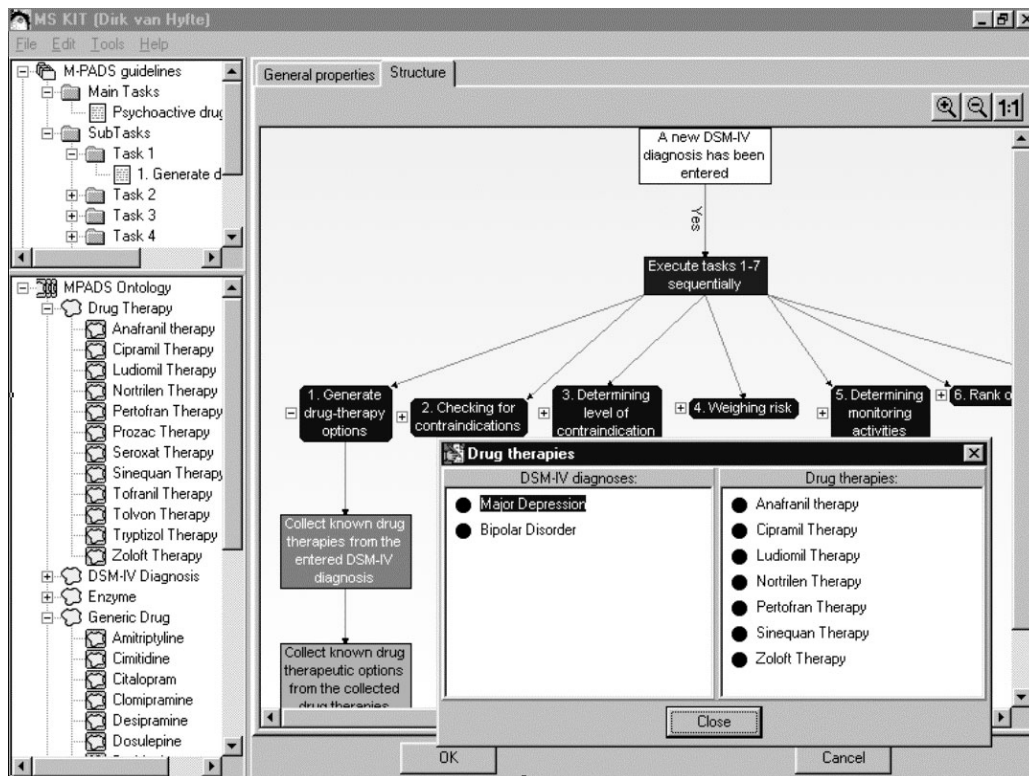


Figure 11: The user interface of the KA-Tool that visualizes the generate drug-therapy option task

Implementation

After entering the guidelines, The KA-Tool generates a knowledge base, which translates the guideline knowledge base into a format that is interpretable by Decision Support System (DSS), known as the Multidisciplinary Psychoactive Drug Selection Advisory System (M-PADS).

The DSS is activated when the clinician selects the psychoactive drug selection task. First the clinician has to enter the established DSM-IV diagnosis of the patient. A list of possible drug-therapy options is now generated. The clinician may now enter the concurrent medication (e.g. Zantac) and/or the special disease state (e.g. hypotension), and/or the special patient group (e.g. elderly) of the patient. The decision support system

generates on request a list of possible contraindications. The level of each contraindication is mentioned (e.g. absolute/relative). The clinician can now request the required monitoring activities of the relative contraindication. The user can accept the allowed relative contraindications. On the basis of the accepted drug-therapy options, the user can order the options according to different psychopharmaco-pharmacoeconomic parameters. At this moment, this rank ordering is now by hand. For each drug-therapy option, the associated therapeutic with its modality (e.g. tablets, capsule, ampoule etc.), dosage and cost can be requested. The patient and the clinician can now discuss the options and select the preferred therapeutical one. The guideline knowledge base currently contains knowledge of 15 psychoactive drugs, 200 elementary concepts and 300 compositional concepts related to the antidepressant drug therapy domain. Figure 12 shows a first prototype of the M-PADS DSS.

The screenshot shows the M-PADS Decision Support System interface. It is a window with a title bar and several input fields. The fields are organized into five columns: DSM-IV Diagnosis, Concurrent Medication, Special Disease States, Special Patient Group, and Patient Preference. Each column contains a list of options. Below these columns is a section for Drug Therapy Options, which includes a list of therapy options and a SUBMIT button. At the bottom of the window are five buttons: TRUE?, MONITORING ACTIVITY?, ACCEPT RISK?, EXPLANATION, and MORE.

Figure 12: The user interface of M-PADS, illustrating the required patient data

4.3 Results

By using the methods, described in the previous section, it was possible to develop a psychopharmacological advisory system based on explicit models of the neuropsychopharmacological domain and the problem solving method related to the psychoactive drug selection task. These explicit models integrate the clinical pharmacological, pathophysiological and pharmacotherapeutic knowledge required to support rational psychoactive drug selection. At the moment, M-PADS is able to give patient specific advice, based on up to date knowledge to treat major depressions. Since the knowledge base is organized in a modular fashion with declarative and

procedural knowledge separated, it can be easily expanded or modified so that the knowledge remains up to date. We believe that M-PADS will have opportunities to built reusable and explainable knowledge based systems for pharmacotherapy. The results of this project have led to the publication of a PhD thesis [23].

5 Consumer Health Records for managing chronic diseases

5.1 Introduction

Patient-centered care is an emerging theme in healthcare. In patient-centered care the patient is actively involved in the care delivery process. The gradual but deliberate transition of health services from the hospital and clinic to the home and community creates an environment in which patients must independently assess and interpret symptoms, seek appropriate health services in a purposeful manner, and engage in health promotion, disease prevention, and illness management activities [24]. To accomplish these tasks, patients require access to information about disease processes, credible intervention strategies, and personal health data. Information systems are needed that provide patients with access to these types of information. A recent study showed that for chronic patients a patient-centered approach 1) increases the patients' satisfaction with their physician's care, 2) increases the patients' interest in the contents of their medical records and 3) improves the patients' overall health status [25].

Physicians increasingly start using clinical information systems such as Electronic Patient Records (EPRs) to store and present patient data [26]. Vendors increasingly sell (disk) space to consumers (which may be patients) in which they can store relevant information about their health. These systems are referred to as consumer record systems. The patient information that is stored in these systems can be obtained from various sources such as the information system of the patient's physician or pharmacist and other sources such as a Hospital Information System (HIS) or a laboratory system. In addition, the patient could enter the information provided by the healthcare provider into his/her consumer record. Although this may increase the involvement in the management of their disease, it may decrease the validity of the medical data. Since patients usually do not have the medical knowledge and background of a care provider, they may interpret the care provider's information incorrectly, resulting in erroneously entered data. In this case, automated decision support systems are able to guide patients (and physicians) during the entry of such data and also provide feedback about the patient's disease.

This section describes the development of consumer health record systems that have been developed during two projects: 1) the TANDEM project [27], which focuses on the treatment of Diabetes and 2) the Medical Guideline Technology (MGT) project [28] that focuses on the treatment of Hypertension.

In these projects, data entry was carried out both by the care provider and the patient, depending on the subject. During the development of the system the reactions of the users about the functioning of the system are constantly monitored via questionnaires. Since only a limited number of patients and care providers were involved in the assessment of the system an ethnographic-like approach (see later) was selected that in principle provides useful results even with a small number of assessors. The architecture of the system is displayed in figure 13. It serves as a conceptual framework that identifies specific components and linkages that will engage the patient's perspective in the design of a healthcare information system. The patient is the center of the model. The surrounding elements denote the diverse sources of health information.

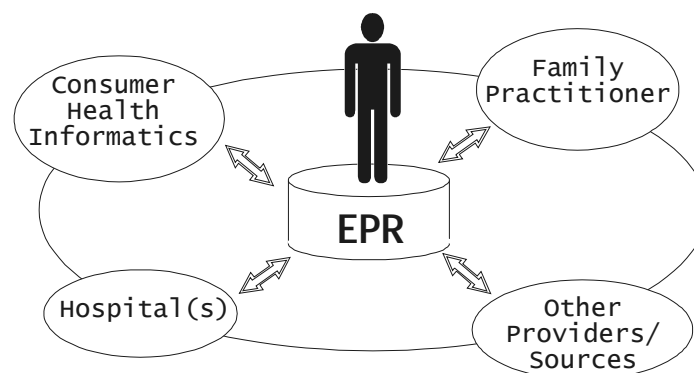


Figure 13: A model for patient-centered information systems (adapted from Brennan et al. [24])

The consumer health record should be easily accessible by both the patient and the care provider. The care provider is responsible for the medical data, the patient for those data that (s)he is able to provide trustworthy. The contents of the record must be presented in a flexible manner so that it can be changed or updated as a result of changing requirements of the patient and/or provider.

A drawback of many health record systems is that these systems are not 'open', meaning that there is no means of information exchange between the health record system and other information systems. Therefore, another requirement is that the consumer health record system must be able to

exchange information with other systems by means of standardized protocols (e.g., HL7 messages).

To support the active role of a patient, the patients must 1) be able to discuss public topics with other patients and care providers, but also private topics with care providers only and 2) be provided with feedback or advice, based on data stored in the consumer health record.

The necessary functions to carry out the evaluation study should be incorporated into the system.

5.2 Methods

5.2.1 Environment

A toolkit was developed that enables system engineers to develop consumer health record systems that meet the above-mentioned requirements.

To fulfill the criteria of easy accessibility, the toolkit applies web-technology for viewing and entering patient information. As a result, patients and care providers only require a web-browser to view and edit patient data. In order to exchange data with other information systems, consumer health record systems can be configured for information exchange by means of pre-defined communication protocols (e.g. HL7 messages).

To support the active role of the patient, the toolkit allows care providers to enter computer-based guidelines concerning chronic diseases that can be executed by the consumer health record system to provide advice or feedback to patients and care providers. The computer-based guidelines are created and executed by means of the Gaston framework, which is included in the toolkit. Gaston consists of a suite of tools and reusable software components that support the various stages in guideline development, from guideline design to guideline execution. The framework includes design-time components to facilitate the guideline authoring process along with execution-time components for building decision support systems that incorporate these guidelines [4].

Also, the toolkit supports the use of discussion forums, in which patients and care providers are able to discuss various topics related to the chronic disease.

Finally, it was decided to evaluate the development of the consumer health record system by means of an ethnographic-like approach [29]. This is a subjective approach with which among others the development of information resources can be evaluated. In contrast with more objective evaluation

methods, this approach seeks to represent the viewpoint of the system's users (e.g., patients and care providers) as well as other significant participants in the clinical environment where the system operates. The goal of this approach is illumination rather than judgment. The investigators seek to build an argument that promotes deeper understanding of the information resource. It addresses the deeper questions: the detailed 'whys' and 'according to whoms' in addition to the aggregate 'whethers' and 'whats'. Researchers play an active role in the evaluation process and immerse themselves physically in the consumer health record's environment. They collect data primarily through interviews and document reviews. In our case this means observing the actions of patients and care providers, as apparent from their contacts with the consumer health record system, and constantly asking them about their opinion with respect to the system's functioning. To this end patient-specific questionnaires are embedded in the health record system to collect the necessary data from patients and care providers. The design of a consumer health record system is not rigidly predetermined and does not unfold in a fixed sequence, but develops dynamically as the experience of the researcher or developer increases. Therefore the health record system can be updated easily by means of the toolkit.

The output of the toolkit is a web-based system that contains the following functions:

1. *Viewing and entering patient-specific information.* The main purpose of a health record is to store and present patient-specific information and share it between patient and care provider. Therefore, patients as well as care providers are able to view and enter data in the health record.
2. *Exchange patient data with other information systems.* Through the toolkit, the consumer health record system can be configured for information exchange. For example, when instructed, the consumer health-care record system developed in the TANDEM project automatically acquires glucose values from a glucose meter and stores these values in the health record.
3. *Provide patient and care provider-specific advice, based on guidelines.* The Gaston framework was used to enable care providers to acquire and execute computer-based guidelines. First, guidelines were created by means of the Gaston KA-Tool, after which the guidelines were compiled and uploaded to a server on which the execution engine (DSS) resides. During execution, patient data was sent from the health record system to the Gaston DSS that combined received patient data and acquired guidelines and sent advice back to the health record system
4. *Provide discussion forums.* Discussion forums provide a means for patients and care providers to discuss topics related to the domain of the

chronic disease. Patients are able to submit messages to a discussion forum or react to messages that were submitted earlier. The consumer health record system provides two types of discussion: public and private. The public discussion forum contains public messages that are accessible by every patient who has access to the web pages. The private discussion forum contains messages that are only accessible by the patient who enters the private forum and the patient's care provider. This discussion forum is meant for personal questions and topics.

5. *Evaluate the consumer health record system design.* The design process of the consumer health record system is evaluated by means of the ethnographical-like approach. The ethnographical approach is an iterative process. Based on the outcome of previous questionnaires combined with current patient data, new patient-specific questionnaires are developed. Also, the structure of the health record is regularly updated by using the outcome of the questionnaires.
6. *Provide additional information about the disease.* This section contains background information about the chronic disease that is managed by means of the health record. In case of the TANDEM health record system, the information consists of links to pages with information on diabetes.

Figure 14 shows an example of the health record system that was developed for the management of diabetes in the TANDEM project. The patient information section is selected, resulting in an overview of all available patient data. The information is divided into a number of tab pages such as general information, complications, medications, etc (the format of these tab pages is configured through the toolkit). In this case, the patient has selected the Psychological Health Profile (PHP) tab page, where the patient can enter personal information (one of the research goals of the TANDEM project is to determine if a PHP can be used to improve the patient's well-being [30]). The patient information is shared between patient and care provider, although certain information can only be entered by the care provider (e.g., the patient's medical history).

The health record is used by three groups of consumers: patients, care providers and researchers. Patient information is entered and updated by care providers as well as patients. During a (pilot) study, a study protocol is formulated, which specifies what kind of information should be updated or added. The protocol also specifies the time-interval between new entries. For example, the protocol in the TANDEM project states that at least every week, the patient enters new relevant personal data such as the patient's weight, medication dosages and PHP. The glucose values are acquired from the glucose meter. Advice is also provided when asked for, based on

implemented guidelines. Furthermore, patients are encouraged to discuss diabetes-related topics with other patients and care providers. The patients and care providers have to answer a questionnaire on a regular basis. The questions are based on the results of previous questionnaires and the current patient data. These current patient data indicate which functions were used most recently, so that the questions in the questionnaire can focus on these functions.

The screenshot shows a web browser window titled "Patient information - Microsoft Internet Explorer". The address bar shows a local URL. The main content area is titled "Information" and contains a "Save information" button. Below this is a tabbed interface with tabs for "General", "Complications", "Medication", "Recent", "Glucose", and "PHP". The "PHP" tab is selected, displaying a questionnaire titled "PHP:". The questionnaire consists of a table with "Description" and "Judgement" columns. The "Judgement" column has five sub-columns: "Always", "Nearly always", "It Varies", "Almost never", and "Never". Each sub-column contains a radio button for selection. The "Description" column lists ten items: Discontentment, Depression, Hopelessness, Worrying, Panic anxiety, Obsession/Compulsion, Vulnerability, Expressed anger, Contained anger, and Mistrust. To the left of the main content area, there is a vertical menu with buttons for "Information", "Glucose Upload", "Advice", "Discussion", "Links", and "Comments". At the bottom left, there is a note: "For questions, remarks and comment, mail the webmaster."

Description	Judgement				
	Always	Nearly always	It Varies	Almost never	Never
Discontentment	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Depression	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Hopelessness	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Worrying	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Panic anxiety	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Obsession/Compulsion	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Vulnerability	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Expressed anger	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Contained anger	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Mistrust	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 14: Part of the patient information section in the health record, developed for the management of diabetes. Other available sections are shown in the left part of the record

5.2.2 Development and implementation

Acquisition

Regarding the TANDEM project, the SAR (Situation Action Rule) model [5] was chosen that defines guidelines in terms of 'if-then' rules. These rule-based guidelines were aimed at providing advice to patients about hypertension (high blood pressure), rapid weight loss or HbA1c-increase.

The guidelines for the treatment of hypertension for use in the MGT project were defined as temporally sequenced graphs (flowcharts) These guidelines, translated from the WHO paper-based guideline for the treatment of

hypertension [31], were aimed to provide treatment-related advice to care providers. In both projects, the Gaston KA-Tool was used to acquire the guidelines. Examples of the user interface of the KA-Tool that visualizes SARs and flowcharts are found elsewhere [4].

Implementation

In both the TANDEM and MGT projects, the web pages were installed on a server that is accessible via a browser by registered patients and care providers. The execution engine of the advice systems was also installed on a server and was able to communicate with the health record systems via an XML/TCP interface. For this purpose, an interface component was developed that communicated by sending and receiving XML-messages over the TCP networking protocol. The guidelines that were created by means of the KA-Tools could be automatically uploaded to this server by the guideline authors themselves. Whenever the patient or care provider requests advice through the 'advice' section, all relevant guidelines were executed. Each guideline retrieves the necessary patient information from the consumer health record and checks if advice must be given. If so, the action component of the execution engine translates this advice into a webpage and sends it back to the health record system so that it could be shown to a patient or care provider. Examples of generated web pages are found elsewhere [4, 5].

5.3 Results

5.3.1 Implementation

The TANDEM consumer health care record system has been in operation for several years. During the last part of the development a pilot study was carried out with 10 patients and two care providers both to assess the usability of the system and the appreciation of the system. The results are promising. Patients as well as care providers received the consumer health record system very well. The user interface was regarded useful and sufficiently 'intuitive'. Patients as well as care providers agreed that the system was easy to use and that no further training was necessary. For the TANDEM project, the care providers insisted that during the pilot project, patient-specific advice would be hidden from the patients as the care providers were afraid that this would be regarded as to 'harsh' by the patient without proper instruction or training. Instead, all patient-specific advices were sent as email messages to the patient's care provider.

5.3.2 Experiences with the consumer health record

For the patient data-related pages, remarks were being made about the layout of the screens. For example, some patients regularly measured only height

and weight. From the screens they got the impression that they had to enter more data. At least it was not clear to them initially which screens would be filled in by the care provider and which screens they had to fill in. These questions came in an early stage at a time that the diabetologist had not yet entered the medical data of these patients. But indeed a better layout could improve the visibility and better differentiate the medical parts from the consumer parts. The system was updated according to these remarks. On the other hand, other patients asked whether it was possible to enter more data, for example the entering of (changes in) activities, the occurrence of certain diseases (e.g. flu) and (related) changes in therapy.

Also, remarks were made about the terminology used in the health record. The diabetologist and diabetes nurse assumed that the patients would be able to interpret the medical terms used in the screens as the various screens were designed based on the content of existing forms, which were regularly discussed with the patients. However, some patients did not understand some medical terms although they had been filling in the paper-based forms for some time.

The patients had mixed feelings about the public discussion forum but were rather positive about the private discussion forum as this provided for them an easy opportunity to consult their diabetologist or nursing staff.

The main attitude of the patients was that they were convinced that a consumer health record as the one provided via these projects would be a common feature of the future. It encourages the patients to actively work together with the physician to solve their medical problems. The fact that you have to enter your own data also provides a better insight in your situation. This is especially true for patients with a chronic disease, who are becoming more and more aware of their own health condition. As a result, the role of the patient in the process of disease-management is changing from passive to more active. The development of sharable consumer health records is crucial in this process. However, there are several issues that have to be dealt with such as the validity of the entered data, the presentation of information and the ability of exchanging information with other systems and users. Although the use of guideline-based decision support systems in these projects was still limited, the first results and experiences convinced us that these techniques are an integral part of patient-centered health records.

6 Discussion

Although a lot of progress is being made in the area of guideline-based decision support, actual decision support systems are still not implemented on a large scale. One of the largest problems is that the medical community is very heterogeneous by nature. Numerous medical specialties exist, each with their own types of guidelines, intended users and information systems. Local institutions usually have their own local customs and regulations, which demand that it must be possible to 'override' national guidelines with local adaptations. Also, interfacing decision support systems with third-party systems (e.g., EPRs) as well as with the system's users (e.g., care providers) usually requires a lot of effort and resources due to a lack of standardization.

The Gaston approach was developed in order to limit the amount of time and resources by means of developing an architecture that can be used to implement a large range of guideline-based decision support systems. The experiences, described in this paper show that it is possible to use Gaston to develop systems that differ in application domain (e.g., family practice, critical care, psychiatry, chronic disease management), application environment (e.g., FP information system, PDMS, consumer health record system) and application users (e.g., FP, physician, nursing staff, patient).

Using the developed representation model, combined with the corresponding KA-Tools, it was possible to develop and acquire different types of guidelines such as rule-based guidelines, EBMTs and multiple-step guidelines. In the four projects, custom-developed domain ontologies were used (although partly based on existing terminologies such as ICPC and IMPACT). In order to improve standardization aspects, it might be more favorable to use standard terminologies such as UMLS [32] or SNOMED [33] for all projects. On the other hand, it is important that guideline authors in local institutions 'recognize' their own concepts. For example, a guideline may refer to the medication Fluoxetine Hydrochloride that is defined in a standard domain ontology, which might be better known in some specialties by its brand name Prozac. In Gaston, mappings tools were utilized to reuse similar concepts in various projects.

The Gaston KA-Tool was used to acquire all guidelines, varying from the relatively simple rule-based guidelines (GRIF, TANDEM, CritICIS) to the very complex hypertension (MGT) and weaning guidelines (CritICIS). Similar to the development and application of domain ontologies, it is important to reach a balance between standardization and easy-of-use. Defining multiple user interfaces in the Gaston KA-Tool, based on the underlying guideline

representation model, made it possible to reuse the KA-Tool in all projects. The results from the CritICIS system show that local adaptation and versioning aspects are very important. Although the Gaston tools contains methods that facilitate 1) overriding national guidelines with local adaptations and 2) updating local guidelines by guideline authors without the assistance of knowledge engineers, the organization of institution must ensure that these tasks are also carried out. If not, too much false reminders may be given, which will decrease the system's acceptance dramatically.

In all projects, the Gaston execution engine was used as a DSS and interfaced with existing patient information. Two of the systems (GRIF and CritICIS) are used in daily practice. In all projects, the main bottleneck was interfacing the execution engine with the external patient information systems. The fact that the domain ontologies were often developed with the terminology of the patient information system in mind (e.g., the IMPACT ontology was used in CritICIS as well as in the PDMS) simplified the interfacing between the patient information systems and the Gaston execution engines. Existing standard ontologies will be harder to interface as there may exists syntactic as well as semantic differences between concepts from the ontology and concepts from the target information system [34]. Separating the Gaston execution engine into multiple components that each performs a different task (e.g., guideline inference, system interfacing and user communication) increased the reusability of the Gaston execution engine in multiple application domains and settings.

In conclusion, although the number of systems that were developed using the Gaston approach is still limited, the first experiences and results are very promising. The fact that Gaston covers the entire guideline development and implementation process and is supported by a number of generic tools related to the various phases in that process is one of the key elements that made it possible to reuse the approach in various projects. Although still a number of problems have to be addressed, especially related to standardizing, interfacing, organization and local adaptation, the foundation of Gaston is strong enough to build on further.

References

1. Van der Lei J, Talmon JL. Clinical Decision-Support Systems. In: Van Bommel and Musen (eds). Handbook of medical informatics. Houten: Bohn Stafleu Van Loghum, 1997.
2. Johnston ME, Langton KB, Haynes RB, Mathieu A. Effects of computer-based clinical decision support systems on clinician performance and patient outcome. A critical appraisal of research. *Ann Intern Med* 1994;120(2):135-42.

3. De Clercq PA, Blom JA, Korsten HHM, Hasman A. Approaches for Creating Computer-interpretable Guidelines that Facilitate Decision Support: a Review. Submitted for publication.
4. De Clercq PA, Hasman A, Blom JA, Korsten HHM. Design and implementation of a framework to support the development of clinical guidelines. *Int J Med Inform* 2001;64:285-318.
5. De Clercq PA, Hasman A, Blom JA, Korsten HHM. The Application of Ontologies and Problem Solving Methods for the Development of Shareable Guidelines. *Artif Intell Med* 2001;22(1):1-22.
6. Bindels R, De Clercq PA, Winkens RAG, Hasman A. A test ordering system with automated reminders for primary care based on practice guidelines. *Int J Med Inf* 2000;58-59(1):219-33.
7. De Clercq PA, Blom JA, Hasman A, Korsten HHM. A strategy for development of practice guidelines for the ICU using automated knowledge acquisition techniques. *Int J Clin Monit Comput* 1999;15:109-17.
8. Van Hyfte DM, de Vries Robbe PF, Tjandra-Maga TB, van der Maas AA, Zitman FG. Towards a more rational use of psychoactive substances in clinical practice. *Pharmacopsychiatry* 2001;34(1):13-8.
9. De Clercq PA, Hasman A. Design of a Consumer Health Record for Supporting the Patient-centered Management of Chronic Diseases. *Medinfo* 2001;10(2):1445-9.
10. Winkens RAG, Pop P, Bugter-Maessen AMA, Grol RPTM, Kester ADM, Beusmans GHMl. Randomised controlled trial of routine individual feedback to improve rationality and reduce numbers of test request. *Lancet* 1995;345:498-502.
11. Lamberts H, Wood M. *International Classification of Primary Care*. 3rd ed. Oxford: Oxford University Press; 1987
12. Engelbrecht R, Rector A, Moser W. Verification and validation. In: van Gennip E, Talmon J, editors. *Assessment and evaluation of information technologies in medicine*. Amsterdam: IOS Press; 1995.
13. Fieschi M. Towards validation of expert systems as medical decision aids. *International Journal of Bio-Medical Computing* 1990;26:93-108.
14. Blom JA. Temporal logics and real time expert systems. *Computer Methods and Programs in Biomedicine* 1996;51:35-49.
15. Bindels R, Winkens RAG, Pop P, van Wersch JWJ, Talmon J, Hasman A. Validation of a knowledge based automated feedback system for diagnostic test ordering in general practice. *Int J Med Inform* 2001;64:341-54.
16. Project IMPACT Page. SSCM Web Page. Available at http://www.sccm.org/impact/impact_home_set.html.
17. Lewis JR. IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction* 1995;7(1):57-78.
18. Linden M. Therapeutic standards in psychopharmacology and medical decision-making *Pharmacopsychiatry* 1994;27(1):41-5.
19. Van Hyfte D, Van Der Maas A, Tjandra-Maga T, De Vries Robbe P. A formal framework of knowledge to support rational psychoactive drug selection. *Artif Intell Med* 2001;22(3):261-75.
20. Erdman HP. A computer consultation program for primary care physicians. Impact of decision making model and explanation capability. *Med Care* 1987;25:138-47.
21. Rogers J, Roberts A, Solomon D, van der Haring E, Wroe C, Zanstra P, Rector A. GALEN ten years on: tasks and supporting tools. *Medinfo* 2001;10(1):256-60.

22. Van Hyfte DMH, de Clercq PA, Tjandra-Maga TB, Zitman FG, de Vries Robbé PF. Modelling the psychoactive drug selection application domain at the knowledge level. Proc Belgium-Netherlands Conf on Artificial Intelligence 1999;:187-8.
23. Van Hyfte DMH. Rational psychoactive drug selection: combining clinical, pathophysiological and pharmacotherapeutic knowledge within a patient-specific framework. PhD-thesis 2000. ISBN: 90-9013868-4
24. Brennan PF, Kuang Y.-S, Volrathongchai K. Patient-centered information systems. Yearbook of Medical Informatics 2000, Schattauer Verlag, Stuttgart. 2000: 79-86.
25. Maly RC, Bourque LB, Engelhardt RF, A randomized controlled trial of facilitating information giving to patients with chronic medical conditions: effects on outcomes of care. J Fam Pract 1999;48(5):356-63.
26. Shortliffe EH. The Evolution of Electronic Medical Records. Acad Med 1999;74(4):414-9.
27. The TANDEM project. Leonardo da Vinci Program DK/97/2/00376/PI/II.1.1.c/FPC. Homepage available at <http://www.tandem.v-chi.dk/>.
28. The Medical Guideline Technology project. INCO-COPERNICUS Project IC15 CT 98-0315. Homepage available at <http://frost.open.ac.uk/mgt/>.
29. Friedman CP, Wyatt JC. Evaluation methods in Medical Informatics. New York: Springer-Verlag 1997:205-54.
30. Hays RD, Wells KB, Sherbourne CD, Rogers W, Spritzer K. Functioning and well-being outcomes of patients with depression compared with chronic general medical illnesses. Arch Gen Psychiatry 1995;52(1):11-9.
31. National High Blood Pressure Education Program. The Sixth Report of the Joint National Committee on Detection, Evaluation, and Treatment of High Blood Pressure. Washington: NIH; 1998.
32. Lindberg C. The Unified Medical Language System (UMLS) of the National Library of Medicine. J Am Med Rec Assoc 1990;61(5):40-2.
33. Spackman KA, Campbell KE, Cote RA. SNOMED RT: a reference terminology for health care. Proc AMIA Symp 1997;:640-4.
34. Chandrasekaran B, Johnson TR, Smith JW. Task-Structure Analysis for Knowledge Modeling. Communications of the ACM 1992;35(9):124-37.

Appendix: CritICIS questionnaire

1 = *strongly agree*,
 2 = *somewhat agree*
 3 = *neutral*
 4 = *somewhat disagree*
 5 = *strongly disagree*

Usability

Agreement

Overall, I am satisfied with how easy it is to use CritICIS.	1	2	3	4	5
It was simple to use CritICIS.	1	2	3	4	5
I could effectively complete the tasks and scenarios using CritICIS.	1	2	3	4	5
I was able to complete the tasks and scenarios quickly using CritICIS.	1	2	3	4	5
I was able to efficiently complete the tasks and scenarios using CritICIS.	1	2	3	4	5
I felt comfortable using CritICIS.	1	2	3	4	5
I believe I could become productive quickly using CritICIS.	1	2	3	4	5
It was easy to understand the advices given by CritICIS.	1	2	3	4	5
The organization of information on the screens was clear.	1	2	3	4	5
It was easy to find the information I needed.	1	2	3	4	5
Whenever I made a mistake using CritICIS, I could recover easily and quickly.	1	2	3	4	5
CritICIS gave error messages that clearly told me how to fix problems.	1	2	3	4	5
The interface of CritICIS was pleasant.	1	2	3	4	5
I liked using the interface of CritICIS.	1	2	3	4	5

Training and Support

Training in the use of CritICIS was sufficient.	1	2	3	4	5
It was easy to get acquainted using CritICIS.	1	2	3	4	5
The manual of CritICIS was clear.	1	2	3	4	5
The help-function of in CritICIS was clear.	1	2	3	4	5
It was easy to find guideline-related information in CritICIS.	1	2	3	4	5
Technical support in the CritICIS project was sufficient.	1	2	3	4	5
Support regarding the content in the CritICIS project was sufficient.	1	2	3	4	5

User-satisfaction

Overall, I am satisfied with CritICIS.	1	2	3	4	5
Overall, I find CritICIS useful.	1	2	3	4	5
CritICIS generates correct reminders regarding most patients.	1	2	3	4	5
CritICIS generates the right amount of reminders.	1	2	3	4	5
Behavior change					
Working with CritICIS has changed my way of entering patient data.	1	2	3	4	5
Working with CritICIS makes me more aware on how to use patient data.	1	2	3	4	5
Working with CritICIS has limited the amount of entered patient data.	1	2	3	4	5
I prefer feedback before my actions rather than reminders afterwards.	1	2	3	4	5
I am prepared to encode patient information in ICIS for use in CritICIS.	1	2	3	4	5

Usefulness

I support the use of decision support systems in the ICU.	1	2	3	4	5
CritICIS is usable as a training-tool.	1	2	3	4	5
The patient will benefit from CritICIS.	1	2	3	4	5
I like to see CritICIS-like systems implemented in other departments.	1	2	3	4	5

CHAPTER 7

GENERAL DISCUSSION AND CONCLUSIONS

1 Introduction

The research described in this thesis aimed at exploring the potential of a generic methodology for the development and implementation of clinical guidelines with the purpose of providing decision support. In order to get more insight into the necessary steps that are required to reach the above-mentioned goal, a number of relevant research questions were postulated in **Chapter 1**. These questions address various difficulties, related to guideline representation (e.g., *'how to represent and share various types of guidelines using a formal and unambiguous representation'*), guideline acquisition (e.g., *'how to translate guidelines from a textual format into this formal representation'* and *'how to handle local adaptation and synchronization between (inter)national and local guidelines'*) and guideline-based decision support (*'how to interface guideline-based decision support systems with external patient information systems'* and *'how to provide decision support to a care provider in daily practice'*).

The remaining part of this chapter discusses various aspects related to these questions, after which a number of recommendations for future research and conclusions are presented.

2 The Gaston representation model

2.1 The frame-based formalism

Chapter 2 described a number of different approaches that have been developed during the last years. The models of these approaches are based on various formalisms such as rules, frames or description logic. The Gaston model is based on a frame-based representation, in which guidelines and Problem-Solving Methods (PSMs) are represented in terms of sequences of frame instances (e.g., primitives).

In the Gaston representation model, frames are used in two different ways: 1) to represent knowledge related to the application domain (domain ontologies) and 2) to represent knowledge related to the guideline's control structure (method ontologies). An advantage of using a frame-based representation is that it is an intuitive way of modeling knowledge and is commonly used to build ontologies. Also, the set of primitives (frames) can be easily extended, for example by adding new domain or method classes. Logic-based representations are often less intuitive, making it more difficult to build ontologies.

Frame-based formalisms also have their drawbacks. A logic-based representation is able to describe various kinds of relations between ontology elements such as *ForEvery*, *ThereExists*, *IsNot* and *Disjoint*. Not all of these relationships can easily be expressed in traditional frame-based representations. These restrictions are especially important when describing domain-specific knowledge such as drug interactions (although some of these relations can be expressed in a frame-based language, as shown in **Chapters 3 and 4**). These restrictions are less crucial for method ontologies, as method ontology primitives contain fewer of these types of relations compared to domain ontologies. A drawback of using frames to represent method ontology primitives is that these frames do not contain explicit procedural information, which can be interpreted by a generic interpreter such as PROLOG. In our model, each primitive has a custom-programmed procedure attached to it, which is only able to execute that specific primitive. Compared to description formalisms such as the PROforma L_{R2L} language, which allows for the processing of each primitive by a single generic interpreter, guideline verification in our model is more difficult as different procedures may use different (programming) languages to describe the primitive's procedural aspects.

However, we still chose a frame-based model as we felt (similar to the EON and GLIF approaches) that guidelines, represented in this way are understandable by humans as well as interpretable by automatic parsers. Another advantage of the Gaston frame-based model is the ability to introduce additional behavior in order to represent guidelines that differ in complexity and application domain (e.g., new primitives or PSMs). Also, the classes of the Gaston model can be used both to describe single guideline steps as well as the internal structure of PSMs.

Recently, various studies were performed in which a number of guideline approaches (among which Gaston) were compared [1, 2], including rule-based, frame-based and logic-based approaches. The comparison led to the identification of a number of common guideline components, which are also largely supported by the Gaston representation model. From the results of these studies, combined with the results of the projects that are described in this thesis, we argue that current languages such as Gaston are becoming powerful enough to capture the most important features, necessary for a guideline representation model. Finally, languages such as the Ontology Inference Layer (OIL) are currently being developed that combine aspects from frame-based and logic-based representations [3], which may be incorporated in the Gaston representation language.

2.2 Primitives vs. Problem Solving Methods

The examples, shown in this thesis, concern guidelines that differ in complexity as well as application domain. Although some (parts of) guidelines were represented through PSMs, primitives were more often used as building blocks, especially in complex guidelines such as the hypertension and weaning guidelines. Relatively simple strategies such as Situation Action Rules (SARs) and Event-Based Modular Tasks (EBMTs) were frequently represented by means of PSMs (**Chapter 3**). The more complex propose-and-revise PSM [4] was used only once in the MGT project.

As a guideline's control structure is usually very heterogeneous and built from collections of small diverse tasks, primitives seem to be more suitable as building blocks than PSMs. As a result, guidelines do not easily fit the predefined structures of a PSM. On the other hand, describing (parts of) guidelines by means of PSMs facilitates the authoring of guidelines by hiding the control structure and providing task-specific user interfaces, in which domain experts are able to specify relevant knowledge roles. This will increase the reusability and shareability of guidelines among different domains and applications.

Similar to PSMs in Gaston, the concept of using complex constructs that internally hide the control structure was also recently included in other approaches such as Macros in GLIF. We believe that, for certain classes of relatively simple PSMs, the application of such constructs will facilitate the acquisition of (complex) guidelines, when supported by user interfaces that visualize the constructs, understandable by guideline authors that may have little notion of the precise structure of the underlying representation. However, more PSMs have to be developed that describe guideline-specific tasks.

2.3 Domain-specific knowledge

The Gaston approach uses the concepts of domain ontologies, method ontologies and PSMs to separate knowledge that describes the control structure (e.g., decisions and actions) from knowledge that describes the application domain (e.g., used medications, treatments and contraindications).

Although the results from our projects show that it is possible to reuse different types of knowledge, applying a single domain ontology in various clinical specialties remains difficult. The IMPACT ontology in the CritICIS project and the ICPC ontology in the GRIF project for example, were especially developed for the intended application domain (e.g., ICU and family practice), which greatly simplified the development process. These ontologies were however difficult to interchange. For example, the IMPACT ontology was

not detailed enough to capture specific concepts that are commonly used in the field of family practice and vice versa.

Another difficulty is the mapping problem. For example, one guideline defined the concept 'age' whereas another guideline defined the concept 'date_of_birth', which have to be mapped in order to be interchangeable. Although this is a relatively simple one-to-one example, there also exist examples in which the mappings are less trivial [5]. We did not encounter such complex problems however, also because the ontologies in our projects were (partly) specifically designed for the application domain. The results of our projects led to the development of a standard domain ontology. This ontology consisted of a number of generic concepts (e.g., drugs, treatments, diagnoses, laboratory tests), in which each concept was defined by means of a standard set of attributes. For example, the concept 'drug' contained the attributes 'dose' and 'start_of_prescription', whereas the concept 'diagnosis' contained an attribute named 'diagnosis_date'. For particular domains, the domain ontology was extended with new classes and attributes. When we used this ontology to build the diabetes guideline for example, the diagnosis 'diabetes' was added, which, beside the standard 'diagnosis_date' attribute, contained an additional attribute 'type' that denoted the diabetes type (e.g., type I or type II). The development of large reusable ontologies will form a solution in the future. Therefore, current developments such as the development of the HL7 Reference Information Model (RIM) [6], combined with the development of standard terminologies such as SNOMED [7] will be of crucial importance. However, we also believe that for the time being, domain ontologies have to be adjusted, not only to the guideline's application domain but also to a guideline-based Decision Support System (DSS) that must be able to communicate with external patient information systems such as Electronic Patient Records (EPRs).

2.4 Local adaptation and communication

The above-mentioned difficulties are heavily related to another problem in guideline representation: the local adaptation and implementation of guidelines. The possibility to adapt the contents of (inter)national to local institutions is crucial for guideline acceptance [8]. This requires sophisticated versioning and synchronization mechanisms. On the one hand, local guideline authors must be able to override certain settings in the guideline in order to adapt the guideline to local standards. On the other hand, whenever the original guideline is updated, these updates must be reflected in the adapted guideline without losing earlier-made overrides. Therefore, the knowledge that describes the local adaptations must be separated from the knowledge that describes the original guideline.

Local guidelines not only differ with respect to the contents but also with respect to how these guidelines are used in an institution. For example, a local implementation of a guideline may lead to some form of communication with its users (e.g., generating a reminder on the screen of a nurse practitioner) whereas another implementation of the same guidelines may lead to another form of communication (e.g., sending a email to a physician). In order to specify 1) local adaptations of the guideline's contents and 2) details concerning the communication between a guideline-based DSS and external information systems and users, we argue that a guideline representation model must contain multiple layers, containing information related to the guideline's structure, global and local contents and communication.

Based on the results of the projects, described in this thesis, and other current projects, we have chosen to extend the two-layered approach that was already implemented in the KA-Tool (**Chapter 4**). Similar to the concept of knowledge roles that give an abstract description of the function (domain) knowledge plays, it is also possible to define different roles a guideline author may play during the guideline acquisition process. Examples are 1) the principal guideline author, who defines the control structure and initial contents of the guideline, 2) the local guideline author, who adapts the contents of the guideline to local standards and 3) the local information manager, who specifies communication and implementation details. Currently, we have implemented four layers in the KA-Tool:

- The *Structure* layer that describes the guideline control structure in terms of primitives and PSMs.
- The *Global Contents* layer that describes the contents of each primitive and PSM in terms of domain ontology concepts.
- The *Local Contents* layer that contains local adaptations of the contents of each primitive and PSM;
- The *Communication* layer that contains communication and implementation details such as the method of acquiring data from patient information systems or the form of communication (e.g., showing warning messages).

Figure 1 shows the KA-Tool that was used for the acquisition of weaning guidelines in the CritICIS project (**Chapter 6**), extended with two additional layers.

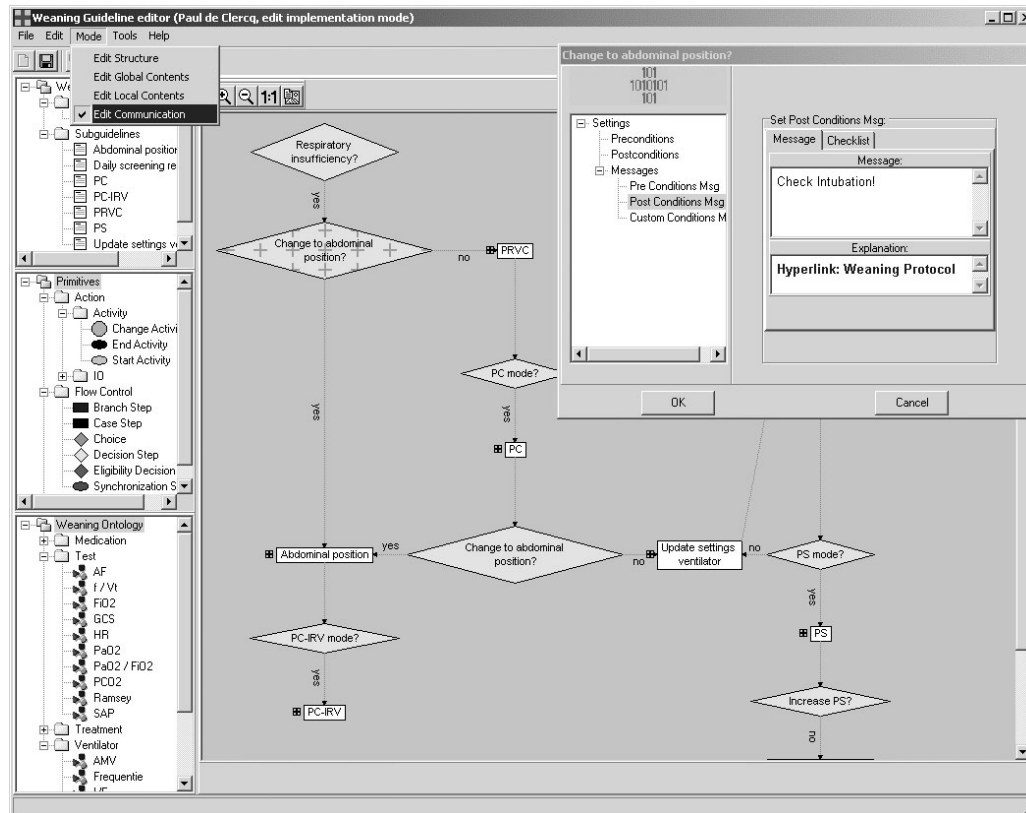


Figure 1: The CritICIS weaning KA-Tool, containing four layers

In this case, the *Communication* layer has been chosen (through the 'Mode' menu item), after which a guideline information manager has selected the 'Change abdominal position?' guideline step, shown in the top left corner of the guideline's control structure (the 'plus'-signs in this primitive indicate that additional details have been added). The communication details in this case consist of a message that is sent to the nursing staff when certain conditions apply (stored in the 'Postconditions' item). These conditions are for example used during guideline execution to check whether the current user is a nurse or a physician, as different messages may apply for different users.

Compared to the structure of the same weaning guideline, shown in figure 7 of **Chapter 6**, a number of primitives have been removed from the guideline's control structure in figure 1. These primitives such as 'Report intubation!' and 'Weaning response is increasing!' were instances of the 'Generate_Message' primitive, which was used to generate messages (e.g., reminders or advice) to users (**Chapter 3**). In the four-layered approach, these instances are now transferred to the *Communication* layer as they are related to the communication aspect of providing decision support. However, as they are still implemented as instances of the 'Generate-Message' primitive, the *Communication* layer is defined by concepts from the same representation

model that is used throughout the guideline. Technical implementation details (e.g., are messages shown on a computer screen or sent as HTML to a browser) are not specified in this layer as these are handled by the components of the DSS, described in **Chapter 4**.

Similar to *Global Contents* layer, the *Local Contents* layer also allows the specification of the contents of each primitive and PSM in terms of domain ontology concepts. However, content that is entered in the '*Local Contents*' mode may differ from content that is entered in the '*Global contents*' mode. For example, a hypertension guideline may contain a decision step primitive that decides whether a patient has a high blood pressure or not (**Chapter 5**). In the *Global Contents* layer, a high blood pressure can be defined as '*a blood pressure that is higher than 150/90*'. In the '*Local Contents*' layer however, a high blood pressure can be defined as '*a blood pressure that is higher than 155/95*'. Knowledge that is already present on the *Global Contents* layer (e.g., the definition of a high blood pressure) is copied to the *Local Contents* layer where it can be changed (e.g., redefining the high blood pressure). It is also possible to specify additional knowledge (e.g., adding a new criterion). This technique is similar to the inheritance of attributes in frames and classes, where items or values can be overwritten or added. However, the techniques that perform the synchronization and versioning between the different guideline layers are more complex than those, used in conventional object-oriented techniques.

Using layers to represent and store various kinds of guideline-related information separately has its pros and cons. One advantage is that the same set of primitives and PSMs as well as the underlying language can be used for all layers. Also, the results from our projects showed that guideline authors have the tendency to include primitives that contain decision support-related information in the guideline's control structure (e.g., the '*Generate_Message*' primitive), which may be favorable in some cases such as the application of SARs in the CritICIS and GRIF projects (in these cases, the *Communication* layer is omitted). However, ultimately this information should not be stored in the guideline's control structure. As each layer uses the same primitives and PSMs, it facilitates transferring these concepts from the *Structure* layer to other layers.

Besides the weaning protocol, we are currently applying the same four-layered approach in a project for the development and implementation of clinical trial oncology guidelines aimed at the treatment of Acute Myelogenous Leukaemia (AML) patients [9].

3 The Gaston guideline development environment

3.1 Task-specific user interfaces

Besides serving as building blocks for guidelines and PSMs, another reason of using a frame-based model is related to knowledge acquisition. As primitives and PSMs symbolize certain (sub)tasks in a guideline (e.g., decision, action, selection), a specific user interface corresponds with each primitive or PSM that can be embedded as a separate module in the KA-Tool. This shortens the development time of new primitives and improves the acceptance by guideline authors.

As mentioned earlier, the real advantages of using PSMs are related to knowledge acquisition. For each PSM, a guideline author only has to fill in the relevant knowledge roles by selecting the proper concepts from the domain ontology, while the internal structure of the PSM is completely hidden. It is questionable however, whether the same technique can be applied with more complex PSMs.

3.2 Guideline verification

Apart from *PROforma*, most approaches do not consider guideline verification as one of their top priorities. Although guideline verification is an important part of the guideline development process, it is understandable that the issues of guideline representation, acquisition and execution have more priority in projects that are also usually constrained by time and resources. It is natural that first it must be established *whether* a computer-based guideline will function in practice, before it can be established *how* a computer-based guideline will function in practice.

The verification methods used in the Gaston approach also are still limited: although we have applied methods for the detection of various logical and procedural errors (**Chapters 5 and 6**), these were only tested on guideline knowledge bases that mainly consisted of SARs such as the guidelines in the CritICIS and GRIF projects.

Using drag-and-drop techniques that enabled authors to specify the control structure and contents of a guideline by means of selecting, configuring and combining concepts from method and domain ontologies prevents the guideline authors to make syntactic errors. This, in contrast to approaches where the knowledge is entered as text (e.g., the criteria in the Arden syntax or GLIF). Naturally, semantic errors can still be made, as was shown in the CritICIS project (**Chapters 5 and 6**). Although some of these errors can be

detected by automated verification tools, simulation environments where guidelines can be tested against actual patient data are crucial.

3.3 Guideline execution tasks

Guideline-based decision support usually receives less attention than guideline representation and acquisition. However, we believe that developing, implementing and evaluating DSSs in daily practice will improve the acceptance of computer-based guidelines by health-care workers considerably.

Similar to the tasks a health-care worker usually carries out in order to solve a problem, DSSs that were developed and applied in our projects are able to perform four basic tasks: 1) recognize relevant events from the outside world, 2) make assessments based on available knowledge, 3) retrieve more information when necessary and 4) perform certain actions when necessary. All tasks that are described in a guideline have to be carried out by one or more of the four DSS tasks. For example, a warning message that is specified in an action step can directly be carried out by the *'perform certain actions when necessary'* DSS task. Sometimes, tasks that are specified in a single guideline primitive have to be carried out by more than one DSS task. For example, a decision step may contain the criterion *'is the patient's blood pressure too high'*. When this decision step is executed by a DSS, it executes two tasks. First, the criterion is evaluated, which is carried out by the *'make assessments based on available knowledge'* task. When during the execution of this criterion, the value of the patient's blood pressure is required, the *'retrieve more information when necessary'* task is executed. As mentioned earlier, we have defined different types of guideline developers such as the principal and local guideline author, who define the control structure and contents of the guideline, and the local information manager, who specifies communication and implementation details. In this case, the (principal or local) guideline author specifies information that will be carried out by the *'make assessments based on available knowledge'* DSS task, in contrast to the local information manager, which specifies information that will be carried out by the *'retrieve more information when necessary'* DSS task. During the guideline authoring process, information that is relevant only to the guideline author has to be hidden from the local information manager and vice versa.

It is possible to develop guideline primitives that can be directly mapped to corresponding DSS tasks. For example, each of the four tasks of the PROforma task ontology corresponds with one of our four DSS tasks. However, this implies that the control structure of a guideline explicitly consists of primitives that are relevant to the guideline author (e.g., the

PROforma decision task) as well as the local information manager (e.g., the PROforma enquiry task). This makes it more difficult to separate these types of information from the various guideline developers.

The Gaston framework uses another approach by using the earlier-mentioned layers: all communication-related information is stored into a *Communication* layer, which is separated from the layers that contain inference-specific information. The advantage of the PROforma approach is that the tasks that are specified in a guideline primitive can be directly carried out by a DSS. In our approach, the tasks in the *Communication* layer still have to be mapped onto corresponding DSS tasks. The disadvantage of the PROforma approach is that the guideline representation language is very 'low-level', which may cause problems for guideline authors as well as local information managers during the acquisition phase.

3.4 Interfacing external patient information systems

One of the great challenges will be interfacing a single DSS with multiple patient information systems such as EPRs, consumer health record systems and patient monitoring systems. DSSs must be able to react on events or initiate a conversation to acquire data. For example, the current versions of the CritICIS and TANDEM systems acquire data from two different patient information systems simultaneously. Whenever the CritICIS system receives an event from the PDMS (e.g., an antihypertensive drug is being prescribed), it queries a monitoring system in order to acquire the patient's real-time blood pressure and breathing frequency.

Not all patient information systems are able to provide flexible communication interfaces with a DSS such as Gaston (this is especially true for legacy systems). If such interfaces do exist, retrieving data often consumes a lot of time. Also, during execution (intermediate) conclusions such as '*based on the patient's recent blood pressures, (s)he is diagnosed with hypertension*' must be sent back to the patient information system or stored locally. A (partial) solution is that the DSS must also be able to store patient data by itself, for example in a self-managed patient database. An advantage is that, when a DSS is executed multiple times, it is able to 1) cache recent patient data to speed up the process and 2) store intermediate conclusions (e.g., '*the patient is diagnosed with hypertension*') or other patient-related information (e.g., '*this is the patient's third visit*'), which can be reused during subsequent executions. However, this requires a synchronization mechanism between the DSS and external patient information systems in order to keep the various types of patient information up-to-date.

3.5 Decision support

As the name indicates, the aim of a guideline-based DSS is to provide support to health-care workers (e.g., physicians, nursing staff and patients). As already mentioned, we argue that communication-related information must be separated from other guideline-related information (e.g., inferencing). The advantage of this method is that various communication methods can be applied with respect to the same guideline such as proactive communication (e.g., ‘guide’ the user actively through the guideline) or reactive communication (e.g., critique the user whenever a guideline is not followed). Each of these methods has its pros and cons. The advantage of a reactive DSS such as the CritICIS and GRIF system is that it does not interfere with the daily work of the care providers [10]. The advantage of a proactive system such as the MGT hypertension advisory system is that it can suggest the best-known treatment, for example to novice users. When guideline-based DSSs contain additional institution-specific information related to workflow management (e.g., which tasks are to be carried out by physicians and which by the nursing staff), they could also be used as workflow management systems [11].

4 Current and future research

4.1 Representation

Certain parts of the Gaston representation can still be improved, especially concerning topics related to uncertainty, temporal logic and intentions. For example, current Gaston projects focus on the integration of Asbru’s intentions [12] and temporal logic and the possible use of aspects from other approaches such as the PAL and L_{R2L} languages in EON and PROforma.

The syntax of the underlying language has not yet formally been written down, mainly as a result of time constraints. Also, the description of PSMs on the task level is still informal. Consequently, there is no automatic mapping from the PSM’s task description to a corresponding control structure. We are currently investigating the integration of formal PSM languages within our framework to make such mappings explicit. For this purpose, we will look at more complex PSMs than the ones described in this thesis.

4.2 Verification

As shown in the CritICIS and GRIF projects, a number of methods were used to verify rule-based guideline knowledge bases. We are currently expanding these tests in order to detect other primitive-related errors such as ‘*a drug cannot be discontinued before it is prescribed*’ or ‘*every step following a branch step must eventually lead to a corresponding synchronization step*’.

4.3 Integration

An important issue is the development and implementation of standardized interfaces between patient information systems (e.g., EPRs) and guideline-based DSSs, taking into account the existence of multiple domain ontologies and terminologies, patient information systems and local organizations. We have developed a number of standard interface components, which are able to communicate with various types databases and information systems (e.g., ODBC, HL7 version 2, XML). New standards that are currently being developed must be incorporated in the modern information systems as well as DSSs.

Current Gaston projects focus on integrating a number of these new standardized communication interfaces and terminologies such as interfaces and terminologies developed by the OMG Healthcare Domain Task Force (formerly known as CORBAMed) [13] and the earlier-mentioned HL7 [6] and SNOMED [7] groups. Most of these are still under development, however.

5 Conclusions

This thesis presented the Gaston approach: a methodology and accompanying framework for the development and implementation of guideline-based decision support systems. The various projects, described in this thesis, showed that this framework was generic and flexible enough to be reused in different medical and applications domains. This is supported by the fact that two developed DSSs are still being used in daily practice.

Similar to other approaches, the Gaston frame-based formalism was able to represent guidelines that differ in complexity and application domain by means of primitives as well as (relatively simple) PSMs. Current developments will probably lead to a representation that contains elements from frame-based and logic-based approaches.

The various projects in this thesis showed that it was possible to reuse the same KA-Tool in all projects. Advantages of using PSMs are heavily related to knowledge acquisition. PSMs facilitate the acquisition of (complex) guidelines, supported by user interfaces that visualize the constructs, which are understandable by guideline authors that may have little notion of the precise structure of the underlying representation. On the other hand, primitives will always be necessary for building custom-tailored guidelines.

Guidelines contain various types of knowledge such as domain-, procedural- and implementation-specific knowledge. The possibility of specifying local

adaptations and implementation details is crucial for the acceptance of guidelines by individual institutions. The verification of all these types of knowledge will become more important in the future.

To improve the acceptance of guideline-based DSSs, the greatest challenge concerns the implementation of actual DSSs in daily practice, focused on the areas of developing standardized terminology mapping and interfacing techniques. Results of current developments such as OMG HDTF, HL7, SNOMED and UMLS will be essential, just as the results of implementations of locally adapted DSSs. As a result of the vast increase of clinical knowledge in general and clinical guidelines in particular, generic frameworks such as Gaston will be of crucial importance for the use and acceptance of these guidelines in daily practice.

References

1. Wang D, Peleg M, Tu SW, Boxwala AA, Greenes RA, Patel VL, Shortliffe EH. Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines: A literature review of guideline representation models. *Int J Med Inf* 2002;68(1-3):59-70.
2. Peleg M, Tu S, Bury J, Ciccarese P, Fox J, Greenes RA, Hall R, Johnson PD, Jones N, Kumar A, Miksch S, Quaglini S, Seyfang A, Shortliffe EH, Stefanelli M. Comparing computer-interpretable guideline models: a case-study approach. *J Am Med Inform Assoc* 2003;10(1):52-68.
3. Fensel D, Harmelen F, Horrocks I, McGuinness D, Patel-Schneider PF. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems* 2001;16:38-45.
4. Marcus S. Automated knowledge acquisition for expert system. Norwell: Kluwer Academic Publishers, 1988.
5. Chandrasekaran B, Johnson TR, Smith JW. Task-Structure Analysis for Knowledge Modeling. *Communications of the ACM* 1992;35(9):124-37.
6. Schadow G, Russler DC, Mead CN, McDonald CJ. Integrating Medical Information and Knowledge in the HL7 RIM. *Proc AMIA Symp* 2000;:764-8.
7. Spackman KA, Campbell KE, Cote RA. SNOMED RT: a reference terminology for health care. *Proc AMIA Symp* 1997;:640-4.
8. Fridsma DB, Gennari JH, Musen MA, Making Generic Guidelines Site-Specific. *Proc AMIA Symp* 1996;:597-601.
9. Van Oosterhout EMW, Talmon JL, De Clercq PA, Schouten HC, Jansen MPF, Hasman A. The PropeR way to support medical doctors in daily practice. Part I: developing the Protocol Based DSS. *MIE* 2003. Accepted for publication.
10. Miller RA, Masarie FE Jr. The demise of the "Greek Oracle" model for medical diagnostic systems. *Meth inform Med* 1990;29:1-2.
11. Quaglini S, Stefanelli M, Cavallini A, Micieli G, Fassino C, Mossa C. Guideline-based careflow systems. *Artif Intell Med* 2000;20(1):5-22.
12. Advani A, Lo K, Shahar Y. Intention-based critiquing of guideline-oriented medical care. *Proc AMIA Symp* 1998;:483-7.
13. Object Management Group. Healthcare Domain Task Force. Homepage available at <http://www.omg.org/healthcare>.

SUMMARY

During the last decade, studies have shown the benefits of using clinical guidelines in the practice of medicine such as reduction of practice variability and patient care costs, while improving patient care. A variety of guidelines have been developed that focus on different phases of the patient care process (e.g., patient screening, diagnosis, workup, referral and management), application domains (e.g., disease management, protocol-based care and consultation), and modes of use (e.g., clinical reference, knowledge source, education, quality assurance).

The project, described in this thesis, aims at answering the question: '*how to represent, acquire and implement computer-based guidelines*' by describing the development and evaluation of a generic approach that addresses various aspects related to the guideline development process. The approach has led to the development of the Gaston framework, which is described and discussed in this thesis.

Chapter 1 presents an overview of the research area of this thesis: the development and application of computer-based guidelines and guideline-based decision support. Although the potential application of guidelines as a form of decision support is enormous, a number of difficulties arise when developing and implementing guidelines in daily care. A major problem is related to the fact that guidelines are often represented as (structured) textual documents, which is a passive form of decision support: the care provider must decide whether consultation of a guideline is necessary. Often, care providers are convinced that their actions agree with guideline standards and that there is no need to consult the corresponding guideline in order to be sure. In reality however, these actions may oppose the guideline's intentions.

Implementing guidelines in active computer-based decision support systems promises to improve the acceptance and application of guidelines in daily practice because the actions and observations of care providers are monitored and advice is generated whenever a guideline is not followed. Various studies, covering a wide range of clinical settings and tasks, showed that the use of these systems significantly improves the quality of care, especially when used in combination with clinical information systems such as Electronic Patient Records. It is stated that these decision support systems are in fact not only crucial elements in long-term strategies for promoting the use of guidelines but are also necessary for the future of medical decision making in general.

Chapter 2 explores existing approaches by means of literature review. This chapter describes and discusses existing approaches and postulates a

number of functional requirements that form a basis for the development of the Gaston approach.

Based on these requirements, chapters 3 and 4 describe the methods used to develop the Gaston approach, which consists of a methodology for the development and implementation of computer-based guidelines and guideline-based decision support systems. Chapter 3 focuses on the aspect of guideline representation. It describes a new guideline representation formalism that is based on the concepts of ontologies, primitives and Problem-Solving Methods, which aims at improving the acceptance of guidelines. Chapter 4 describes the Gaston framework: a framework that facilitates the development and implementation of computer-based guidelines and guideline-based decision support systems. This chapter describes a guideline authoring environment that enables guideline authors to define guidelines in terms of the developed representation model and a guideline execution environment that is able to execute guidelines and interfaces with external patient information systems.

In order to evaluate whether the Gaston approach was able to facilitate the development and implementation of guideline-based decision support systems in different medical and application domains, a number of decision support systems were developed. These systems and experiences with the development of these systems are discussed in chapters 5 and 6. Chapter 5 describes and discusses the CritlCIS system that was created by means of the Gaston approach. This system was developed for use in Intensive Care Units (ICUs) and provided decision support to ICU health care workers by means of generating reminders when guidelines were not followed. Chapter 6 describes the experiences with a number of other systems that were developed with the Gaston approach in the areas of family practice, psychiatry and chronic disease management.

From the results of these projects, Chapter 7 concludes that the Gaston approach and framework was generic and flexible enough to be reused in different medical and applications domains, supported by the fact that various developed decision support systems are still being used in daily practice. It also concludes that, as a result of the vast increase of clinical knowledge in general and clinical guidelines in particular, generic approaches such as Gaston will be of crucial importance for the use and acceptance of clinical guidelines in daily practice.

SAMENVATTING

Gedurende het laatste decennium hebben studies de voordelen aangetoond van klinische richtlijnen in de dagelijkse zorgpraktijk zoals een afname van de zorgvariabiliteit en de kosten voor patiëntenzorg. Een verscheidenheid aan richtlijnen is ontwikkeld die zich richt op verschillende fasen van het proces van patiëntenzorg (b.v. screening, diagnose, workup, doorverwijzing en management van patiënten), applicatiedomeinen (b.v. ziekte management, zorg gebaseerd op protocollen en consultatie), en verschillende gebruiksvormen (b.v. klinische referenties, bron van kennis, onderwijs, kwaliteitsgarantie).

Het project, zoals beschreven in dit proefschrift, richt zich op het beantwoorden van de vraag: 'hoe computer-gebaseerde richtlijnen te representeren, verkrijgen en implementeren' door het beschrijven van de ontwikkeling en evaluatie van een generieke methodologie die gebruik maakt van verscheidene aspecten gerelateerd aan het richtlijn-ontwikkelproces. De methode heeft geleid tot de ontwikkeling van het Gaston framework dat beschreven en bediscussieerd wordt in dit proefschrift.

Hoofdstuk 1 geeft een overzicht van het onderzoeksgebied dat in dit proefschrift bestreken wordt: de ontwikkeling en toepassing van computer-gebaseerde richtlijnen en op richtlijn gebaseerde beslissingsondersteuning. De mogelijke toepassingen van richtlijnen als beslissingsondersteuning zijn enorm. Echter, er ontstaan een aantal problemen wanneer deze richtlijnen ontwikkeld en geïmplementeerd worden in de dagelijkse zorg. Een belangrijk probleem is dat richtlijnen vaak bestaan uit (gestructureerde) tekstdocumenten; een passieve vorm van beslissingsondersteuning: de zorgverlener moet namelijk beslissen of het raadplegen van de richtlijnen noodzakelijk is. Vaak zijn zorgverleners overtuigd dat hun acties overeenkomen met richtlijnstandaarden en dat het niet nodig is om overeenkomstige richtlijnen te raadplegen. De realiteit is echter dat de gepleegde acties niet altijd in overeenstemming zijn met datgene wat in de richtlijnen aanbevolen wordt.

Het implementeren van richtlijnen in actieve, op computer gebaseerde, beslissingsondersteunende systemen zal de acceptatie en toepassing van richtlijnen in de dagelijkse zorg verbeteren, omdat hierdoor de acties en observaties van zorgverleners geobserveerd worden en advies wordt gegeven indien een richtlijn niet nageleefd wordt. Verscheidene studies, met betrekking tot klinische settings en taken lieten zien dat het gebruik van deze systemen een significante verbetering van de dagelijkse zorg biedt, zeker wanneer deze systemen gebruikt werden in combinatie met klinische informatie systemen zoals Elektronische Patiënten Dossiers. Deze

beslissingsondersteunende systemen zijn feitelijk niet alleen cruciale elementen in lange-termijn strategieën voor het promoten van het gebruik van richtlijnen maar zijn ook noodzakelijk voor de toekomst van het ‘medische beslissingen nemen’ in het algemeen.

Hoofdstuk 2 bevat een literatuurstudie van bestaande benaderingen. Dit hoofdstuk beschrijft en bediscussieert reeds bestaande toepassingen en postuleert een aantal functionele eisen dat de basis van de Gaston benadering vormt.

Aan de hand van deze eisen beschrijven hoofdstuk 3 en 4 de methoden die geleid hebben tot de ontwikkeling van de Gaston benadering, bestaande uit een methodologie voor het ontwikkelen en implementeren van computer-gebaseerde richtlijnen en op richtlijnen gebaseerde beslissingsondersteunende systemen. Hoofdstuk 3 richt zich op de representatie van richtlijnen. Er wordt in dit hoofdstuk een nieuw ontwikkeld richtlijn-representatieformalisme beschreven, gebaseerd op de concepten van ontologieën, primitieven en Probleem-Oplos-Methoden, dat zich richt op het verbeteren van de acceptatie van richtlijnen. Hoofdstuk 4 beschrijft het Gaston framework: een framework dat de ontwikkeling and implementatie van op richtlijnen gebaseerde beslissingsondersteunende systemen ondersteunt. Dit hoofdstuk beschrijft een richtlijn-ontwikkelomgeving waarmee richtlijnauteurs in staat worden gesteld om hun richtlijnen te definiëren in termen van het ontwikkelde representatiemodel, en een richtlijn-executieomgeving die in staat is om richtlijnen uit te voeren en te communiceren met externe patiënteninformatie systemen.

Om te evalueren of de Gaston benadering in staat is om de ontwikkeling en implementatie van op richtlijn gebaseerde beslissingsondersteunende systemen in verschillende medische toepassingsdomeinen te vergemakkelijken is een aantal beslissingsondersteunende systemen ontwikkeld met behulp van Gaston. Deze systemen en de ervaringen die opgedaan zijn tijdens het ontwikkelen van deze systemen worden bediscussieerd in de hoofdstukken 5 en 6. Hoofdstuk 5 beschrijft en bediscussieert het eerste systeem dat ontwikkeld is met de Gaston benadering. Dit systeem, CritICIS genaamd, is ontwikkeld voor gebruik in Intensive Care Units (ICUs) en verzorgt beslissingsondersteuning voor ICU zorgverleners door middel van het generen van ‘reminders’ wanneer bepaalde richtlijnen niet worden gevolgd. Hoofdstuk 6 beschrijft de ervaringen die opgedaan zijn met een aantal andere systemen ontwikkeld met de Gaston benadering, in de specialismen huisartsgeneeskunde, psychiatrie en behandeling van chronisch zieke patiënten.

Uit de resultaten van deze projecten is geconcludeerd dat de Gaston benadering en bijbehorend framework generiek en flexibel genoeg zijn voor hergebruik in verschillende medische toepassingsgebieden, te meer daar verschillende beslissingsondersteunende systemen nog steeds gebruikt worden in de dagelijkse zorg. Ook is geconcludeerd dat als gevolg van de snelle toename van medische kennis in het algemeen en klinische richtlijnen in het bijzonder, generieke toepassingen zoals Gaston een cruciale rol zullen gaan spelen bij het gebruik en de acceptatie van klinische richtlijnen in de dagelijkse zorg.

DANKWOORD

Ook al is de voornaamste schrijver en samensteller van een proefschrift de persoon wiens naam op de omslag staat, vaak hebben meerdere personen op verschillende wijzen bijgedragen aan het uiteindelijke resultaat. Ook dit boekje zou nooit totstandgekomen zijn zonder de bijdrage van velen. Aan al diegenen: ontzettend bedankt!

Graag wil ik een aantal mensen speciaal bedanken. Als eerste natuurlijk mijn beide promotoren: Arie Hasman en Erik Korsten. Ze hebben samen laten zien dat de twee onderzoeksgebieden die tezamen Medische Informatica vormen uitstekend met elkaar te combineren zijn. Arie wil ik hierbij hartelijk bedanken voor de onvermoeibare ondersteuning en het geduld gedurende de afgelopen jaren, vooral als ik (soms na een tussenliggende periode van enkele maanden) aankwam met versie 9.2 van hetzelfde artikel waarin de helft van de voorgestelde wijzigingen weer ongedaan was gemaakt. Verder heeft hij me (zoals het een goede promotor betaamt) ook nog andere essentiële levensvaardigheden bijgebracht, bijvoorbeeld met betrekking tot de commercie: dankzij zijn uitstekende onderhandelingen met een tuk-tuk bestuurder heb ik in drie uur tijd zes verschillende juwelierszaken in Bangkok gezien om daarna in een uithoek achtergelaten te worden. Arie, bedankt voor de steun en het geloof de afgelopen jaren!

Ook Erik wil ik hartelijk bedanken voor zijn niet aflatende enthousiasme en steun de afgelopen jaren. Zonder hem zouden de dingen die we samen hebben bedacht nooit in de praktijk geïmplementeerd zijn. Zijn steeds terugkerende uitspraak 'we moeten nu toch echt regels gaan maken' zal me waarschijnlijk nog jarenlang blijven achtervolgen. Ambassadeurs zoals Erik zijn noodzakelijk voor het welslagen van elk project waarin zowel universiteiten als zorginstellingen deelnemen. Verder is dankzij Erik mijn bewondering voor de Beatles omgeslagen in een lichte aversie en weet ik nu hoe ik steenkastelen moet bouwen op het strand van San Francisco. We zullen elkaar waarschijnlijk de komende jaren nog vaak tegen het lijf lopen. Misschien komt het tennissen er nu eindelijk een keer van!

Verder wil ik mijn copromotor Hans Blom bedanken. Hij heeft ervoor gezorgd dat ik de praktische kant van het onderzoek niet uit het oog verloren ben en heeft ook borg gestaan voor de kwaliteit van mijn publicaties. Helaas is het je niet gelukt om mij van het 'komma syndroom' af te helpen, maar je kunt natuurlijk niet alles hebben.

Alle personen die betrokken zijn geweest bij mijn promotietraject wil ik natuurlijk ook niet vergeten. Als eerste de verschillende artsen en verpleegkundigen in het Catharina ziekenhuis die hebben bijgedragen aan het

onderzoek, met name Arnoud Roos, Alex Bindels, Hanny Megens en Jan van de Berk. Op de tweede plaats wil ik graag de mensen van Stanford Medical Informatics bedanken voor de gastvrijheid die zij me geboden hebben, met name Mark Musen, Mor Peleg en Samson Tu voor de vriendelijkheid en de vele nuttige en leuke discussies die we gevoerd hebben.

Verder natuurlijk de vele TU/e kamergenoten die ik de laatste zes jaar hier versleten heb, waarvan ik bang ben dat ik ze niet eens allemaal onthouden heb. Speciaal wil ik Harald en Susanne noemen voor hun belangstelling, hulp en commentaar gedurende het afgelopen jaar. Susanne, bedankt dat je mijn paranimf wil zijn (en natuurlijk voor de speciale theekan)!

Natuurlijk wil ik hier ook mijn tweede paranimf, Rianne, bedanken. Samen hebben we laten zien dat onze hersenspinsels ook echt in praktijk gebruikt kunnen worden. Verder wil ik je bedanken voor het prettige gezelschap tijdens alle installaties en de verschillende reizen naar het buitenland per trein, boot en auto (per auto soms zelfs meerdere op één dag). Je hebt me wel ingehaald wat de promotiedatum betreft maar dat gun ik je van ganzen harte!

Ook privé is zeven jaar een lange tijd geweest waarin verschillende omslagen hebben plaatsgevonden. Voor wat betreft de eerste jaren wil ik Annemiek bedanken voor de steun die ze me in die tijd gegeven heeft. Verder wil ik alle personen bedanken voor de geboden hulp of getoonde belangstelling tijdens de afgelopen jaren. Ik heb me voorgenomen om jullie in mijn toekomstige privé-leven vaker te gaan zien. Nick, jou wil ik natuurlijk bedanken voor je hulp bij het ontwerpen van de kافت. Speciaal wil ik ook Inge noemen, waarbij ik de laatste periode altijd terechtkon indien nodig: hoe vaker we vanaf nu gaan tennissen of snookeren, hoe minder tijd ik heb om te gaan werken!

Vanzelfsprekend ontbreken hier mijn ouders niet die altijd achter me gestaan hebben. Zonder jullie zou dit proefschrift er nooit zijn gekomen. Jullie hebben, samen met Eric, me altijd onvoorwaardelijk gesteund, welke keuzes ik ook maakte. Zonder jullie drieën zou ik nooit staan waar ik nu sta: dank voor alles!

Als laatste natuurlijk Anouk. Ik hoop echt dat onderzoeken zoals deze ooit een verschil zouden kunnen betekenen voor jou of anderen in de toekomst. Ik wil je bedanken voor al het plezier dat we, ook al is het nog maar een korte tijd geweest, samen al hebben gehad en ik weet zeker dat dit plezier de komende jaren alleen nog maar toe zal gaan nemen!

Paul de Clercq,
Breda, april 2003.

CURRICULUM VITAE

Paul de Clercq was born in Eindhoven, the Netherlands on August 9, 1969. He received his undergraduate education at the Bisschop Bekkers College in Eindhoven (Atheneum-B) from 1981-1987.

In 1992, he obtained a bachelor degree in Technical Physics at the Hogeschool Eindhoven with his work entitled 'Development of Chaos Theory Applications for Educational Purposes'.

In 1996, he obtained his masters degree in Electrical Engineering at the Eindhoven University of Technology with his work entitled 'Implementing a Critiquing System to Provide Decision Support in the ICU: the CritICIS System'.

In October 1996, he started his PhD study reported in this thesis at the Signal Processing Systems group of Eindhoven University of Technology.

From 2001 to 2002, he worked as a researcher for the Medical Informatics group at Maastricht University where he developed decision support systems and patient-centered information systems.

In 2002, he founded the company 'Medecs', which aims at the development and application of clinical decision support systems.

He currently lives in Breda and is the father of a daughter named Anouk.