

THE PRECISE CONSTRUCTION OF PATIENT PROTOCOLS:  
MODELING, SIMULATION AND ANALYSIS OF  
COMPUTER INTERPRETABLE GUIDELINES

By

Janos Laszlo Mathe

Dissertation

Submitted to the Faculty of the  
Graduate School of Vanderbilt University  
in partial fulfillment of the requirements  
for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

December, 2012

Nashville, Tennessee

Approved

Professor Gabor Karsai<sup>1</sup>

Professor Gautam Biswas<sup>1</sup>

Professor Xenofon D. Koutsoukos<sup>1</sup>

Professor Bradley A. Malin<sup>2</sup>

Professor Douglas C. Schmidt<sup>1</sup>

<sup>1</sup> Department of Electrical Engineering and Computer Science, Vanderbilt University

<sup>2</sup> Department of Biomedical Informatics, Vanderbilt University

## **ABSTRACT**

Standardizing the care of patients with complex problems in hospital settings is a difficult challenge for physicians, nurses and other medical professionals. Diverse conditions further complicate patient management. While in acute care settings such as intensive care units, the inherent problems of stabilizing and improving vital patient parameters is further complicated by the division of responsibilities among different individuals and teams, in outpatient settings the management of chronic diseases introduces additional complications related to the long-term treatment of patients. The use of evidence-based guidelines for managing complex clinical problems has become the standard of practice. Computerized support for implementing such guidelines has tremendous potential; however, addressing this problem requires a carefully coordinated use of various techniques from the field of computer science, as guidelines developed by the medical community are not directly interpretable by computers.

In this thesis, first, we present a survey of literature and a study on the open questions from the field of clinical decision support focusing on the use of model-based techniques for specifying and implementing evidence-based guidelines. Following the survey, we describe a model-based architecture for enabling the construction, management, verification and execution of such guidelines. The presented architecture is model-based in the sense that it relies upon the formal modeling of medical guidelines, including the specification of input parameters such as signs and symptoms, output parameters such as medical actions, and other guideline-related constraints such as rules, regulations and policies. The behavioral semantics of these models is provided by the application of custom-built formal behavioral templates defined with the help of Matlab Simulink/Stateflow and model composition. The benefits of our approach are illustrated with the modeling, execution and formal analysis of a clinically relevant example, a sepsis management guideline.

## **Keywords**

Executable medical guidelines, Model-based development, Patient workflow management system, Design languages, Domain-specific architectures, Medical information systems, Modeling, Ontology

I dedicate this thesis for my family,  
who offered me unconditional love and support all the way since the beginning of my studies,  
including throughout the course of this thesis.

## ACKNOWLEDGMENTS

There are a number of people without whom this thesis might not have been written, and to whom I owe an immense debt of gratitude.

To my advisor, Professor Gabor Karsai, who from the formative stages of this thesis, to the final draft, provided me with his sound advice and careful guidance, which were invaluable as I attempted to transform myself into the researcher I am today.

To my former advisor, now father-in-law, Professor Janos Sztipanovits, who inspired me to pursue the Ph.D. candidacy in the first place. He not only handled this unique transition of roles professionally and gracefully, but also, while being my advisor, provided me with the complex and fascinating projects that allowed me to make a mark in my field. He is a person, who never stopped encouraging me and enlightening me when my knowledge was lacking.

To many of my collaborators in the STEEP project, including Andras Nadas, Professor Bradley Malin and the experts from the Medical Center and the Informatics Center at Vanderbilt University, without whom the STEEP project would have not been possible.

To my wife, Dora, who had the patience to endure my occasional self-doubts, allowed me to have the time to complete my work, and surrounded me with everlasting love and positivity.

At last, but not at least, to my family, especially to my parents, Janos and Eva and to my brother, Daniel, who even though they missed me back in Hungary, have motivated and supported me throughout the process.

This work was supported in part by TRUST (Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: AFOSR (#FA9550-06-1-0244), BT, Cisco, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia, and United Technologies.

Support also came from the Vanderbilt HealthTech Laboratory, a research facility of Vanderbilt Medical Center, the Division of Allergy, Pulmonary and Critical Care Medicine in the Department of Medicine of Vanderbilt Medical Center, the Emergency Department of Vanderbilt University Hospital and the Informatics Center of Vanderbilt Medical Center.

To each of the above, I extend my deepest appreciation.

## TABLE OF CONTENTS

ABSTRACT.....	II
DEDICATION .....	III
ACKNOWLEDGMENTS.....	IV
TABLE OF CONTENTS.....	VI
LIST OF TABLES.....	IX
LIST OF FIGURES.....	X
LIST OF ALGORITHMS.....	XII
LIST OF ABBREVIATIONS .....	XIII
LIST OF NOMENCLATURE.....	XVI
LIST OF GUIDELINE-RELATED PROJECTS .....	XIX
CHAPTER I. INTRODUCTION TO GUIDELINE-DRIVEN CLINICAL INFORMATION SYSTEMS .....	1
Challenges in health care.....	1
Potential solutions .....	1
Clinical Practice Guidelines .....	4
Computer Interpretable Guidelines: Explaining the need for computerized CPGs.....	6
Representing CIGs.....	13
CHAPTER II. REVIEW OF FRAMEWORKS FOR MODELING, VERIFYING AND EXECUTING GUIDELINES.....	18
Early CDSSs.....	18
CIG lifecycle.....	21
Components of a CIG-based CDSS .....	24
Evaluation criteria for guideline-based clinical information systems.....	26
Guideline modeling languages, formalisms and frameworks selected for evaluation.....	36
Guideline modeling languages not selected for evaluation .....	70
Other important medical formalisms, frameworks and organizations .....	70

CHAPTER III. EVALUATION OF OPEN QUESTIONS IN CIG LANGUAGE DESIGN .....	75
Earlier comparison efforts .....	75
Discussion.....	80
Open problems .....	85
CHAPTER IV. OVERVIEW OF THE SEPSIS PROJECT .....	113
Medical Context: Sepsis Management .....	114
Functional Architecture of STEEP.....	115
Modeling Language and Model Development.....	118
Software design and implementation .....	119
Evaluation .....	120
CHAPTER V. A MODEL-INTEGRATED IMPLEMENTATION ARCHITECTURE FOR STEEP .....	121
Implementation architecture overview .....	121
Treatment Management Console.....	122
System integration interfaces .....	125
Contributions .....	126
CHAPTER VI. MODELING LANGUAGE AND MODEL DEVELOPMENT .....	128
Modeling language development approach.....	129
Design of CPML .....	131
CPML vocabulary.....	136
CPML metamodels - Structural semantics.....	147
CHAPTER VII. DEFINING THE BEHAVIORAL SEMANTICS FOR CPML.....	153
Specification of behavioral semantics .....	154
Template-based specification of behavioral semantics.....	155
Analysis tool chain .....	161
CHAPTER VIII. VALIDATION AND VERIFICATION .....	168
Validation through simulation .....	168
Verification through model checking.....	170

CHAPTER IX. CONCLUSION.....	183
Summary .....	183
List of contributions .....	185
Lessons learned and future directions.....	185
APPENDIX A : LIST OF RELATED PUBLICATIONS .....	195
Peer-reviewed journal articles (first author) .....	195
Peer-reviewed journal articles (not first author).....	195
Book chapters .....	195
Conference papers.....	195
APPENDIX B : CURRENT STATUS OF THE INTEGRATED STEEP ARCHITECTURE AT VUMC .....	197
APPENDIX C : ALGORITHM FOR TEMPLATE-BASED MODEL TRANSFORMATION .....	198
BIBLIOGRAPHY .....	202



## LIST OF TABLES

Table 1	Suitability of various formal language groups for satisfying requirements for modeling CPGs.	16
Table 2	Questions related to Scope .....	32
Table 3	Questions related to Knowledge concepts.....	32
Table 4	Questions related to Formal semantics.....	33
Table 5	Questions related to Protocol composition .....	33
Table 6	Questions related to Security and privacy.....	34
Table 7	Comparison of features of six CIG formalisms [157] .....	76
Table 8	CIG formalisms compared in published studies .....	80
Table 9	CPML vocabulary .....	138
Table 10	CIG-based CDSS verification opportunities .....	171
Table 11	The most significant development task in the STEEP project .....	186

## LIST OF FIGURES

Figure 1	The specializations of CDSSs.....	13
Figure 2	Lifecycle of a CIG .....	21
Figure 3	Schematic of the components of a general CIG-based CDSS.....	26
Figure 4	The domino model, the basis of the PROforma language [26].....	43
Figure 5	The inheritance tree of PROforma component object types [36] .....	45
Figure 6	Icons of the generic and specific task types in PROforma [97,102] .....	46
Figure 7	Contents of an example for a PROforma guideline (plan) [36].....	46
Figure 8	PROforma task state transitions [42] .....	47
Figure 9	Arezzo's PROforma-based architecture [26] .....	48
Figure 10	PROforma patient-diagnosis scenario modeled in the Tallis composer [82].....	49
Figure 11	Overview of the high-level classes in GLIF3 [109].....	53
Figure 12	Patient-diagnosis scenario modeled in GLIF3.5 using Protégé [82].....	55
Figure 13	Execution states of a GLIF guideline step and possible transitions in-between [116].....	55
Figure 14	The internal structure of GLEE (of GLIF3), and its interactions with the environment [116] .	58
Figure 15	Screenshot of GLEE's standalone GUI during development and testing (client side) [116] ...	59
Figure 16	Overall structure of a plan library in Asbru [124].....	62
Figure 17	State machine of an Asbru plan [124].....	64
Figure 18	Patient-diagnosis scenario model in AsbruView [82].....	65
Figure 19	DeGeL architecture for Asbru guidelines [26].....	66
Figure 20	The Asbru engine [138] .....	67
Figure 21	Extended state machine of an Asbru plan in Spock [143].....	69
Figure 22	Deciding on protocol applicability (Directed versus empiric therapy).....	94
Figure 23	Standard versus fuzzy logic for representing uncertainty.....	96
Figure 24	Forms of non-determinism in CIGs.....	97
Figure 25	Simplified decision support flowchart.....	98
Figure 26	Grey tracking (illustrated for a TNM-based CIG).....	99
Figure 27	User aspects of a CIG.....	106
Figure 28	CIG portability.....	107
Figure 29	Functional Architecture of STEEP .....	115
Figure 30	CPMA: The implementation architecture for STEEP .....	122

Figure 31	STEEP GUI: Structure of the Treatment Management Console .....	123
Figure 32	STEEP GUI: The Treatment Management Console.....	125
Figure 33	The Multiple Urgent Sepsis Treatments (MUST) protocol [22].....	133
Figure 34	Initial version of the VUMC sepsis treatment protocol.....	135
Figure 35	Main concepts of the CPML’s Medical Library (represented with MetaGME) .....	140
Figure 36	Main concepts of the CPML’s Orderables (represented with MetaGME) .....	142
Figure 37	Example Protocol hierarchy .....	143
Figure 38	Sepsis treatment CIG component hierarchy in CMPL .....	144
Figure 39	Model-based configuration of the STEEP GUI.....	147
Figure 40	Protocol Modeling in CPML: Protocol, Process (represented with MetaGME) .....	148
Figure 41	Protocol Modeling in CPML: Event, Activation (represented with MetaGME).....	149
Figure 42	Protocol Modeling in CPML: Activity, Step, Selection Set (represented with MetaGME) ....	149
Figure 43	Exert from the sepsis CIG (in CPML).....	151
Figure 44	Generation of SF models from CPML .....	156
Figure 45	Behavioral template of an Activity (defined in MATLAB Stateflow) .....	157
Figure 46	CPML to Matlab Simulink/Stateflow transformation tool .....	162
Figure 47	Screenshot of the Matlab script generator tool.....	164
Figure 48	Generation of SF models from CPML (Sepsis).....	165
Figure 49	Generated sepsis CIG in Matlab .....	166
Figure 50	Generation of SF models from CPML (Simplified sepsis) .....	174
Figure 51	Generated functional models in MATLAB (Example 1 and Example 2) .....	176
Figure 52	Generated functional models in MATLAB (Example 3) .....	178
Figure 53	Generated functional models in MATLAB (Example 4) .....	179
Figure 54	Generated functional models in MATLAB (Example 5) .....	180
Figure 55	STEEP project timeline .....	197

## LIST OF ALGORITHMS

Algorithm 1	Simplified template-based model transformation process .....	160
Algorithm 2	Detailed template-based model transformation process.....	198


## LIST OF ABBREVIATIONS

ANSI	-	American National Standards Institute .....	37
ASTM	-	American Society for Testing and Materials .....	37
BNF	-	Backus Naur Form .....	44
CDSS	-	Clinical Decision Support System .....	2
CIG	-	Computer Interpretable Guideline .....	7
CIS	-	Clinical Information System .....	1
COGS	-	Conference on Guideline Standardization .....	5
CPG	-	Clinical Practice Guideline .....	4
CPM	-	Clinical Process Management .....	3
CPMA	-	Clinical Process Management Architecture .....	121
CPML	-	Clinical Protocol Modeling Language .....	115
CPN	-	Colored Petri Nets .....	79
CPOE	-	Computerized Physician Order Entry System .....	3
CSP	-	Communicating Sequential Process.....	136
CWF	-	Clinical Workflow .....	3
CWS	-	Clinical Workstation .....	122
DB	-	Database .....	25
DSML	-	Domain-Specific Modeling Language.....	13
DSS	-	Decision Support System .....	3
EBM	-	Evidence-Based Medicine .....	5
EBNF	-	Extended Backus-Naur Form.....	138
ECA	-	Event, Condition, Action .....	19
EE	-	Execution Engine.....	10
EHR	-	Electronic Health Record.....	2
eMAR	-	Electronic Nursing Medication Administration Records System .....	103
EMR	-	Electronic Medical Record .....	2
FOS	-	Free and Open Source.....	77
GELLO	-	Guideline Expression Language, Object-oriented.....	52
GEM	-	Guidelines Elements Model .....	70
GLARE	-	GuideLine Acquisition, Representation and Execution.....	80

GLEE	-	Guideline Execution Engine.....	52
GLIF	-	Guideline Interchange Format .....	51
GME	-	Generic Modeling Environment.....	132
GUI	-	Graphical User Interface .....	26
HCI	-	Human-Computer Interaction.....	119
HCO	-	Health care Organization .....	1
HeCaSe2	-	Health Care Services (release 2).....	80
HL7	-	Health Level Seven .....	37
ICU	-	Intensive Care Unit.....	114
IF	-	Interface .....	25
ISIS	-	Institute for Software Integrated Systems.....	113
LTL	-	Linear Temporal Logic.....	176
MDA	-	Model-Driven Architecture .....	121
MIC	-	Model-Integrated Computing .....	113
MLM	-	Medical Logic Module .....	37
MoC	-	Model of Computation.....	33
PIM	-	Platform Independent Models.....	121
PIS	-	Pharmacy Information System.....	103
PRODIGY	-	Prescribing RatiOnally with Decision-support In General-practice studY.....	70
PSM	-	Platform Specific Models .....	121
RIM	-	Reference Information Model .....	52
SAGE	-	Standards-based Sharable Active Guideline Environment .....	70
SIRS	-	Systemic Inflammatory Response Syndrome.....	110
SpEM	-	Specification Execution and Management Plan.....	70
SSC	-	Surviving Sepsis Campaign .....	114
STEEP	-	Sepsis Treatment Enhanced through Electronic Protocolization.....	113
TMS	-	Treatment Management System .....	116
TNM	-	Task Network Model .....	83
UI	-	User Interface.....	25
UML	-	Unified Modeling Language .....	13
UMLS	-	Unified Medical Language System .....	73
vMR	-	Virtual Medical Record.....	40

VOIS	-	Vanderbilt Oncology Information System.....	189
VUMC	-	Vanderbilt University Medical Center.....	113
WSBPEL	-	Web Services Business Process Modeling Language .....	13

## LIST OF NOMENCLATURE

The following list contains phrases, for which definitions are defined in this document. These  phrases are highlighted with a custom style (italicized and underlined). Their definitions can be found either before or after the phrase.

Action (in CIGs).....	9
Atomic Action (in CIGs) .....	9
Behavioral Semantics.....	33, 118
CIG-based CDSS.....	10
Clinical Decision Support System (CDSS) .....	2
Clinical Information System (CIS).....	1
Clinical Pathway .....	11
Clinical Practice Guideline (CPG).....	4
Clinical Process Management (CPM) .....	3
Clinical Workflow (CWF) .....	3
Cognitive Function .....	2
Communication Protocol .....	102
Composite Action (in CIGs) .....	9
Computer Interpretable Guideline (CIG) .....	7
Computerized Physician Order Entry System (CPOE) .....	3
Condition (in CIGs) .....	8
Consistency (of CIG-based treatment).....	99
Context (in CIGs) .....	10
Contextualized Guideline (see ‘Personalized Guideline’) .....	10
Contraindication .....	12
Data Point (in CIGs).....	8
Decision Support System (DSS).....	3
Delegated Atomic Action (in CIGs).....	9
Directed therapy .....	95
Domain-Specific Modeling Language (DSML).....	13
Empiric therapy.....	95



Engine (see 'EE') .....	10
Event (in CIGs).....	8
Event-Condition-Action (ECA) Rule.....	19
Evidence-Based Medicine (EBM) .....	5
Executable Atomic Action (in CIGs).....	9
Execution Engine (EE).....	10, 24
Execution Semantics (also see 'Behavioral Semantics') .....	33, 118
Formal Grammar.....	13
Formal Guideline.....	6
Formal Language.....	13
Formalism (see 'Formal Language') .....	11
Goal (in CIGs).....	8
Graphical User Interface (GUI).....	26
Grey Tracking .....	98
Guideline (see 'CPG') .....	4
Health indicator .....	94
Health indicator - Abstract.....	109
Health indicator - Composite .....	110
Health Level Seven (HL7) .....	74
Information Exchange Interface .....	102
Instance (of a CIG or Guideline) .....	11
Interference (of two or more CIGs) .....	90
Medical Logic Module (MLM) .....	37
Medical Terminology .....	28
Mid-flight Update.....	99
Model of Computation (MoC).....	33
Modeling .....	13
Model-Integrated.....	113
Objective data.....	94
Open World Assumption.....	7
Order Sentence .....	140
Order Set.....	9, 140

Orderable (see ‘Order Set’).....	9
Outcome Expectancy .....	6
Parameter (in CIGs).....	8
Personalized Guideline .....	10
Problem (in CIGs) .....	8
Protocol (see ‘CIG’) .....	7
Reference Information Model (RIM) .....	72
Rule-Based CIG Language .....	83
Rule-Based System.....	19
Scope.....	17
Self Efficacy .....	6
SF - Stateflow (in Matlab) .....	155
Side Effect Free .....	82
SLDV – Simulink Design Verifier (in Matlab) .....	155
Solution (in CIGs).....	8
Structural Semantics .....	132
Ternary Logic.....	60
TNM-Based CIG Language (see ‘Workflow-Based CIG Language’) .....	84
Unified Medical Language System (UMLS) .....	73
Validation of CIGs.....	22
Verification of CIGs .....	22, 101
Version fork.....	92
Versioning (of CIGs).....	92
Virtual Medical Record (vMR).....	72
Workflow-Based CIG Language.....	84

**LIST OF GUIDELINE-RELATED PROJECTS**

AAPHelp ..... 19

ABEL ..... 20

Arden Syntax ..... 36

Arezzo (see 'PROforma') ..... 42

Asbru ..... 61

AsbruView (see 'Asbru') ..... 64

Asgaard (see 'Asbru') ..... 61

CareVis (see 'Asbru') ..... 67

CASNET ..... 19

DeGeL (see 'Asbru') ..... 65

Delt/A (see 'Asbru') ..... 67

DXplain ..... 20

EMYCIN (see 'MYCIN') ..... 19

EON ..... 70

GASTON ..... 80

GELLO (see 'GLIF') ..... 52

GEM ..... 70

GEODE-CM ..... 80

GLARE ..... 80

GLEE (see 'GLIF') ..... 52

GLIF ..... 51

GUIDE ..... 70

HeCaSe2 ..... 80

HELEN ..... 80

INTERNIST I ..... 19

MYCIN ..... 19

NewGuide ..... 80

ONCOCIN ..... 20

PIP ..... 19

Prestige ..... 80

PRODIGY.....	70
PROforma.....	42
PUFF (see 'MYCIN') .....	20
QMR (see 'INTERNIST') .....	21
SAGE.....	70
SIEGFRIED.....	80
SpEM .....	70
Spock (see 'Asbru').....	68
Stepper.....	80
Tallis (see 'PROforma').....	42
TMYCIN (see 'MYCIN').....	20

## CHAPTER I.

### INTRODUCTION TO GUIDELINE-DRIVEN CLINICAL INFORMATION SYSTEMS

In order to understand why the use of guideline-driven clinical information systems is important in modern health care one needs to examine the current challenges that health care faces today.

#### Challenges in health care

According to a recent study of the Institute of Medicine, the primary challenges of health care currently are to make health care delivery safe, effective, patient-centered, timely, efficient, and equitable [1].

The study stresses that although addressing these challenges entails many different factors (e.g. emphasis on disease prevention rather than disease treatment), none is more important than the **effective use of information**. The study provides examples for effective use of information that includes: (1) cognitive support for health care professionals to help integrate evidence-based practice guidelines and research results into daily practice and to help integrate patient-specific data where possible, (2) instruments and tools that allow clinicians to manage a portfolio of patients and to highlight problems as they arise both for an individual patient and within populations, (3) rapid integration of new instrumentation, biological knowledge, treatment modalities, etc. into a “learning” health care system that encourages early adoption of promising methods

#### Potential solutions

##### **Clinical Information Systems**

To alleviate some of the above mentioned challenges, namely to reduce preventable errors in patient care and minimize administrative burdens, *health care organizations* (HCOs) are migrating from traditional, paper-based records to clinical information systems (CISs), a collection of computer-based applications that enables sophisticated services for patients and health care providers.

Various empirical evidence indicates that CISs can decrease health care costs [2–5], strengthen staff productivity [6–8] and promote patient safety [9,10]. Consequently, HCOs are adopting CISs to enable a

wide array of functions, including data sharing, decision support, employee training, student education, research, and access to reference materials.

### **Possibility for further improvements**

The increasing role of CIS is reflected in another recent study published by the Committee on Engaging the Computer Science Research Community in Health Care Informatics [11]. This study defines general principles for success for future health care systems, developed as a guide for CIS designers.

While elaborating on these principles, the committee agreed that a 21st century vision of health care would require intensive use of information technology to acquire, manage, analyze, and disseminate health care information and knowledge. The primary **design challenges were categorized** as: (1) data management, (2) integration and (3) medical knowledge management.

The design and use of CISs that base their operation on *electronic medical records* (EMRs)<sup>1</sup> adheres to the principles mentioned earlier and directly addresses the design challenges that fall into the first two categories. Accordingly, data management functions (1) take advantage of advanced information technology in creating and maintaining large-scale data repositories and in the automatic (e.g. sensor-based) capture of patient data. Solving the integration challenge for CISs (2) includes providing support for collaboration and for information sharing among software systems.

However, some of the most complex challenges are related to (3), the **management of medical knowledge**. Open problems include support for assisting the cognitive functions<sup>2</sup> of all caregivers (e.g. health professionals, patients, and their families) and support for capturing, managing and executing evidence-based guidelines. Due to their significance, we discuss these problems in more detail below.

The components of CISs that are designed to assist cognitive functions in care delivery are called *clinical decision support (advisory) systems*<sup>3</sup> (CDSSs). They form a significant part of the field of clinical

---

<sup>1</sup> The phrase EMR and EHR (electronic healthcare record) are often used synonymously. Typically, "EMR" is used in a clinical setting, often as a direct reference to an EMR-based CIS, whereas "EHR" is more broad expression and can refer to personal health records or home-based care.

<sup>2</sup> A *cognitive function* is a "mental process that involves symbolic operations (e.g. perception, memory, creation of imagery, and thinking). It encompasses awareness and capacity for judgment." [12]

<sup>3</sup> There are many definitions for CDSS in existence. An example can be found in [13,14], which says CDSSs "link health observations with health knowledge to influence health choices by clinicians for improved health care". In this document

knowledge management technologies through their capacity to use a combination of technology and expert knowledge to support the clinical process, from diagnosis and investigation through treatment and long-term care [16]. Among the full spectra of software components of a CIS there can be several that include clinical decision support functionality in one way or another. Here, we discuss knowledge-based versions of CDSSs<sup>4</sup>, which link health observations (patient data) with health knowledge to generate case specific advice through inferencing, and thus influence choices made by clinicians for improved health care [8] (see right side of Figure 1).

Below are examples for established knowledge-based CDSS system categories (a more detailed presentation of CDSSs including their history is presented in Chapter II).

1. Computerized physician order entry systems (CPOEs): These systems in general depend on comprehensive EMRs to provide means to physicians and nurses to create and execute orders for medications, tests, procedures and consults. They often include CDSS functionality to provide help with the completion and the validation of orders (e.g. recommending proper dosing and checking for drug-drug interactions). CPOE and related systems are often termed 'physician workflow' systems because they are designed to fit into and assist the normative group of activities that flow in between the specific surrounding systems and are considered the standard practice of medicine.
2. Clinical process management (CPM) systems: In these systems knowledge, tools and techniques are applied in a goal-oriented manner to define, visualize, measure, execute, control, report and improve processes of a clinical workflow. A clinical workflow (CWF) in this context can be thought of as a collection of organized tasks<sup>5</sup> designed to carry out a process in a clinical setting through facilitating some combination of software systems and groups of people. CWFs often

---

however, the term CDSS is defined to be more general: as a decision support system (DSS) in a clinical setting. The way DSS is interpreted in this document is described in [15]. DSS is "an umbrella term used to describe any computer application that enhances the user's ability to make decisions. More specifically, the term is usually used to describe a computer-based system designed to help decision-makers use data, knowledge and communications technology to identify problems and make decisions to solve those problems."

<sup>4</sup> Knowledge-driven decision support systems are systems designed to recommend actions to users. These AI-based systems are typically designed to sift through large volumes of data, identify hidden patterns in that data and present recommendations based on those patterns.

<sup>5</sup> In this thesis, we use the words *task*, *action* and *step* interchangeably to describe a task potentially deemed for completion by a person or a system when dealing with a given (sub-)problem in a medical guideline.

incorporate the use of various CIS components, including CDSSs to implement one or multiple steps.

The quality and usefulness of knowledge-based CDSS is determined by many factors, but most importantly by the quality of the knowledge that they rely on. Thus, it is easy to see that representation and management of that knowledge is an important factor.

Figuring out how to efficiently represent and manage medical expertise in general is a daunting challenge due to the fact that it involves so many aspects of medicine. A relatively well-defined subset of medical knowledge and one of the most desired functionalities of a CDSS is the support for guideline-based health care delivery. This direction offers documents describing guiding decisions and criteria regarding diagnosis, management, and treatment in specific areas of healthcare.

### **Clinical Practice Guidelines**

The central focus of our research is to aid the process of capturing, managing, verifying and executing evidence-based clinical guidelines. We believe that studying the general principles for future health care systems described in [11] can help us define what guideline-driven systems need to incorporate.

*Clinical practice guidelines* (CPGs) (or *guidelines* in short) are “systematically developed statements (recommendations, strategies) to assist practitioners in making decisions about appropriate health care in specific clinical circumstances” [17]. CPGs are often implemented around or as a part of CWFs. In [18], the **typical uses of guidelines** are listed as: screening and prevention, diagnosis and pre-diagnosis management of patients, indications for use of surgical procedures, appropriate use of specific technologies and tests as part of clinical care and care of specific clinical conditions.

In essence, CPGs provide guidance in (1) **identifying clinical situation** (e.g. diagnosis of a clinical condition) and/or (2) the **management of a clinical process associated with a predefined clinical situation** (e.g. executing the steps of a treatment).

Modern medical guidelines are based on the systematic examination of current evidence within the paradigm of evidence-based medicine to describe appropriate care based on the best available scientific evidence and broad consensus. Besides their primary goal, CPGs typically have **additional objectives**, such as to standardize medical care, raise quality of care, support workflows (including the management of responsibilities), reduce several kinds of risk (to the patient, to the health care provider, to medical



insurers and health plans), provide a focus for continuing education and achieve the best balance between cost and other medical parameters such as effectiveness, specificity, sensitivity, etc. [19,20].

Evidence-based (or best-practice) medicine (EBM) is described in [21] as conscientious, explicit, and judicious use of current best evidence in making decisions about the care of individual patients. It should not be interpreted as “cookbook medicine” as in practice, EBM usually means integration of individual clinical expertise with the best available external clinical evidence from systematic research.

Using CPGs to disseminate medical knowledge is a common practice; there are a lot of guidelines already published [22–24] and reportedly used [25].

Relying on CPGs provides many benefits. According to [26], the use of CPGs can improve the quality of clinical decisions and activities. This, in consequence, improves patient outcomes (e.g. a clinician will remember to check an important aspect before ordering a specific treatment) and reduces unnecessary and inappropriate variation in practice. CPGs promote interventions of proved benefits and discourage those that are ineffective. They also help physicians to use the clinical knowledge about the patient at the appropriate point of care. Furthermore, CPGs facilitate the reuse of knowledge, because guidelines can be adapted, tailored and applied to different clinical situations. Last, but not least, guidelines provide a relatively quick method for the dissemination of state of the art clinical knowledge, including updates and changes.

Reusability and portability of CPGs to disseminate and promote best clinical practice has major significance. Guideline authors are encouraged to employ rigorous formal techniques, which help to ensure syntactic, logical and medical validity of CPGs. Two notable efforts facilitating this guideline formalization are done by the National Guideline Clearinghouse [27] and the Conference on Guideline Standardization (COGS).

1. The National Guideline Clearinghouse provides means for standardizing CPGs by offering recommendations on what components guidelines should include. These recommendations define various guideline attributes that help specify a CPG, such as categories for classifying the major focus of the guideline (e.g. counseling, diagnosis, management), clinical specialty for classifying the field of medicine that might use the guideline professionally (e.g. anesthesiology, cardiology, hematology), intended users (e.g. people, such as dietitians, nurses and patients or entities, such as hospitals), objectives, and target population [28].

2. The other standardization effort, provided by the COGS, is known as the “COGS checklist”. This checklist provides a framework (i.e. a recommended schema) for supporting comprehensive documentation of practice guidelines. The COGS checklist contains 18 topics (e.g. goal, target population, potential benefits and harms, algorithm, etc.) and their definitions in [29].

### **Computer Interpretable Guidelines: Explaining the need for computerized CPGs**

The use of evidence-based CPGs for managing complex clinical problems is already standard practice. The integration of published CPGs into clinical workflows through the use of CISs has tremendous potential and would further extend the usefulness of CPGs. However, according to [30] there are numerous limiting factors to the adherence of CPGs. These include the lack of awareness of the existence of guidelines, the lack of agreement with using specific guidelines (or even guidelines in general), the lack of physician self-efficacy<sup>6</sup>, the lack of outcome expectancy<sup>7</sup>, and the inherent difficulty to change habits in daily behavior.

Still, one of the most difficult problems in the adoption of CPGs is that they are not directly applicable in CISs, as general guidelines are informal (regardless of the fact that guideline authors are encouraged to employ rigorous formal techniques to help ensure syntactic, logical and medical validity of CPGs [26]). This is because CPGs are phrased using natural languages. To be truly effective, **CPGs must be captured in a formal manner, built on information that already exists (or could exist) in CISs, and must be customizable to different clinical situations thus be transformed into individualized clinical care plans (guideline instances)**. The *formalness* of guidelines here means that the meaning of guidelines has to be unambiguous in the context in which they are planned to be used. If the goal is human interpretation then the guidelines have to use terms that are unambiguous to health care professionals. If the goal is guideline execution on a specific (software) platform then guidelines have to use concepts that are unambiguously understood by the platform. Finally, if the goal is theoretical execution and analysis then the guidelines require a mathematically precise definition.

---

<sup>6</sup> *Self-efficacy* is the belief that one can actually perform a behavior. It influences whether a behavior will be initiated and sustained despite poor outcomes.

<sup>7</sup> *Outcome expectancy* is the expectation that a given behavior will lead to a particular consequence.

The first step in this direction would be the development and acceptance of guideline representation languages with well-defined syntax and semantics. However, the implementation of guideline-based CDSS, including the representation of clinical knowledge still lacks proper standards. Certainly, there have been several formalization attempts – discussed in Chapter II – each tackling a certain aspect of clinical knowledge representation with various breadth and depth. Some of them have even been accepted as standards. Nevertheless, none of them can be seen as a complete solution, thus the building of these systems remains difficult.

Current implementations of knowledge-based systems typically become “one-offs”: to achieve a specific objective, system developers study a specific clinical process and environment, then develop specific code using high-level programming languages to address the problem in that local environment. Over long periods of time (years to decades), adding new functionality and evolving CIS (including CDSS) capabilities can become more difficult than initial development. Commercial vendors experience difficulty in supporting “one size fits all” systems for diverse customer bases. The growing complexity of biomedical research and clinical practices exacerbates the local “re-inventing the wheel” problem to a breaking point.

The integration of CPGs into existing or future CIS requires the guidelines to be interpretable and manipulatable by computers. We define computer interpretable guidelines (CIGs) as **formal, computer-readable versions of CPGs consisting of a coordinated composition of (medical) tasks or actions, with a mathematically precise notation and well-defined semantics**. These properties of CIGs allow them to serve as a substitute for direct code-based implementation of CPGs.

CIGs (in this document also referred to as *clinical protocols*, or *protocols*<sup>8</sup> in short) can be seen as more specific than CPGs, defined in greater detail. Regular CPGs can be incomplete, which means that directly implementing the logic described within can lead to **unwanted non-determinism** (i.e. ambiguity). In addition, – in the case of CPGs – the **open world assumption**<sup>9</sup> is practiced: information is provided only on the matter of interest and a lot of the described conditions, actions, etc. are context dependent.

---

<sup>8</sup> In this thesis, we use the terms *CIG*, *protocol* and *guideline* interchangeably. To resolve ambiguity for the word “guideline” we use the term CPG whenever it is necessary to make the distinction from CIGs.

<sup>9</sup> The open world assumption is the opposite of the closed world assumption, which holds that any statement that is not known to be true is false. This means that – in case of open world assumption – what is not explicitly stated cannot be known or assumed.

Protocols, on the contrary, employ the closed world assumption: they provide "a comprehensive set of rigorous criteria outlining the management steps for a single clinical condition or aspects of organization" [20]; they also must not contain unplanned non-determinism.

### Basic design of CIGs

The design of CIGs has attracted significant attention. Based on much shorter descriptions provided in [31] and in [32], we extended the list of components of guidelines to be the following.

1. The problem (e.g. an unwanted health condition) is a situation of interest, for which a solution (e.g. treatment) needs to be provided. The problem is defined by:
  - a set of conditions describing the problem (e.g. treatment is for female subjects who have a blood pressure reading outside of a specified target range for at least five minutes). Conditions are usually specified with the help of logic formulas that reference various relevant (clinical) parameters.
  - a set of parameters (i.e. attributes) used in conditions. They are situational beliefs made available directly through measurements and observations, or indirectly through calculations, including estimations. The type of these parameters determines whether the automatic monitoring of conditions is possible (e.g. a patient's heart rate can be automatically monitored with a sensor, but a condition such as "stomach ache" needs to be entered in to the EMR by someone manually). Parameters either can be linked to
    - events, which are detectable or derivable occurrences that are triggered by the changes in the state of patients (e.g. disease progression), or the changes in the state of treatments (e.g. performed provider action), or
    - data points, which are queryable, time stamped values for storing the result of measurements and observations.
2. The solution (e.g. treatment plan) for the problem specified. It is composed of a set of goals that allow a solution to be determined successful (or unsuccessful) and a coordinated set of actions that are designed to achieve these goals. Specifically:
  - a goal (i.e. intention), which according to [33] represents "temporal patterns of provider actions or patient states, to be achieved, maintained, or avoided". In further

elaboration, a goal describes the motivation by associating a problem and a solution and defines targeted patient and protocol states (e.g. blood pressure of the patient has to stay inside a specified range for two weeks).

- a set of actions, which range from diagnostics through message exchange to clinical interventions (e.g. administer a certain drug 3 times a day before meals). An action can be
  - an atomic (or simple) action: these actions are atomic from the point of view of the entity interpreting the CIG. This means that decomposition of the action is either not needed (i.e. execution is done by the interpreting entity), or performed by another entity (i.e. delegation).
    - delegated atomic action: these actions are assumed to be completed by other (servicing) entities. The example provided above is considered atomic if the CIG was designed for physicians and the action (“administer a certain drug 3 times a day before meals”) is performed by the patient. Another example for atomic, but complex actions would be an order set<sup>10</sup>, where the entity interpreting the CIG (e.g. physician) understands that another entity (most likely a CPOE system) will handle all necessary tasks<sup>11</sup> related to the order.
    - executable atomic action: these actions are assumed to be completed by the entity interpreting the CIG. An example for this would be an information-gathering step describing the need to measure the weight of the patient at a routine visit.
  - a composite action: these actions are a coordinated composition (e.g. serial execution) of (atomic or other composite) actions to provide a solution to an associated problem.

---

<sup>10</sup> An order set (or orderable) is “a functional grouping of orders in support of a protocol that is derived from evidence based best practice guidelines” [34]. Order sets have a specific purpose, they may contain conditional logic, they may be part of a larger care plan, and some items in an order set may be fully specified, others may have more optionality.

<sup>11</sup> Necessary tasks, in case of a medication bundle, include checking for contraindications (e.g. drug-drug interactions, allergies, etc.), transportation, and administration.

The *context* is also often specified, which includes all relevant clinical conditions. By extending the definitions for the *problem* and the *solution*, this component aids in identifying the situations when a CIG is allowed, recommended, required or contraindicated.

Composition and decomposition are important notions in the design of CIGs. On one hand, the use of appropriate logic allows the decomposition of problems into smaller sub-problems, to which simpler solution can be linked. On the other hand, with the help of composite actions, solutions can also be split into a set of sub-solutions. Both of these require the ability to express precise coordination of sub-components. Both the coordination and the decomposition are always performed by the entity interpreting the CIG.

### **CIG execution - Patient management based on CIGs**

Both CPGs and CIGs are designed to define what actions need to be taken in specific clinical situations. There is however, a great difference between the two when it comes to their application.

In CPG-based patient management, clinicians study the guidelines by understanding all suggested actions and their specified context (i.e. clinical conditions, indications, etc.). Then for any given patient in (guideline) execution time, they apply the knowledge by deciding which parts of the CPGs are relevant. Performed actions, the state of the patient, and the intentions of the medical staff are recorded, observed, and abstracted over time. This process of applying general medical knowledge to a care process and then to a specific patient's medical condition(s) can be considered as a mapping of medical knowledge to cognitive decision support. The result is a *personalized (or contextualized) guideline*. It involves interacting with models (i.e. abstractions) of the patient placing the raw data into context and combining them with medical knowledge in ways that make clinical sense for a given patient. The mapping is also influenced by many non-medical factors, such as resource constraints (e.g. cost-effectiveness analysis, value of information), patient values and preferences, cost and time.

Assuming that CIGs are captured properly, systems facilitating them can do some of this functionality automatically. In a *CIG-based CDSS*<sup>12</sup>, CIGs are interpreted by a special software package called the *execution engine* (EE, or *engine* in short). An ideal EE is developed to provide two main functionalities.

---

<sup>12</sup> A *CIG-based CDSS* is a software system or group of systems that rely on the internal representation of CIG for providing clinical decision support.

First, it needs to understand and configure its behavior according to the guideline specifications provided in the form of CIGs. This step involves setting up communication channels towards other CISs (such as EMR and CPOE systems) and interpreting defined actions that can theoretically include ones, such as displaying recommendations on a dedicated interface, automatically administering a certain dose of medication through an intravenous line, performing a complete sub-protocol, or requesting other systems to perform an action. Second, it needs to continuously maintain the state of the guideline by communicating with and building on information from users and other systems. This way raw patient data can be automatically received, filtered, monitored, put in context and (in case their initiating condition is satisfied) a series of predefined actions can be automatically performed by the engine. Personalized CIGs are also referred to as *guideline instances*. The resulting series of steps form a *clinical pathway*<sup>13</sup>. The quality of support is dependent on (1) the expressivity of the CIG formalism<sup>14</sup>, (2) the quality of the CIG models (including number of various aspects and the level of detail), (3) the quality of the input parameters (including sampling rate and precision of objective clinical data), and (4) the implementation of the execution system.

### Benefits of Computer Interpretable Guidelines

Computerized support for implementing and executing such guidelines as formalized patient management protocols provides several benefits over and above those offered by CPGs [20]:

- **CIGs are adaptable:** CIS that capture general evidence-based CPGs can be tailored to fit into daily practice.
- **CIGs can be specific:** based on the evidence-based recommendations they encode, they can automatically generate recommendations tailored for an individual patient/case. Recommendations may include:

---

<sup>13</sup> *Clinical pathway* is a trajectory of a CPG/CIG; a subset of tasks and split-paths in the CPG/CIG followed by the medical team for a given patient that results in one specific traversal of the guideline.

<sup>14</sup> In this thesis, we use the terms *CIG representation language* and *CIG formalism* interchangeably to refer to a method for describing CIGs.

- offering a set of potential problems (i.e. diagnostics),
  - offering a set of valid solutions (actions) in a given scenario,
  - helping with the selection from alternatives (while considering timing, *contraindications*<sup>15</sup>, cost, local preferences, etc.),
  - helping with setting up their parameters of actions (e.g. dosing),
  - helping with the timing of actions (including their relative order), and
  - helping with the delegation of actions.
- **CIGs are explicit and formal:** they help improve the clarity of a guideline, e.g. in decision criteria and clinical recommendations. CIGs also help offer better descriptions of patient states.
  - **CIGs support automation:** they are an attractive paradigm for clinical decision support tools, since much of the knowledge contained in guidelines has already been rendered explicit [36]. With the help of such systems, CIGs can automatically propose triggers for timely patient screening and patient-specific decision support, and associated alerts and reminders. This feature also enables automatic documentation of CIG execution (i.e. trace logs).
  - **CIGs support analysis:** they can reveal errors in the content of a guideline by validation, simulation or verification. A CIG-based system can facilitate the tracking of protocol execution, which would help not only with encouraging compliance, but also improve the protocols themselves by enabling the analysis of outcomes (e.g. effectiveness of one solution versus another, time to respond metrics, cost analysis, etc.).
  - **CIGs can be linked to other knowledge sources:** they offer a readily accessible reference, providing selective access to guideline-specific knowledge.
  - **CIGs enable knowledge transfer:** they are based on the best practice available at the time and models can be updated on a regular basis as new findings emerge in the medical literature.

Figure 1 illustrates how the two different guideline representations can be used to drive guideline-based CDSs, which are a small subset of all possible CIGs (see Euler diagram on the right side of the figure).

---

<sup>15</sup> “A *contraindication* is a specific situation in which a drug, procedure, or surgery should NOT be used, because it may be harmful to the patient” [35].



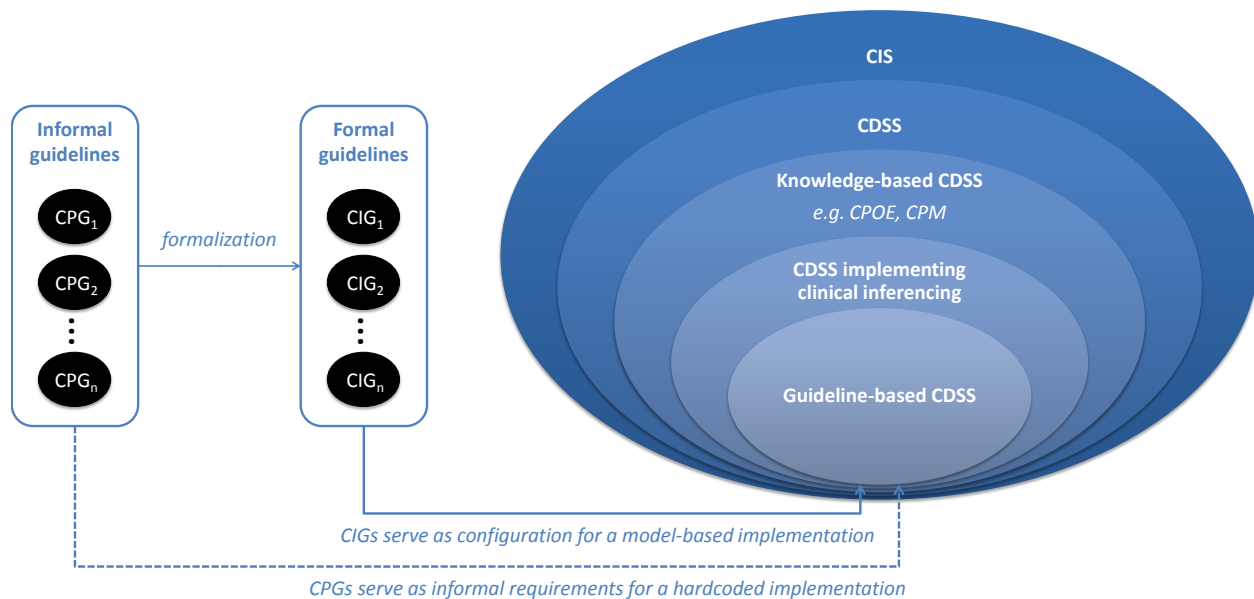


Figure 1 - The specializations of CDSSs

### Representing CIGs

The formal specification and representation of CIGs require the use of a *formal language*<sup>16</sup>. Using the abstractions provided by such a language enables the precise representation of the components (main concepts, flow of logic, etc.) of CIGs through a process called *modeling*.

There are many different languages that are formal<sup>17</sup>, and thus could be considered for modeling CIGs. This list includes standard general-purpose programming languages (such as C++ and Java), standard general-purpose modeling languages (such as UML (Unified Modeling Language) [39] and WSBPEL (Web Services Business Process Modeling Language) [40]) and specialized, purpose-built *domain-specific (modeling) languages* (DSML or DSL), which are programming or specification language designs with a

<sup>16</sup> A *formal language* is defined by a set of words (finite strings of letters, symbols, or tokens), where the letters are taken from an alphabet over which the language is defined. The set of valid words in a formal language is a subset of all possible combinations of letters from the alphabet, where validity of the words in the language is often defined by means of a formal grammar (also called its formation rules). Formal languages are entirely syntactic in nature but may be given semantics that give meaning to the elements of the language. [37] A *formal grammar* is a quad-tuple:  $G = (N, \Sigma, P, S)$ , where  $N$  is a finite set of non-terminals,  $\Sigma$  is a finite set of terminals and is disjoint from  $N$ ,  $P$  is a finite set of production rules of the form  $w \in (N \cup \Sigma)^* \rightarrow w \in (N \cup \Sigma)^*$ ,  $S \in N$  is the start symbol [38].

<sup>17</sup> They are formal in the sense that their constructs (or abstractions) have well-defined semantics.

particular problem domain, a particular problem representation technique, and/or a particular solution technique in mind [41]. Formal languages use different levels of abstractions and vary in their expressiveness and their usability for a given task. Consequently, selecting an appropriate formal language for modeling CPGs as CIGs is not a trivial task.

### **Modeling requirements**

There are certain generic modeling requirements that a potential formal language would need to satisfy in order to be considered “usable”. These are the following:

1. **Protocol models (i.e. formally modeled CIGs) need to capture medical knowledge explicitly and avoid ambiguity:**
  - a. Having an explicit and precise semantics allows for the understanding of the CIG without the system in which the CIG is used (e.g. EE).
  - b. The semantics of the language should be specified in such a way that any operation defined by a guideline has a precisely defined and unambiguous effect [42]. Unambiguity is also the requirement of a computer-based execution. CIGs of course often allow some degree of planned non-determinism, but these are resolved in runtime. (E.g. during the treatment of a patient, the used guideline provides alternative recommendations and the acting physician makes the selection on which alternative to go with).
2. **Protocol modeling languages need to represent concepts that are specific to the medical domain:** The authors of [33] describe essential characteristics of the medical domain in the following way:
  - a. Actions and effects are not necessarily instantaneous: actions are often continuous (i.e. have a duration) and their effects might be delayed,
  - b. goals often have temporal extensions,
  - c. there is uncertainty regarding the effect of available actions,
  - d. unobservable, underlying processes determine the observable state of the world,
  - e. goals may not be achievable, and

- f. parallel and periodic execution of plans, treatment processes is common.
  - g. In addition to these, previously recorded clinical data is often questioned and may be deemed to be untrusted or false.
3. **Protocol modeling languages need to support various levels of specialization:** Ideally, a protocol could be phrased in a general manner without being tied to specific medications, workflows or software systems at specific places. On the other hand, ideally, they would also support specialization by allowing the use of local abstractions (e.g. local interpretations and best practices). This would allow for a much faster adoption of CIGs developed by other groups and HCOs.
  4. **Protocol modeling languages need to support various modeling aspects and their composition:** There are many different aspects affecting guideline execution. These include laws and regulations, HCO-specific rules, physical constraints (such as availability of a certain medication at a given place and time), separation of roles performing actions, locations, etc. These aspects would ideally be separated and maintained by different personnel. The composition of these aspects would also need to be supported.
  5. **Protocol modeling languages need to support end-user programmability.** Medical professionals need to comprehend and field experts need to edit and update the models easily, eliminating the need for IT personnel to mediate between the medical and computer fields. This would allow domain experts to directly manipulate the expected behavior of the guideline-based system.

### **Selecting a suitable modeling language**

There are no widely accepted textual or visual languages for capturing CPGs. By examining the three classes of languages in the light of the previously defined requirements, we get Table 1.

Table 1 - Suitability of various formal language groups for satisfying requirements for modeling CPGs

	Requirement 1 Unambiguous Explicit Semantics	Requirement 2 Specialized Medical Concepts	Requirement 3 Levels of Specialization	Requirement 4 Composition of Modeling Aspects	Requirement 5 End-user Programmability
General-purpose programming languages	Possible	Not suitable	Not suitable	Not suitable	Not suitable
General-purpose modeling languages	Suitable	Not suitable	Suitable	Suitable	Not suitable
DSMLs	Suitable	Suitable	Suitable	Suitable	Suitable

As general-purpose programming languages provide an extremely low abstraction layer for modeling guidelines they are not ideal for representing CPGs. Similarly, general-purpose modeling languages are not suitable if requirement 2 and 4 are important factors. DSMLs on the other hand can support all listed requirements.

This raises an important question: **Is there (or is it possible) to define a single DSML to represent all guidelines or not?** The history of standardization indicates that people usually want one, but there are many factors making the problem of finding an ideal language difficult. In short, there are different formalization requirements, which are based on the differences in guidelines, users, locations, etc. In more detail:

- In practice, people use local abstractions and configurations.
- Guidelines usually provide guidance in identifying and executing the steps of a solution for which the guideline was designed for (both in diagnosis and in treatment), but they often put emphasis on different things. They can
  - help identify a given scenario (e.g. automatic diagnosis based on background data monitoring and evaluation),
  - offer a set of valid alternative solutions in a particular scenario,
  - help with the selection from a list of alternative solutions (while considering timing, contraindications, cost, local preferences, etc.),
  - help with setting up their parameters of selected solutions (e.g. dosing calculation),
  - help with the scheduling of actions, including absolute timing and relative order (i.e. scheduling triggers, alerts, and reminders),

- help with task assignment (i.e. who is supposed to do and what),
- provide means of documentation, and
- provide means of education.
- There are also many various types of clinical problems to be expressed. To determine the scope of a proposed solution, the following differentiating factors need to be specified:
  - Focus: to categorize the type of the problem being addressed (e.g. counseling, diagnosis, management),
  - Clinical specialty: to identify the related field/department of medicine (e.g. anesthesiology, cardiology),
  - Intended users (e.g. hospitals, dietitians, nurses, patients),
  - Specificity: to determine the details included in the solution (e.g. does the CIG define a list of specific medications or just a medication type),
  - Timespan: to specify how long can the guideline be relevant (e.g. problem spanning over a visit or a lifetime),
  - Time granularity: to define the required frequency with which data needs to be updated (e.g. millisecond-based or visit-based information sampling),
  - Patient population: to mark the targeted group of patients (e.g. everyone or between ages 12-14), and
  - Representation style: by gaging the state space, including the number of alternative routes and concurrent paths, appropriate representation can be chosen (e.g. all options represented with relatively rigid workflows, or many options limited by constraints).

To understand the solution existing formalisms need to be analyzed first. This issue is further discussed in the “Open problems” section.

## CHAPTER II.

### REVIEW OF FRAMEWORKS FOR MODELING, VERIFYING AND EXECUTING GUIDELINES

Today there are many different frameworks for modeling, verifying and executing medical guidelines in existence. Built by various groups, each of these CIG-based systems<sup>18</sup> differ in their scope and implementation, but share the fact that they were developed with the intention that by using them non-programmers will be able to create, maintain and facilitate computerized clinical guidelines [36].

In this chapter, first we present a quick overview of early CDSS efforts, mostly concentrating on ones implementing clinical inferencing<sup>19</sup> (see Figure 1). Non-inferencing CDSS, such as clinical dashboards, which show each patient's status in a chart are not evaluated. The historical overview is followed by an evaluation of a selected set of current CDSSs implementing CIGs, which includes an assessment of both guideline formalisms and their respective implementations. Finally, in the last section commonalities, differences and shortcomings are discussed.

#### Early CDSSs

The formalization of medical knowledge has been an active area of research since the 1960s. Research into the medicine-related use of artificial intelligence, knowledge representations and formal reasoning started in the early 1970's and produced a number of experimental systems [16]. Early efforts were focused on creating systems that mapped signs, symptoms and laboratory results to probabilistic estimates of different diagnoses [43]. These expert systems did not prove to be practical for the everyday practice of medicine. Only with the development and use of the EMR, have knowledge-based systems been adopted by practitioners [44].

[16] and [45] provide an informative summary of these early systems:

---

<sup>18</sup> In this chapter, we use the words "system", "framework" and "approach" interchangeably to refer to a CIG-based CDSS.

<sup>19</sup> Clinical inferencing here means logical inference over clinical knowledge.

- CASNET (Causal ASSociational NETworks) (1960) [46], developed at Rutgers University and implemented in FORTRAN, is a general tool for building expert system for the diagnosis and treatment of diseases. Its most significant application was the CASNET/Glaucoma system, designed for the diagnosis and treatment of glaucoma.
- PIP, the Present Illness Program (1970) [47], was built by MIT and Tufts-New England Medical Center. It gathered data and generated hypotheses about disease processes for patients with renal disease.
- AAPHelp (1972-2002) [48], published de Dombal at Leeds University, UK [49] is a system for supporting clinical assessment and decision-making in case of acute abdominal pain. Based on a naive Bayesian approach this early attempt implemented automated reasoning under uncertainty.
- INTERNIST I (1974-1985) [50–52], is a rule-based<sup>20</sup> CDSS designed to support diagnosis developed by Myers, Miller, Pople and Yu at the University of Pittsburgh as a successor of DIALOG and the predecessor of CADUCEUS and Quick Medical Reference (QMR) systems, which are discussed below. It uses patient observations to deduce a list of compatible disease states (based on a tree-structured database that links diseases with symptoms).
- MYCIN (1976) [55,56], developed by Shortliffe and colleagues at Stanford University, is a rule-based expert system designed to diagnose and recommend treatment for certain infectious diseases. It uses inferencing over reported symptoms and medical test results. Clinical knowledge is represented as a set of IF-THEN rules with heuristic certainty factors attached to diagnoses and reasoned over to come up with its recommendations. In the case of missing information, MYCIN would request further information concerning the patient, as well as suggest additional laboratory tests, to arrive at a probable diagnosis, after which it would recommend a course of treatment. Upon request, MYCIN would explain the reasoning that led to its diagnosis and recommendation. Successors of MYCIN include:
  - a) EMYCIN, or Essential MYCIN (1980) [56–58], is a system that evolved from MYCIN as a domain-independent framework for building and running new expert systems. The

---

<sup>20</sup> A *rule-based system* (also referred to as a *production (rule) system*) relies on storing and executing *event-condition-action (ECA) rules* (also known as *production rules*) [53]. These rules are basically IF-THEN rules, where the IF part is defined by a combination of triggering events and evaluated conditions and the THEN part is defined by the actions [54].

name was based on the fact that EMYCIN has MYCIN's framework without its medical knowledge base. EMYCIN is a backward-chaining rule interpreter that has much in common with Prolog. However, there are four important differences: (1) EMYCIN deals with uncertainty, so predications have a certainty factor assigned to them as opposed to true or false, (2) EMYCIN caches the results of its computation in order to avoid duplication, (3) EMYCIN provides an easy way for the system to ask the user for information, and (4) it provides explanations of its behavior.

- b) TMYCIN, or Tiny EMYCIN (1987) [59,60], patterned after the EMYCIN, is a simple expert system tool that only is intended to provide some of the most commonly used features of EMYCIN in a package that is small and simple. The internal implementation of TMYCIN has been written from scratch and is therefore different from that of EMYCIN.
  - c) PUFF [61] is a system designed to interpret pulmonary function tests for patients with lung disease.
- ABEL (Acid-Base and Electrolyte program) (1981) [62,63], developed at MIT as a successor of PIP, is an expert system employing causal reasoning, for the management of electrolyte and acid base derangements.
  - ONCOCIN (1981-1987) [64–66], developed at Stanford University, is a rule-based medical expert system for the management of oncology protocols. This system was the successor of MYCIN and predecessor of the Protégé and EON systems. ONCOCIN was novel in the sense that it attempted to model decisions and sequencing actions over time (e.g. history of past events and the duration of actions), using a customized flowchart language (OPAL).
  - DXplain (1984-today) [67], developed at the Laboratory of Computer Science at the Massachusetts General Hospital, is DSS that accepts a set of clinical findings (e.g. signs, symptoms, laboratory data) to produce a ranked list of diagnoses which might explain (or be associated with) the clinical manifestations. DXplain provides justification for why each of these diseases might be considered, suggests what further clinical information would be useful to collect for each disease, and lists what clinical manifestations, if any, would be unusual or atypical for each of the specific diseases. DXplain provides a description of over 2400 different diseases over 5000 different clinical conditions.



- QMR (Quick Medical Reference) (1980) [68,69], developed by the University of Pittsburgh and First Databank, California, is a diagnostic reference tool and consultation program, designed to provide general practitioners with easy access to the INTERNIST-1 knowledge base. QMR was designed with three types of uses in mind: (1) an electronic textbook, (2) an intermediate level spreadsheet for the combination and exploration of simple diagnostic concepts, (3) an expert consultant program that assists users with generating hypotheses for complex patient cases.

### **CIG lifecycle**

An aspect, which has to be understood for our evaluation, is the lifecycle of a CIG. This will help in understanding what the required components of a CIG-based CDSS are, which is described in the following section. We already discussed what a guideline-based execution means in the “CIG execution - Patient management based on CIGs” section, but the steps that get us there have not yet been mentioned. Based on [70] we define the lifecycle of a CIG to be composed of the following steps:

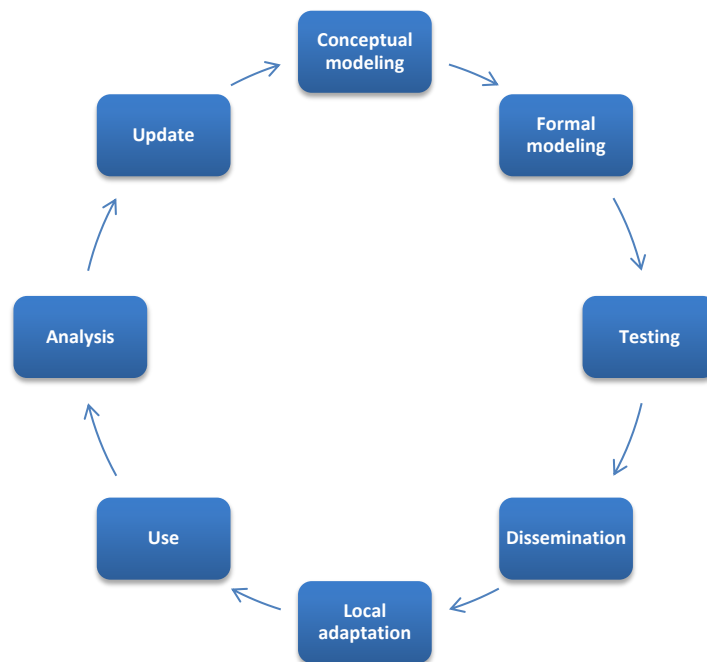


Figure 2 - Lifecycle of a CIG

The steps of Figure 2 are explained below<sup>21</sup>:

1. **Conceptual modeling:** The development of guidelines starts with the modeling of functional requirements of the intended application. This helps define the details and the characteristics that must be present in the resulting guidelines. The produced functional requirements can also serve as explicitly defined properties, which can be validated or verified in the testing phase.
2. **Formal modeling (i.e. encoding or authoring):** As part of the modeling process, guidelines (and their documentation) are created. This process can be aided by using authoring tools designed to capture the details needed by a particular model, while guaranteeing the unambiguity of the created models.
3. **Testing:** “This can be considered a step in authoring, aimed at determining that structured elements are precise, unambiguous, and syntactically and semantically correct (terms defined, with details for attributes such as units and allowed ranges), and that logical expressions and pathways are consistent and fully specified.” Testing may include:
  - a. *Validation*, which in the case of CIGs, means determining the degree to which the logic captured in a CIG implements the expected behavior. During the validation process, domain experts often analyze this through the simulation of test cases in an environment similar to (or the same as) the final execution environment.
  - b. *Verification* of CIGs, which means that there is support for phrasing requirements and specifications that include a range of safety, security and privacy related properties independently from CIGs that can then be used to check whether or not they hold for the CIGs in a particular environment.
4. **Dissemination:** The encoding scheme and the storage solution of guidelines must support retrieval and interpretation upon demand.
5. **Local adaptation (i.e. customization of adopted models):** There is a huge number of potential guidelines, and many possible implementation platforms. Thus, it is desirable to configure and customize guidelines not only for various applications (e.g. similar clinical conditions), but for the handling of various local constraints as well. Local constraints include:

---

<sup>21</sup> Excerpts are taken from [70].

- a. Adaptation of the medical content of guidelines, to **conform to situational constraints and distinctions** such as:
    - i. **lack of availability of certain resources** specified in a guideline (e.g. no MRI scanner)
    - ii. **local policies** (e.g. complying with local user access control rules for preserving privacy)
    - iii. **local preferences and workflows** (e.g. substitution of one medication for another in the same class)
    - iv. **contextual differences** (e.g. field, home, office, hospital).
  - b. **Integration with an implementation system (local dissemination)**: Furthermore, the application may need to be implemented in a variety of information system settings, with differences in:
    - i. **platform** (i.e. the host CIS, including potentially many other interacting CIS components)
    - ii. **user interface**
    - iii. **workflow** (i.e. defining the order in which different functions and systems are invoked)
    - iv. **encoding of data and knowledge.**
  - c. **Local testing** (e.g. integrity, integration, etc.)
6. **Use / application**: The use of CIGs means their execution. During this phase, CIGs help with data aggregation, diagnosis, order management, resource management, etc. depending on their intended purpose<sup>22</sup>.
7. **Analysis**: The effectiveness – including accuracy, cost, compliance and usability – of CIGs needs to be analyzed either during runtime or retrospectively, which means that context (including the measured values and selected decisions) needs to be documented, monitored for effectiveness, and used to improve the support models.

---

<sup>22</sup> More on the execution of CIG can be found in the “CIG execution - Patient management based on CIGs” section.

8. **Revision and update (i.e. evolution):** As guidelines change over time, “it is necessary to identify the impact on local adaptations and on implementations in which guidelines are embedded, so they can be updated appropriately.” Updates then need to be propagated to CIGs in place. In addition, mid-flight update mechanisms are needed to be worked out in order for updates not to interfere with critical CIG-based patient management already in place.

### Components of a CIG-based CDSS

In order to evaluate CIG-based approaches their architecture needs to be studied first. We found that such frameworks usually consist of (1) a **guideline formalism** for representing CIGs and (2) a supporting **software suite** for capturing, managing, executing and evaluating the CIGs.

1. **Guideline formalism:** All examined approaches employ a DSML. The purpose of these languages is to formally represent CIGs (and potentially other related information). Languages vary in terms of their scope, expressivity, the levels of abstractions provided, and their degree of formality.
2. **Software suite:** All examined approaches employ some sort of software suite, which consists of one or more software components. Typical components of software suites are the following:
  - a. **Modeling environment:** The purpose of the modeling environment is to enable the domain experts (i.e. knowledge engineers) to capture CIGs while enforcing the (syntactic and semantic) rules defined by the DSML. The modeling environment can be text or diagram based.
  - b. **EE:** As described in previous sections, the purpose of the execution engine is to provide an implementation of the semantics of the general guideline formalism, or in other words, take CIGs (defined with the help of DSMLs), interpret them and enact them for a specific patient. This means that an ideal EE will (1) configure its behavior according to the guideline specifications, (2) take input from a patient’s EMR and the health care providers, and (3) compute (i.e. infer) and enact the relevant actions from the CIG using the infrastructure provided by the CIS where it has been integrated. During operation the EE maintains a record of the dynamic state of the CIG process, including information on which tasks have been performed, which need (or need not) be performed, and the values of any data items associated with the process. With the help of a dedicated

interface, the engine implements a set of operations to allow other components to read or alter the state of the CIG in certain predefined ways [36]. In general, the execution of the process will require some of these actions to be performed by external human actors (e.g. clinicians) who will interact with the engine via some set of user interfaces.

- c. **Communication layer with software integration interfaces:** An extension to the EE. In order for the EE to communicate with its environment, it needs to have one or more *interfaces* (IFs) to it. These IFs, together with associated communication protocols, enable the exchange of necessary events and data (e.g. EMR and operational data) between the execution engine and other software systems of the CIS. In an ideal case, they build on both health care and IT standards.
- d. **Database:** An extension to the EE. A *database* (DB) can be in charge of storing multiple data sets that are vital to operation. These include:
  1. **The CIGs**, the rules, which will configure the EE. The CIGs themselves can contain the configurations for not only the guideline-based operation, but also the user interface and the software integration interfaces as well.
  2. **User access configuration**, for setting up authentication and to control who can have access to what, when and how. This configuration might be unnecessary if access control is achieved by other components of the host CIS.
  3. **An EMR cache**, which is a predetermined subset of the EMR for storing patient data relevant to the CIGs in use. The contents of this EMR snapshot are what the engine considers as facts in execution time.
  4. **The EE action log**, which is a log of all state changes of the EE. This includes the user actions, such as which data items were accessed and by who, and what instructions were given with the help of the system.
  5. **The current state of the EE**, for providing persistency in case of a system failure.
- e. **Patient management user interface:** Another extension to the EE (which programmatically can be a part of either one of the previous components or can be implemented by other components of the CIS). The purpose of the *user interface* (UI) is to show the status of the CIG enacted by the EE as well as to allow interacting health care providers to provide information, make decisions, etc. Sophisticated UIs usually

present a combination of textual and non-textual (graphical) information, these UIs are often referred to as *graphical user interfaces* (GUIs).

- f. **Testing environment:** This optional component can be either (1) a standalone component or (2) one integrated into other components such as the modeling environment or the EE. As its name suggests a testing environment allows either validation by simulation, verification, or both.
- g. **Analysis environment:** An analysis environment is an optional component that supports the study of the effectiveness of the CIG-based care as well as quality metrics with respect to cost of treatment, compliance with guidelines, etc. Thus, it provides a feedback loop for further guideline improvement. It is not necessarily built together with the rest of the software components as it generally involves a large amount of manual processing.

Figure 3 below represents a schematic view of the described general software suite.

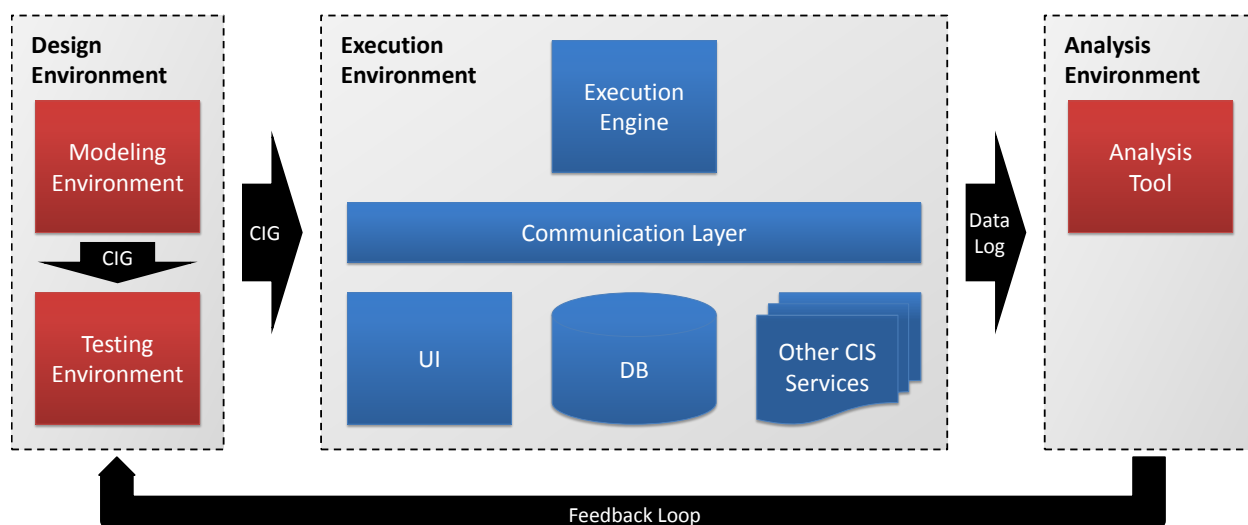


Figure 3 - Schematic of the components of a general CIG-based CDSS

### Evaluation criteria for guideline-based clinical information systems

Based on the review of the published literature on relevant work, as well as personal experience gained in the STEEP project [71], in this section we describe the list of requirements of an idealistic CIG-based CDSS. The reason behind creating such a requirement list is to be used as a checklist in the following

sections where we evaluate and compare various approaches. Evaluation points are categorized according to three major fields: (1) medicine, (2) computer science and (3) information technology.

### **Clinical aspect: Utility**

One of the most important components of an evaluation of a CIG-based CDSS is the clinical (or medical) utility of such a system. A proper evaluation of clinical utility of course would need to involve rigorous testing of the system in its prospective environment, which includes not only other software systems at the adopting HCO, but the feedback from potential users as well. Even though in this section we list many points regarding clinical utility, in our subsequent evaluation we can only consider evaluation points that can be assessed from a technical point of view and without the actual integration of the system into a specific CIS. We understand that a full evaluation (i.e. one, which includes stress testing, system integration, clinical compliance checking, and user evaluation) would result in a more complete evaluation, however such evaluation is out of the scope of this thesis.

### Logical adequacy of the CIG formalism

Evaluating the logical adequacy of a formalism for expressing CIGs is a difficult challenge. The requirement points defined in the “Modeling requirements” section, say that

1. Protocol models (modeled CIGs) need to capture medical knowledge explicitly and avoid ambiguity.
2. Protocol modeling languages need to represent concepts that are specific to the medical domain.
3. Protocol modeling languages need to support various levels of specialization (i.e. abstraction levels).
4. Protocol modeling languages need to support various modeling aspects and their composition.
5. Protocol modeling languages need to support end-user programmability.

Based on these requirements it can be concluded that a CIG representation language needs to provide high-level (preferably domain-specific) abstractions to hide complexity of the underlying infrastructure, while delivering a rich enough logic to express CIGs.

Logical adequacy of a formalism for a given purpose is greatly dependent on its original scope (the intended use of the language). In the previous section, “Selecting a suitable modeling language”, we discussed the factors that determine the scope: focus, clinical specialty, intended users, etc.

### Provided Functionality

Although criteria listed in this section are not all necessary conditions, having support for such functionality can help in the clinical adoption of a formalism:

- **Support for modeling and comprehension**
  - **Domain-specificity:** As opposed to generic approaches, a DSML-based formalism can enable medical professionals to comprehend and update the CIG easier. Thus the syntax should be as close to the domain as possible and the semantics should make it easy to reason about the behavior of a guideline [42].
  - **Relying on standardized medical terminology<sup>23</sup>:** As opposed to proprietary vocabularies, relying on one or more well-known standards<sup>24</sup> eases maintenance and supports portability.
  - **Use of unambiguous and intuitive syntax**
    - Textual notation: A textual notation allows computers to exchange information. The syntax should also conform to or use existing standards (e.g. XML) [42].
    - Graphical models: Although there is no consensus whether graphical models are preferred over textual ones, the existence of graphical representations and methods for graphical model manipulation can most likely ease clinical adoption. Having a purely graphical notation though might hinder understanding the represented logic, so the syntax should take into account

---

<sup>23</sup> Medical terminology is a vocabulary for accurately describing the human body and associated concepts, including anatomical terms, medical conditions, processes, procedures, medications, medical roles, medical fields, synonyms, and abbreviations [72].

<sup>24</sup> Some of the well-known medical terminology standards are described in the “Other important medical formalisms, frameworks” section.



that guidelines may be viewed graphically in ways that hide certain details of the guideline text [42].

- **Existence of a documentation tool:** The CIGs themselves should be – as much as possible – self-explanatory, which can be achieved using a combination of a formal language, comments and references to external knowledge sources. Additionally, there could be a need for a documentation tool that is capable of explaining to its users the reasons why it has recommended a particular course of action, or drawn a particular inference [36].
- **Existence of a modeling tool:** The purpose of a modeling tool is multifold: it allows users to capture CIG and other associated models, while potentially aiding model development by providing benefits such as enforcing correct-by-construction rules (e.g. syntax checking and enforcing structural semantics) and acting as a representation tool (i.e. self-documentation).
- **Support for adaptation**
  - **Control over visualization of CIG-related information:** This determines whether knowledge experts have control over how information related to CIG is rendered in the UI (e.g. there are control mechanisms in the formalism for deciding what elements should appear or whether a vital sign should be charted as values in a table or as a graph).
  - **Support for model adoption (i.e. portability):** Model adoption means that one HCO can either directly facilitate (i.e. incorporate) CIGs defined by another HCO, or it can easily adapt them to local needs. Support for model adoption from the clinical standpoint means that points 3) and 4) of the “Modeling requirements” section are supported.
  - **Reusability of built models:** Support for model reuse when constructing new CIGs is an advocated practice. This is especially true for systems where there is a high number of CIGs captured. This includes versioning.
- **Management of continual change:** Support for managing updates to CIGs including the ones in use.

- **Support for testing:**
  - Testing guidelines (prior release) from the clinical point of view means that there is a testing environment that allows for ensuring that
    - original requirements are satisfied, while
    - patient safety and
    - health information privacy are preserved.
  - Testing requires an environment similar (or identical) to the clinical environment where guidelines are intended to be used.
  - Testing needs to effectively evaluate whether an encoded CIG faithfully reflects the encoder’s intentions. For this evaluation inputs, outputs, decision points, etc. all need to be recorded.
- **Support for analysis:** Evaluation of the existence of an (retrospective) analysis environment, which – similarly to the test environment – at the minimum needs to be able to record the execution trace of each enacted guideline.

#### Technology readiness level

Technology readiness level (TRL) is a measure used by some U.S. government agencies and companies (including the Department of Defense and the National Aeronautics and Space Administration) to “assess the maturity of evolving technologies (materials, components, devices, etc.) prior to incorporating that technology into a system or subsystem. Generally speaking, when a new technology is first invented or conceptualized, it is not suitable for immediate application. Instead, new technologies are usually subjected to experimentation, refinement, and increasingly realistic testing. Once the technology is sufficiently proven, it can be incorporated into a system/subsystem” [73]. Here TRL describes parameters to indicate the readiness of a framework for clinical use:

- Number of protocols modeled
- TRL defined by [73] and adopted for evaluating clinical systems:
  1. Basic principles observed and reported
  2. Technology concept and/or application formulated

3. Analytical and experimental critical function and/or characteristic proof-of-concept
4. Component and/or breadboard validation in laboratory environment
5. Component and/or breadboard validation in relevant environment
6. System/subsystem model or prototype demonstration in a relevant environment: the framework is operational in a test environment
7. System prototype demonstration in an operational environment: the framework is successfully integrated into and is used in a particular CIS. This includes its integration not only into an existing CIS, but into clinical workflows as well.
8. Actual system completed and "flight qualified" through test and demonstration: the framework is ready to be used in other systems
9. Actual system "flight proven" through successful mission operations: the framework successfully integrated into and used in multiple CISs

#### **Computer science aspect: Knowledge representation and maintenance**

While the previous section explained requirements from the clinical side, this section describes the relevant properties and requirements from the computer science side. The computer science aspect deals with the knowledge representation and maintenance, key components for enabling clinical functionality of the formalism. Since knowledge is managed by the guideline formalism, we need to explore its relevant properties.

#### Scope

As stated earlier, we only evaluate domain-specific solutions; however, because solutions vary greatly in terms of what sub-domain they consider within the domain of medicine, the scope (or domain) of the formalism needs to be precisely articulated (see Table 2).

**Table 2 - Questions related to Scope**

Property	Description
Domain-specificity	What is the definition of the domain? (This question is only relevant if the formalism is domain-specific.) The definition of the domain includes stating scope items such as focus (diagnosis, treatment, task assignment, etc.), clinical specialty, intended users, etc.

### Knowledge concepts

Knowledge concepts describe the functional elements of the language. Relevant properties can be seen in Table 3.

**Table 3 - Questions related to Knowledge concepts**

Property	Description
Support for complex structures	Complex structures include using a hierarchy for the decomposition of problems and solutions (tasks) and reuse of existing components with some form of referencing.
Support for capturing parameters of tasks	Parameters of tasks include intentions (goals), relevance, success condition, failure condition, etc.
Support for ranking of alternative solutions	Offering competing alternative treatment options for a patient with a particular configuration of clinical indicators is typical, however, it is up to the formalism whether treatment absolute priorities, or with a help of scoring values.
Support for expressing temporality	Temporality is inherent to CIGs, but formalisms may vary greatly in terms of what time related properties can be expressed by them. A more detailed description can be found in "P1.5 Temporal reasoning" under the "Open problems" section.
Support for constructing derived data points	Derived data points are data elements that are not designed to be received by the CIG system (but could potentially be reused if made available to other systems). Construction of derived data points are usually done using input data values and logical operators of an <i>expression language</i> .

### Formal semantics

Defining the semantics of a DSML provides meaning for models of the language. In the case of DSMLs that allow the modeling of CIGs, semantics attach meaning to the captured CIGs. The benefit of formal specification of semantics<sup>25</sup> is that it removes unintended ambiguity, thus it helps ensure consistent and automated analysis of designs, reuse of models between tools, and increases the extent to which models can be constructed correctly during design. Properties relevant to providing formal semantics are described by Table 4.

---

<sup>25</sup> More on how formal semantics for DSMLs are defined can be found in [74].

**Table 4 - Questions related to Formal semantics**

Property	Description
Support for execution	Support for execution is dependent on whether the <i>behavioral (or execution) semantics</i> <sup>26</sup> of the elements of the formalism is defined or not. It is important that this semantics describe a guideline-independent execution model. In addition, where possible, the semantics should allow operations to be performed in parallel without ambiguity [42].
Support for verification of correct behavior	Verifying whether a CIG is correct (i.e. conforms to the specification) is only possible if the execution semantics are clearly defined and there are concepts for formally describing the specification (e.g. in the form of constraints) against which CIG models can be tested.
Interoperability with other formalisms	If the semantics of the formalisms $F_A$ and $F_B$ are explicitly defined with the help of a formal <i>model of computation</i> (MoC) <sup>27</sup> , the evaluation of whether a translation of the CIGs expressed with formalism $F_A$ to formalism $F_B$ becomes possible.
Error handling	Managing the execution of a CIG in a real-life environment means that there is logic in place for dealing with data that has been deemed false only after it has been processed by the system (e.g. supporting the roll-back feature).

### Protocol composition

Properties relevant to enabling the composition of protocols are described by Table 5.

**Table 5 - Questions related to Protocol composition**

Property	Description
Support for reuse and customization of existing models	Reuse of models often requires the customization of the components of the model to be reused. Customization can range from changing parameters to completely replacing components.
Support for handling protocol-protocol interaction	Support for handling protocol-protocol interaction allows the analysis of the effect and the resolution of potential conflicts of multiple guideline instances that are being executed on one patient (e.g. finding contradicting suggestions).

### Security and privacy

Properties relevant to providing security and privacy are described by Table 6.

---

<sup>26</sup> *Behavioral semantics* in general defines the dynamic evolution of a system's state along some model of time. For a modeling language, this means that it describes how the state of a model evolves over time [75]. *Execution semantics* is closely related to behavioral semantics, and can be used interchangeably unless the difference is stated otherwise.

<sup>27</sup> A *model of computation* (MoC) defines the principles of the behavior and the interaction of components. Examples of MoC include: Finite state machines (FSM), Statecharts – concurrent hierarchical FSM, Timed Automata (TA), Kahn Process Networks (KPN), Dataflow Process Networks (DPN), Petri Nets (PN) [76].

Table 6 - Questions related to Security and privacy

Property	Description
Support for security and privacy criteria modeling	Support for security and privacy criteria modeling includes the definition as constraints or policies over the existing CIG models.
Support for validation and verification of the CIG models	Support for validation and verification of the CIG models against a range of safety, privacy and security related criteria.

### Information technology aspect: System integration

The information technology aspect of the evaluation means listing the requirements regarding the integration of proposed architectures into existing CISs.

#### Information exchange

Information exchange is vital for any framework that intends to provide patient-based instantiation of CIGs. Concerns include:

- **Data consumption:** Data points, which an executing CIG instance bases its decisions upon, must be provided by the environment (either CIS system components or health care providers). Unless the CIG-based system is expected to accept data entered manually, this translates to access to data sources, such as the EMRs. This means that some form of interoperability with the existing CIS components is required. For interoperability (e.g. automatic consumption of data coming from a host system), many concerns need to be addressed. These include specifying the mechanism of access (e.g. used technology and location), addressing security (e.g. access control), managing availability (e.g. what happens if the source system is not functional, or how often does data need to be sent), and specifying quality (i.e. minimum requirements for the data to be accepted, including “shelf life”, acceptable range, etc.).
- **Data maintenance:** Patient data, health care provider actions and provider-patient interactions unique to the system need to be automatically captured and stored (logged) according to the policies of the HCO, state, country, etc.
- **Data provisioning:** As an active component, a CIG-based system should be able to act as a data source to other systems as well.

- **Interface towards the host CIS:** In an ideal case, information exchange makes use of medical data standards<sup>28</sup> for easier interfacing to a host CIS.
- **Interoperability with other systems that use another formalisms:** Although a formalism with fully specified behavioral semantics allows for the understanding of its relation to another, the development of translation methods to and from other known formalism directly supports the reuse of CIGs in systems base their operation on another kind of CIG formalism.

#### System validation

A given formalism must have a well-defined, publicly available syntax and semantics so that it is possible to determine whether any given implementation is correctly reading and processing guidelines of the formalism. If either one is missing, or not available, outsiders will not be able to understand and analyze CIGs built using the formalism.

#### System scalability

System scalability is the ability of the system “to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth” [77]. Inspection of an approach from the system scalability standpoint is critical. This means that issues, such as processing speed, request distribution, and local customization need to be worked out.

#### System security

The objective of system security is to maintain quality, such as confidentiality, integrity and availability. It includes protection of information from theft, corruption, etc., while allowing the information and property to remain accessible and productive to its intended users.

---

<sup>28</sup> Some of the well-known medical data standards are described in the “Other important medical formalisms, frameworks” section.

## Guideline modeling languages, formalisms and frameworks selected for evaluation

Guideline modeling approaches are often categorized into two distinct groups [78]:

- The first group contains *model-centric approaches*, according to which domain experts build up conceptual CPG models using concepts from the language. In this case, the relationship between the model and the original (paper) document is only indirect.
- The second group contains *document-centric approaches*, which facilitate markup-based tools for editing and modeling computerized CPGs. According to these approaches, experts take CPGs (i.e. original, text-based guideline documents) and systematically mark them up in order to create a gradually more formal model of the selected source.

Here, distinction between model-centric and document-centric approaches is not made as with both approaches the result will need to be a CIG. Besides, the selection of the translation method is considered a subjective preference.

The evaluation of all published CIG-based systems is out of the scope of this thesis. In this section, only a selected set of approaches are presented. To determine which frameworks to evaluate out of the vast set of available systems, we studied the published literature, mostly concentrating on already existing evaluations, which helped us **identify the four most popular and most comprehensive approaches**. The selected approaches, namely Arden Syntax, PROforma, GLIF and Asbru, are described in the following subsections.

### **Arden Syntax**

One of the longest established medical knowledge representations is the *Arden Syntax* [79–81]. It sets out to provide a standard for capturing ECA rules and has been widely applied by industry [42,82]. Arden Syntax arose from the need to make medical knowledge and logic explicit and to standardize the way knowledge is integrated into proprietary CISs. This would allow sharing of the captured knowledge within and between institutions and make it available for decision making at the point-of-care [34].

### Development and maintenance

It was first introduced in 1989 at the Arden Homestead Conference in Harriman, New York. Arden Syntax was the result of a project run by Columbia Presbyterian Medical Center in New York City and



IBM Health Industry Marketing in Atlanta, Georgia. In 1992, Arden Syntax for Medical Logic Systems Version 1.0 was adopted by the American Society for Testing and Materials (ASTM) [83]. Version 2.0 was adopted by the Health Level Seven (HL7)<sup>29</sup> [84] and the American National Standards Institute (ANSI) [85] in 1999 and it has been sponsored by HL7 since.

Arden Syntax was formerly a standard of ASTM, but currently it is a standard of HL7. Its development and maintenance is overseen by the HL7 Arden Syntax Special Interest Group and the Clinical Decision Support Technical Committee [86]. At the present time, the official version is 2.8 [34,81,87].

### Use

Initially, the Arden Syntax was based largely on the encoding scheme for generalized decision support used in the HELP<sup>30</sup> system for providing alerts and reminders, developed at the LDS hospital in Salt Lake City [86]. Now it is widely used in the medical industry, an example is the Regenstrief Institute, Inc., where it is used in the CARE system to deliver reminders or hints to clinicians regarding patient treatment recommendations [88]. Other examples can be found in [89].

### Syntax and Semantics

The Arden Syntax is a rule-based formalism that is used to create self-contained (i.e. independent) units, called *Medical Logic Modules* (MLM), each of which encapsulates the logic necessary for an individual medical decision.

A MLM contains information representing the context in which an individual rule may become relevant, the logical conditions necessary for it to be activated, and the action (recommendation) that is performed when it is activated [36]. An individual MLM should contain sufficient logic to make a single medical decision [86].

An MLM is an ECA rule expressed using a custom procedural formalism. Each MLM can be thought of as a single-step "condition-action" rule. However, they can be hierarchically nested, which allows their content to describe a sequence of instructions, (including queries, calculations and logic and write

---

<sup>29</sup> HL7 is described in more detail in the "Health Level Seven" section later in this thesis.

<sup>30</sup> Health Evaluation through Logical Processing (HELP)

statements). Sequencing tasks can be modeled by chaining a sequence of MLM invocations (either synchronous or asynchronous) inside of another MLM container. MLMs can be used to generate clinical alerts and reminders, interpretations, diagnoses, screening for clinical research studies, quality assurance functions, and administrative support.

Arden Syntax has a procedural syntax. The syntax of a MLM is given as a stream of text stored in an ASCII file in statements called *slots* [80]. Recently XML versions were also proposed, which eliminates the need for custom built compilers [90,91]. Each slot consists of a slot name, followed immediately by a colon (e.g. title:), then followed by the slot body, and terminated with two adjacent (double) semicolons (;). The content of the slot body depends upon the actual slot, but it must not contain double semicolons, except inside comments, string constants, and mapping clauses. Implemented by slots, each MLM is composed of three main categories: maintenance, library, and knowledge (in this specific order). In the MLM, the categories and (sub)slots must follow a particular order, however, some slots are considered optional while others are required [89].

### Example

An example MLM written in Arden Syntax from [92] is provided below:

```
maintenance:
  title: CT study with contrast in patient with renal failure;;
  filename: astm_ct_contrast;;
  version: 1.00;;
  institution: ASTM E31.15; SMS;;
  author: Harm Scherpbier, M.D.;;
  specialist: ;;
  date: 1995-09-11;;
  validation: testing;;

library:
  purpose: Issue alert when physician orders CT study with contrast in
  patient with renal failure;;
  explanation: If physician orders CT scan with contrast, this rule retrieves
  most recent serum creatinine. If the value is less than 1 week old, and
  more than 1.5, the system issues an alert to the physician to consider the
  possibility that his patient has renal failure, and to use other contrast
  dyes.;;
  keywords: ;;
  citations: ;;
  links: ;;

knowledge:
  type: data_driven;;
  data: last_creat := read last {"Creatinine level"};
  last_BUN := read last {"BUN level"};
  ;;
```

```

evoke: ct_contrast_order;;
logic:
  if last_creat is null and last_BUN is null
  then alert_text := "No recent serum creatinine available. Consider
  patient's kidney function before ordering contrast studies."; conclude
  true;
  elseif last_creat > 1.5 or last_BUN > 30
  then alert_text := "Consider impaired kidney function when ordering
  contrast studies for this patient."; conclude true;
  else conclude false;
  endif;
  ;;
  action: write alert_text || "\nLast creatinine: " || last_creat || "
  on: " || time of last_creat || "\nLast BUN: " || last_BUN || " on: "
  || time of last_BUN;
  ;;
  urgency: 50;;
end:

```

### Data types

Arden syntax provides a few basic data types essential to medicine (Boolean, number, string, time, and duration), together with well-defined operations on them and the ability to structure them into lists. Dynamic data typing (including automatic type conversion) is also provided, and as a result, types are assigned to variables in MLMs at runtime [80].

### Expression language

Arden Syntax also defines an MLM query language, which is an expression language. This query language allows for the specification of the requirements of MLMs (i.e. input variables for the logic). The query language offers:

1. a method for specifying input parameters based on what is available in CISs (e.g. vocabulary, database schema, queries)
2. the specification of time constraints (temporal search window for data samples)
3. a set of operators for producing derived data points with filtering and aggregation (e.g. sum of measurements, average, maximum, rate of change, newest)
4. and a method for identifying patients.

### External references

Arden Syntax bypasses the problem of different institutions having different CISs with different methods for storing and accessing local data by simply using curly brackets (“{ ... }”) to allow the referencing of

local vocabularies and to allow the definition of local data retrieval methods [80]. It also assumes an underlying data model specified as a *Virtual Medical Record* (vMR)<sup>31</sup> [34].

### Implementation

There is published information on implementation of both Arden Syntax modeling and execution environments. An Arden Syntax prototype implementation was developed in Prolog and an EE implementation using C++ [93] and Java [94]. In addition, in [95] a tool called the MLM Builder was introduced. It is a self-contained, unified development environment for the creation, testing, and maintenance of Arden Syntax MLMs. According to the authors, it also generates C and Delphi code. The implementation of MLMs is usually event-driven. “With an appropriate computer program (known as an event monitor), MLMs can be invoked and run automatically, generating advice where and when it is needed, e.g. to warn when a patient develops new or worsening kidney failure” [86].

### Advantages and limitations

Some of the advantages of Arden Syntax are listed in [86]:

- It is domain specific. MLMs are intended to be written and used by clinicians with little or no programming training.
- It is formal in the sense that it has well-defined syntax and with the help of its prototype implementations execution semantics are also provided.
- It provides mechanisms for defining explicit links to local data, triggering events and messages to users. It also defines how MLMs can be invoked.
- It provides support for time functions and it ensures that every data element and every event has a date/time stamp. It facilitates a three-valued logic (true, false, unknown) to support limited uncertainty as well [32].

Some of the limitations of Arden Syntax are:

---

<sup>31</sup> The Virtual Medical Record is described in the “Other important medical formalisms, frameworks” section.

- The expressiveness of Arden Syntax is limited:
  - Arden Syntax was not designed for encoding complex multistep guidelines that unfold over time. [82]
  - It “does not represent anchored intervals of time directly, nor does it explicitly handle fuzzy times” [80]. However, this issue is being addressed in version 2.8 [34].
  - A potential problem with using Arden Syntax is that it has a limited set of predefined actions. For example, it does not explicitly define notification mechanisms for alerts and reminders. Instead, this is left to local implementation and – like database queries – is contained in curly braces in a MLM. [86]
- The maintenance of the MLM specifications is difficult:
  - It does not offer mechanisms for complexity management and for managing linked MLMs [82].
  - Since MLM specifications are stored as individual text files, Arden Syntax yields CIGs that can be neither easily queried, nor easily manipulated. Thus, Arden Syntax lacks support for higher-level abstract constructs, modularization for its rules, and the support for the manipulation and querying knowledge specifications. [32]
  - Arden Syntax has been defined with the intention to make MLMs swappable between disparate platforms, but much of this sort of logic has been written ad hoc into various EMR systems and is neither transferable, nor – in the case of closed source software – is it readily peer reviewed [96]. The major problem using the formalism to share clinical knowledge is the lack of common format for data encoding and manipulation [34]. This is the root cause of the problem known as the “curly braces problem”: Arden Syntax explicitly isolates references to the local data environment in curly braces (e.g. in order to provide alerts and reminders through interacting with local CIS components, such as a clinical database). Database schema, clinical vocabulary and data access methods vary widely so any encoding of clinical knowledge must be adapted to the local institution to use the local clinical repository. This hinders knowledge sharing. Efforts are underway in HL7 to help solve this problem, but it is not something that the Arden workgroup can do alone; it requires industry-wide standardization. [86]

Although the Arden Syntax has been important and influential, it is recognized that in order to formalize complex decisions and clinical workflows, and develop complete electronic guideline applications, a more expressive formalism will be needed [42].

## **PROforma**

PROforma [97–99] is a clinical guideline representation and interchange format. It is a formal process modeling language allowing clinical guidelines to be expressed in a computer-interpretable manner. PROforma models are executable and have been used successfully to build and deploy a range of decision support systems, guidelines, and other clinical applications. The technology includes the PROforma language and a set of Prolog and Java tools for building applications using the language [98].

### Development and maintenance

PROforma was developed at the Advanced Computation Laboratory of Cancer Research, UK for the general purpose of building decision support and intelligent agents [98]. Work leading to the design and implementation of PROforma was carried out in a series of projects largely funded by European agencies, starting in the late 1980s [100,101]. PROforma itself was a major result of the EC 4th Framework PROMPT project, which started in 1992 and completed in 1998 [98]. The work was awarded the 20th Anniversary Gold Medal of the European Federation of Medical Informatics in Copenhagen in 1996 [98]. The two implementations Arezzo and Tallis were introduced in 1996 and 2000 respectively [97].

### Use

PROforma is a continuing area of research at the Advanced Computation Laboratory of Cancer Research, particularly for safety-critical applications. PROforma is the platform for a number of clinical applications developed by the lab. Some examples include REACT, RAGs, ERA, but a much more comprehensive list can be found in [97]. Web-based PROforma applications are currently under development [98].

### Syntax and Semantics

The PROforma language forms the basis of a method and a technology for developing and publishing executable clinical guidelines [97]. It combines logic programming and object-oriented modeling and is formally grounded in the R2L Language [101]. PROforma is essentially a first-order logic formalism

extended to support decision making and plan execution, but it also incorporates a number of well-known features of non-classical logics (e.g. modal logic, temporal logic) and two novel logics (logic of argument (LA) and logic of obligation and time (LOT)) to support decision making and action control [98].

The PROforma formalism is based on the domino model [98], which is a generalized model of clinical decision-making and protocol management and can be seen in Figure 4.

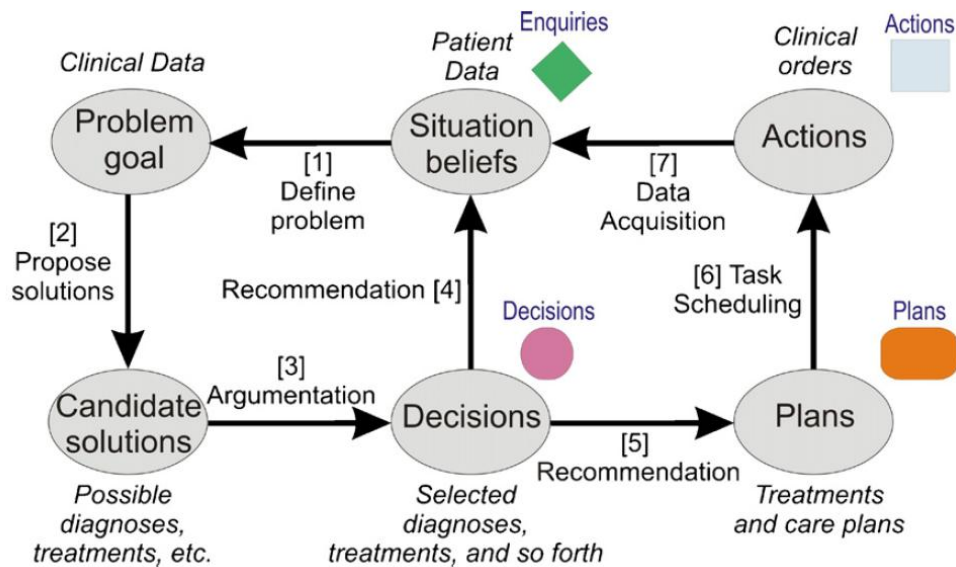


Figure 4 - The domino model, the basis of the PROforma language [26]

Nodes of the domino model represent information relevant to a particular clinical situation (e.g. facts about a patient's history, decisions, tasks in progress, or planned actions). Arrows represent inference procedures that derive conclusions from available information. The inference procedure uses information shown at the arrow's tail in conjunction with information from a patient record and/or general medical knowledge base, in order to generate information shown at its head.

The inference process according to the domino model (Figure 4) is the following:

1. Initially, the computing agent is given a set of beliefs.
2. From the beliefs, it identifies the problem(s) and infers goals (e.g. to diagnose or treat a disease).
3. Then it finds various solutions to these goals.

4. If there are multiple options (such as alternative diagnoses or treatments) the system must consider the arguments for and against these alternatives and make decisions based on the validity and ranking of these arguments.
5. A decision may commit to new beliefs, which could start the cycle all over with now new information on hand (e.g. new diagnosis for the patient).
6. Alternatively, it could commit to a recommended plan of action instead, to achieve a goal (e.g. a clinical care plan).
7. There is task scheduling and management needed as plans may consist of a set of actions carried out over time.
8. Finally, an action will often produce postconditions that change the patient's state.

This means that decision making is performed by the four nodes to the left, and planning (including scheduling) and enactment is done by the two nodes to the right.

While the domino model provides a good framework for defining the formal semantics of PROforma, it is rather abstract. To provide a precise, public definition of PROforma authors in [42] present a high-level overview of the syntax and the operational semantics. A much more detailed description can be found in [99]. The syntax is provided in *Backus Naur Form* (BNF) and an operational semantics for the language is specified in terms of a combination of state machine and process flow models.

The syntax of PROforma can be divided into two parts: (1) the syntax of the high-level guideline structure, which defines how the definitions of tasks and other guideline components should be arranged and separated, and (2) the syntax of the expression language that defines the forms that PROforma allows logical conditions and mathematical expressions to take.

The UML class diagram below [Figure 5] shows the concepts (i.e. object types) of the PROforma language together with the relationships that are defined among them.



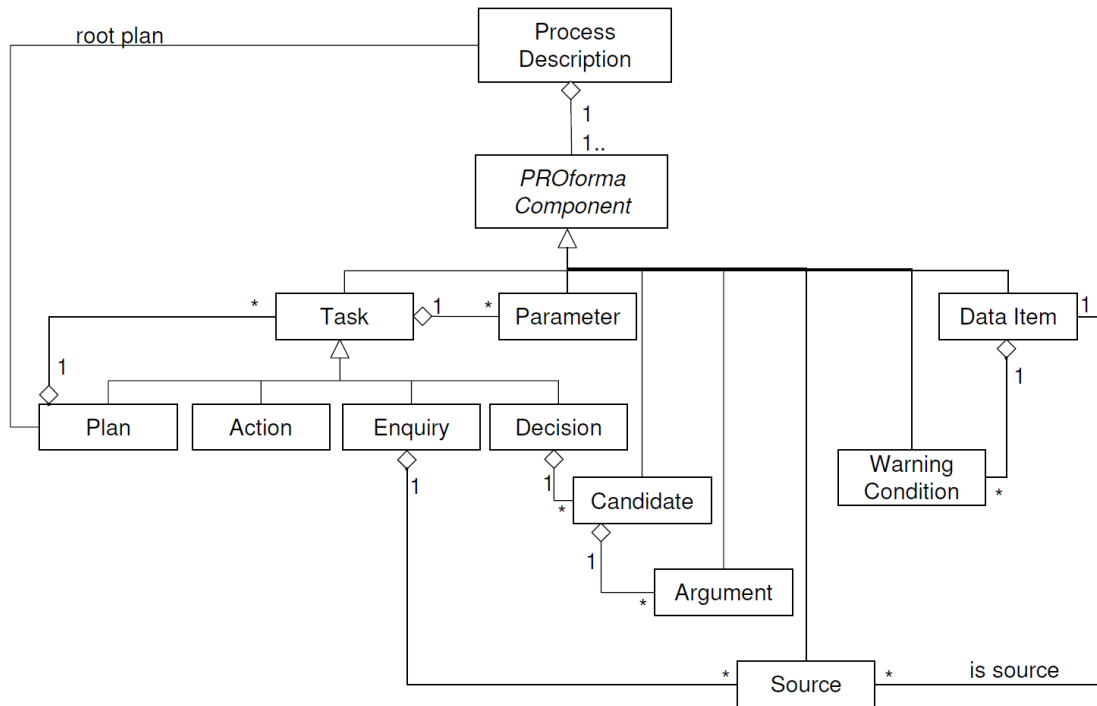


Figure 5 - The inheritance tree of PROforma component object types [36]

According to [82], one aim of the PROforma project is to explore the expressiveness of a deliberately minimal set of modeling constructs. As shown in Figure 5, a guideline is modeled as a set of *tasks* and *data items*. The tasks are organized hierarchically into *plans*. The PROforma task model divides tasks into four classes:

1. Actions represent some procedure that needs to be executed in the external environment (e.g., administering a drug or updating a database).
2. Enquiries represent points in a guideline at which information needs to be acquired from some person or external system.
3. Decisions are points at which some choice has to be made, either about what to believe or about what to do.
4. Plans are collections of tasks that are grouped together for some reason, perhaps because they share a common goal, use a common resource, or need to be done at the same time.

PROforma guidelines can be described graphically using a diagram-based convention (similar to UML Activity Diagrams) in which nodes represent tasks and arcs represent scheduling constraints. In these diagrams, squares represent Actions, circles represent Decisions, lozenges represent Enquiries, and

round-edged rectangles represent Plans. These four sub-classes and their respective icons are represented in Figure 6 below.

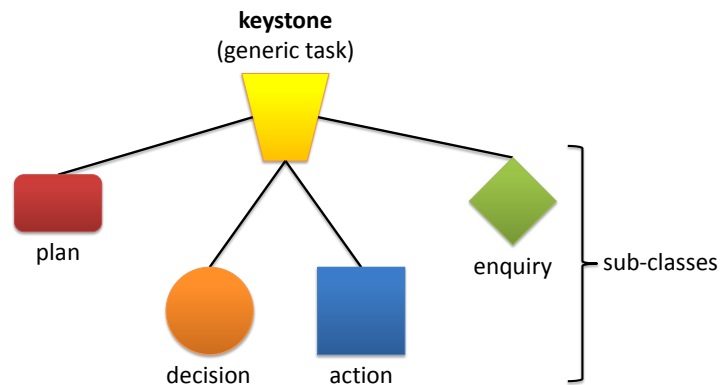


Figure 6 - Icons of the generic and specific task types in PROforma [97,102]

The contents of an example PROforma plan can be seen in Figure 7.

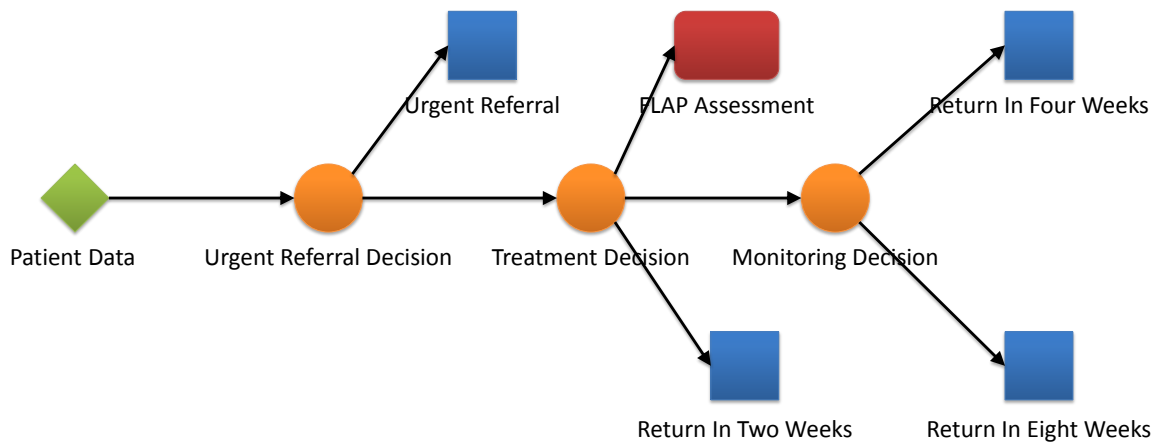


Figure 7 - Contents of an example for a PROforma guideline (plan) [36]

Scheduling constraints express necessary, but not sufficient, conditions for the activation of tasks, which means that the completion of a task does not imply the start of a task it precedes. Tasks in PROforma can also have a number of graphically invisible properties (i.e. attributes) whose values determine how they are to be interpreted. The value of a property may be a scalar value (e.g., an integer), an expression, or it may be an object, with its own set of properties. All tasks share attributes describing goals, control flow, preconditions, and postconditions. Logical preconditions (truth-valued expressions) of tasks are evaluated after their scheduling constraints are satisfied, and tasks are only activated if both their scheduling constraints and their preconditions are satisfied. There are other concepts in PROforma

not explained here that alter behavior, such as termination and abort conditions, and arguments for and against candidates (alternatives) at decisions [36].

### Expression language

In order to define conditions and arguments PROforma includes an expression language. The expression language includes the usual logical, arithmetic, and comparison operators, as well as functions that evaluate the execution states of tasks (i.e. whether they have been, or need to be performed) as well as the values of data items [36].

### Execution

During the enactment of a guideline, each task may undergo multiple state transitions. A task may be in one of four states: *dormant*, *in\_progress*, *completed*, or *discarded* (Figure 8).

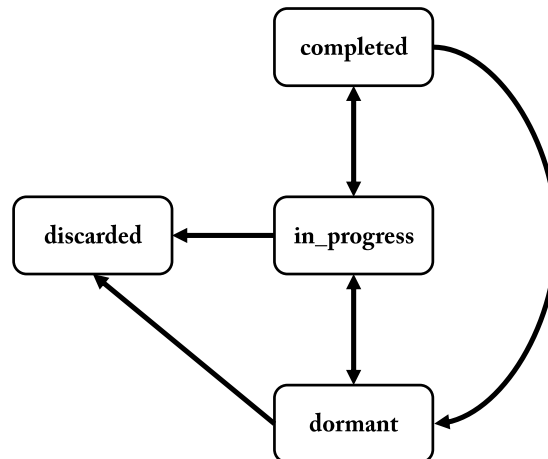


Figure 8 - PROforma task state transitions [42]

The PROforma semantics do not impose any real-world interpretation of task states. However, [42] provides a loosely defined interpretation. According to [42], a task is (1) dormant if it has not been started, and it is not yet possible to say whether it will be started, (2) in\_progress if it has been started, (3) discarded if the logic of the guideline implies that it either should not be started or should not be completed, and (4) completed if it has been done.

### Implementation

In execution time, a PROforma process description (i.e. CIG) is loaded into a software component referred to as the PROforma Engine, which maintains a record of the dynamic state of the process. This includes information on which tasks have been performed, decision on which need (or need not) be

performed, and the values of any data items associated with the process. The EE also implements a set of (“public”) operations to read or change the state of the engine and its guidelines in certain predefined ways (e.g. setEngineTime, loadGuideline, evaluateExpression, etc.). Some of these actions will be executed automatically, however the execution of a guideline often require actions to be performed by external actors (e.g. clinicians), who will interact with the engine via some set of user interfaces [36].

There are two main implementations of a PROforma engine currently available, the Arezzo implementation [103], which is available commercially from InferMed Ltd. (London, UK) and the Tallis implementation [104] from Advanced Computation Laboratory of Cancer Research. The implementations are similar, although the Arezzo implementation is based on a somewhat earlier PROforma language model. PROforma technology includes a suite of guideline authoring and execution software that incorporate CASE and verification tools. It has been shown to meet specific requirements of medical applications even though the language and tools are generic [98].

The Tallis implementation is composed of the following components:

- Composer: graphical knowledge authoring tool for the creation, editing and graphical visualization of CIGs (see Figure 10)
- Tester: for debugging of CIGs
- Parser: for reading and writing CIGs in text format
- Engine: for enactment (execution) of CIGs
- Web IF: Java Servlets for runtime visualization and control (of enactment)

The components of the Arezzo implementation – similar to the ones Tallis has – can be seen in Figure 9:

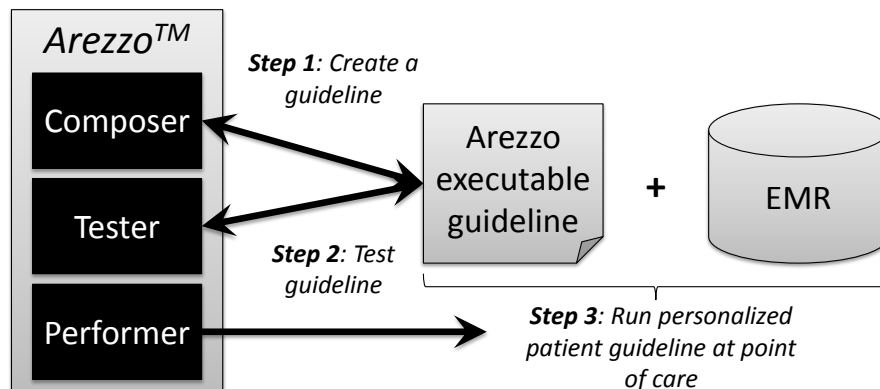


Figure 9 - Arezzo's PROforma-based architecture [26]

A database server for accessing patient records is a required, but not included component. The Arezzo implementation is similar to that for Tallis, but it has a combined engine and tester environment, called the Performer. Another difference is that while Arezzo applications are designed to run on MS Windows platforms, Tallis was designed for delivering web-based services and has a Java-based implementation [97].

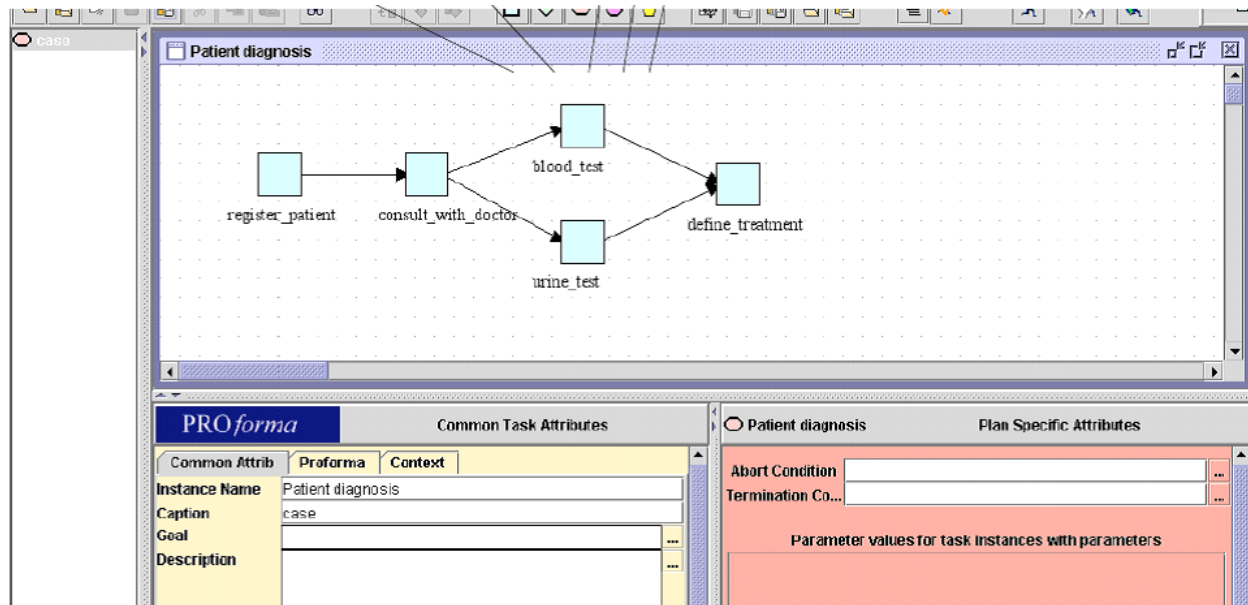


Figure 10 - PROforma patient-diagnosis scenario modeled in the Tallis composer [82]

### Advantages and limitations

Some of the advantages of PROforma are:

- PROforma has two independent implementation platforms, namely Arezzo and Tallis.
- PROforma has a graphical notation.
- PROforma has a mapping to UML Activity Diagrams defined in [105].

Some of the limitations of PROforma are listed in [42] and [36]:

- It was reported in [36] that the lack of notational convenience (e.g. incomplete specification of the graphical notation) led knowledge engineers to use UML activity diagrams to create models to be used during the knowledge acquisition and analysis phases of the project instead of using PROforma process descriptions. These UML diagrams were translated into PROforma during the implementation of the project.

- The PROforma suite does not prevent guidelines containing syntactical errors (in expressions) from being loaded or enacted.
- There is limited support for expressing constraints. There is a need for improved temporal reasoning, including the ability to represent temporal constraints about the state of tasks (e.g., when a task was completed or a data value was acquired) and the evolution of values, (e.g., the ability to define temporal predicates such as “increasing” or “decreasing”).
- The PROforma expression language is not Turing complete. This is because the PROforma expression language does not provide any way of expressing functions that involve recursion or iteration. The creation of recursively evaluated expressions is possible though the use of iterative tasks, however this is a can prove to be a cumbersome solution.
- There is no way to avoid the duplication of PROforma arguments, should they need to be attached to multiple candidates (see Figure 5).
- There is limited support for abstraction and information hiding. It is frequently useful to conceal some aspects of a guideline description and reveal others in order to distinguish between the essential logic of a process and the information that is required by some particular implementation of that logic.
- There is limited support for the definition of classes of tasks. It is frequently the case that a guideline description will contain several tasks that have the same parameters and can therefore be regarded as forming a class. For instance, tasks that involve altering a patient's medication might be grouped together into a class whose common properties are used to express the modifications needed. PROforma provides facilities for the definition of task classes, but with not enough expressive power (for instance they do not allow task classes to be grouped into hierarchies).
- There is limited support for expressing structured data. Data types in a PROforma process are limited to atomic values (e.g. an integer or string) or ordered list of atomic values, all of the same type.

- While UML Activity Diagrams allow a transition between two states to be given a guard condition and will only occur if that condition is satisfied, this is not directly supported in PROforma. In PROforma, transitions between tasks can be constrained using scheduling constraints (arcs), however it is not possible to attach a guard condition to a scheduling constraint. Instead, preconditions are attached to the tasks themselves. This makes the translation of guarded transitions into PROforma difficult as one needs to model the guard transition as a precondition of a task.
- Currently, the specification of PROforma does not make use of medical terminological standards.
- Mapping of the abstract CIGs onto a patient record system requires Java code to be written. This problem is similar to the curly braces problem of the Arden Syntax.
- The PROforma language does not address any data security issues.

## GLIF

The Guideline Interchange Format (GLIF) [106,107] is a computer-interpretable, object-oriented process knowledge model designed for the representation, sharing and execution of CPGs. GLIF has associated tools under development for supporting guideline authoring and execution [107].

### Development

The first published version of GLIF was GLIF2 (GLIF version 2) in 1998 [107]. The latest version of GLIF is version 3.5, also known as GLIF3, was published in 2000 [108] and updated in 2004 [109]. It has been developed by the InterMed Collaboratory<sup>32</sup>, which includes groups from Stanford, Harvard, Columbia and McGill universities [82].

---

<sup>32</sup> The word "collaboratory" in InterMed Collaboratory is a word play created from words "collaboration" and "laboratory".

GLIF is built on top of the most useful features of other guideline formalisms and it incorporates standards that are used in health care. Its expression language was based on the Arden Syntax and its medical data model is based on the HL7 Reference Information Model (RIM)<sup>33</sup> [82].

### Use

GLIF has been used in the modeling of a diabetes foot guideline, which has been locally adapted to the needs of primary care physicians in outpatient clinics in Israel. There, it was linked with a web-based EMR, and was enacted using a GLIF-specific execution engine, called the *Guideline Execution Engine* (GLEE) [110]. Another example, where a GLIF-based guideline execution was implemented, includes post-CABG (Coronary Artery Bypass Grafting) patient care planning at Columbia-Presbyterian Hospital and Columbia University. Here full integration with the local CIS was not completed [107]. Further examples demonstrating the effective use of GLIF3 for encoding and testing various CPGs, including childhood immunization, cough management, and hyperkalemia patient screening, can be found in [111].

Despite these examples and the fact that GLIF was designed to be an open standard [34], only a subset of GLIF became a true standard. GLIF's expression language, the *Guideline Expression Language, Object-oriented* (GELLO), was adopted as an international standard by HL7 International and ANSI in 2005 [112]. The new version – GELLO Release 2, developed in coordination with the HL7 Clinical Decision Support TC (CDSTC) – was completed and approved by ANSI in June 2010 [87]. The rest of GLIF has not received nearly as much attention and after 2009, and its website [113] disappeared.

### Syntax and semantics

GLIF was designed to support guideline modeling as a flowchart of structured steps, where steps represent clinical actions and decisions. For its models, GLIF3 requires a formal definition of decision criteria, action specifications and patient data.

GLIF supports three separate abstraction layers for specification: (1) an abstract flowchart level, (2) a computable level, and (3) an implementation level of specification. Level 1, the abstract flowchart level,

---

<sup>33</sup> More on HL7 RIM can be found in the “Data models” section under the “Other important medical formalisms, frameworks” section.



helps authors and users view and understand guidelines. Level 2, the computable level, formally defines logical criteria, definitions of patient data items, clinical actions and the flow of the guidelines. Level 3, the implementation level, includes non-shareable, institution-specific details, which enable guidelines to be incorporated into operational clinical information systems. Thus, shareable components of a guideline are explicitly separated from institution-specific or vendor platform-specific (non-shareable) components [107].

GLIF uses the Protégé ontology editor and knowledge-base framework [114,115] as its modeling environment. Conforming to Protégé modeling methodology in GLIF3 models the process of clinical care is encoded as the algorithm of a guideline and guidelines are represented as specific guideline instances. GLIF contains various classes and their attributes to represent CPG knowledge and the complex relationships among them. An overview of the high-level concepts and their relationships represented as a UML class diagram can be seen below in Figure 11 [109].

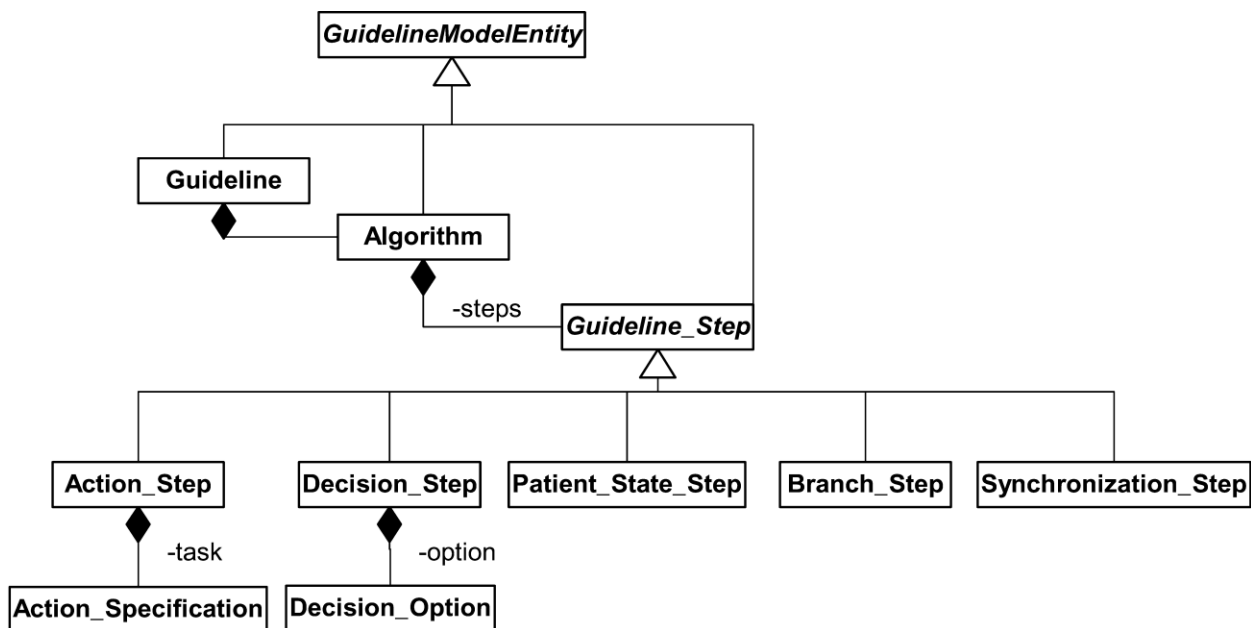


Figure 11 - Overview of the high-level classes in GLIF3 [109]

Within an algorithm, instances of five types of tasks, which are called guideline steps, can be encoded and linked together in a flowchart to specify their scheduling and coordination during guideline application [116]. These five main process-modeling entities, each of which is a subclass of the abstract Guideline Step class are:

- the Action Step, which is a block used to specify a set of recommended tasks (clinical or computational) to be performed without constraints set on the execution order. GLIF3, now allows nesting of sub-guidelines in the model, thus multiple views to the care process with different granularities can be defined [107].
- the Decision step, which are used for conditional and unconditional (user-selected) routing of the flow to one out of multiple steps. (This step is a merged step combining a Case Step and a Choice Step from GLIF 3.4).
- the Branch and
- Synchronization steps, which are used for modeling branching of multiple concurrent paths and their synchronization (merging or parallel branches).
- the Patient-State Step, which is a guideline step used for specifying an entry point(s) to a guideline and for describing the clinical state of the patient (pathophysiological or management states in the specific contexts of a guideline's application).

These concepts are used to formulate a guideline algorithm and provide an overview of the decision-making process of a guideline [111]. The visual representation of the concepts and a simple GLIF patient-diagnosis scenario built using them can be seen in the figure below (Figure 12 from [82]).

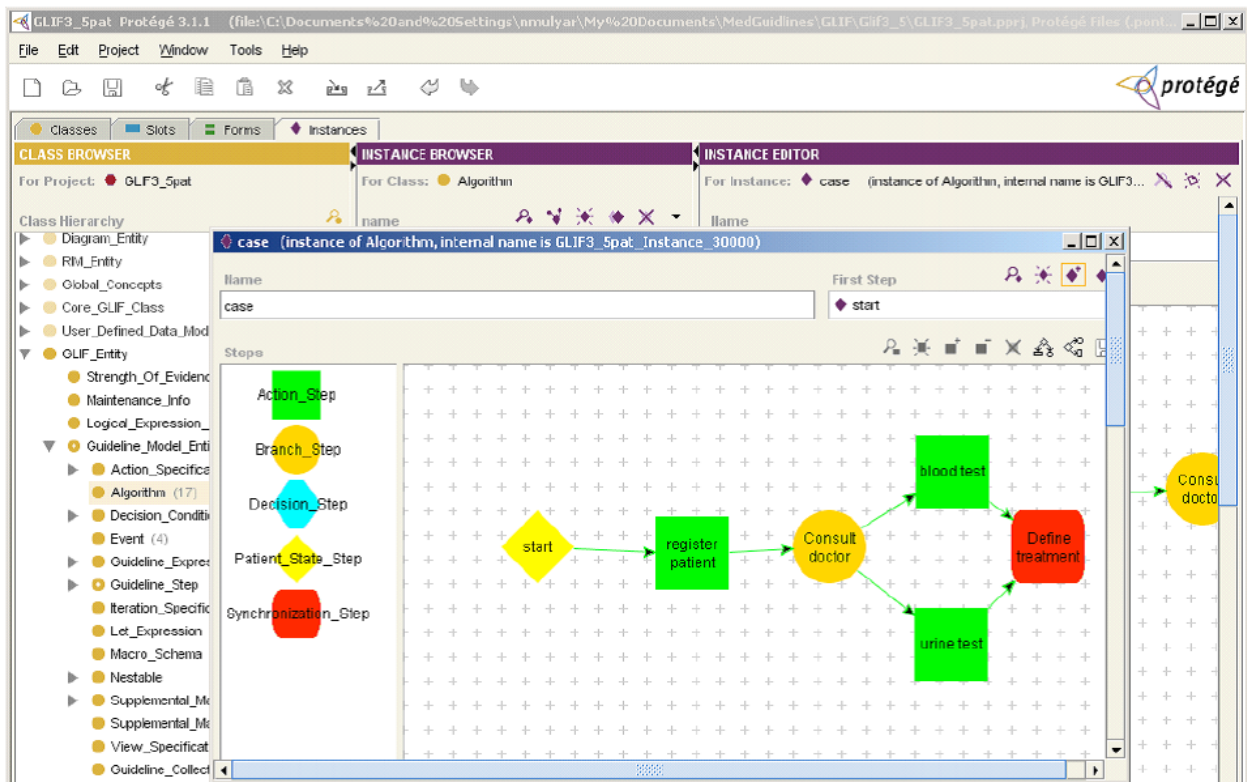


Figure 12 - Patient-diagnosis scenario modeled in GLIF3.5 using Protégé [82]

### Execution

The execution semantics of GLIF models are implemented by the GLEE. Figure 13 shows a schematic state machine diagram of a guideline step.

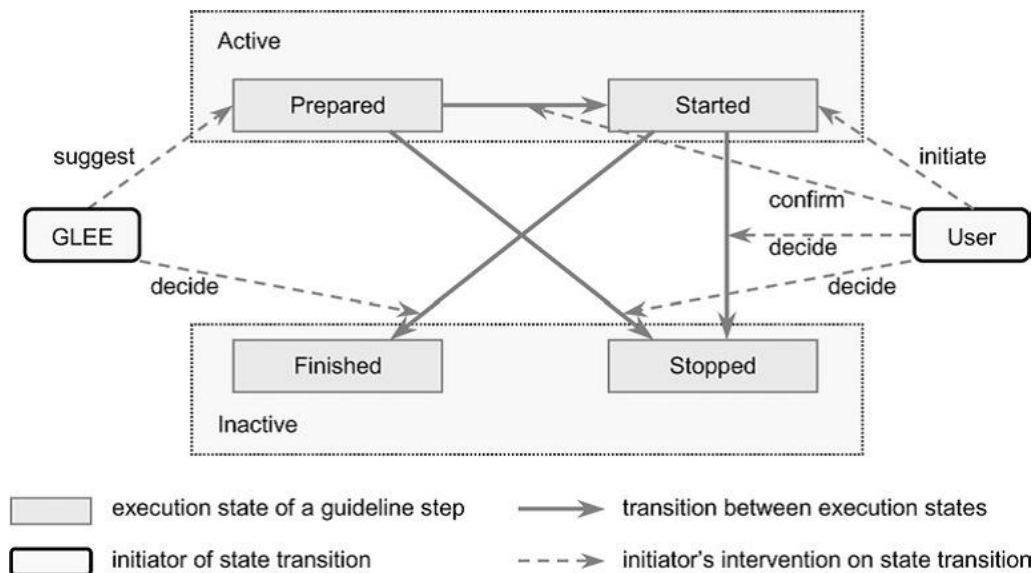


Figure 13 - Execution states of a GLIF guideline step and possible transitions in-between [116]

During operation, GLEE suggests a guideline step deemed executable and puts it into the prepared state. Users then decide whether to follow GLEE's suggestion or to deny it by stopping the suggested step and initiating the start of another step. Users also have the ability to stop a started step that is no longer relevant. Guideline steps deemed completed by GLEE finish their execution [116].

### Data types

Clinical data in GLIF are encoded as data items. These data items are then referenced by expressions, which are used to express decision criteria and patient state. Clinical events are encoded as triggering events, which are used to activate specific clinical tasks [116].

### Expression language: GELLO

Guideline Expression Language, Object-oriented (GELLO) [117,118], which was recently accepted as an HL7 and ANSI standard, is a vendor and platform-independent, extensible, object-oriented, side-effect-free, and executable expression language [82]. GELLO can be used for expressing and sharing decision logic, eligibility criteria, calculations, patient state definitions, conditions, and system actions. GELLO allows the specification of expressions in the form of (1) queries to extract data from EMRs in CDSS, and (2) logical rules to manipulate data and evaluate decision criteria by building up expressions to reason about particular data features and values such as ones found in guidelines [34]. GELLO solely focuses on specifying logical expressions and it was not designed to support specification of entire clinical algorithms.

GELLO's original goal was to serve as a procedural component for the higher-level guideline format (GLIF), but since its adoption as a standard, it has been extended to serve a similar functionality for the current HL7 knowledge representation standard, Arden Syntax [119].

GELLO was based on OCL<sup>34</sup> and much of the functionality of OCL has been integrated into GELLO to provide a suitable framework for manipulation of clinical data for decision support. It provides basic data types and a mechanism to reference underlying standard data model (vMR) in an object-oriented fashion [119]. The authors of [112] provide further details on how GELLO expressions are parsed, compiled, used for data aggregation and evaluation, while [119] provides some examples for (1) queries and (2) expressions.

---

<sup>34</sup> UML's Object Constraint Language (OCL) [120]

### 1. Queries:

```
Observation.select(coded_concept='03245')  
Observation.selectSorted(coded_concept="C0428279")
```

### 2. Expressions:

```
calcium.notEmpty() and phosphate.notEmpty()
```

which returns true if the variables calcium and phosphate are not null, and

```
renal_failure and calcium_phosphate_product > threshold_for_osteodystrophy
```

which returns true if the patient has renal failure and the product of calcium and phosphate exceeds a threshold signifying osteodystrophy.

## Implementation

### Modeling environment: Protégé

As mentioned in the previous section, GLIF uses the Protégé ontology editor and knowledge-base framework as its modeling environment. Protégé provides the means for creating and validating models in GLIF with the help of an ontology schema and a graph widget, which have to be loaded into the Protégé-2000 environment [82]. As GLIF models are captured in Protégé, they automatically have an XML-based syntactical representation as well [34].

Protégé has been used as a GUI for the development of multiple CIGs. These studies indicate that Protégé can be used effectively to validate the encoding of CPGs in the GLIF3 format [111] and potentially extend them through its inferencing engine<sup>35</sup>. This is showcased in [111], where the overall goal of the presented study was to illustrate the steps involved in encoding a guideline in GLIF3 through a case study of a depression screening and management CPG for a nursing decision support system (DSS).

### Execution engine: GLEE

The main purpose of the execution engine for GLIF, called the GLEE, is to assist in the implementation of GLIF3-encoded CIGs and provide a test environment, where an investigation whether an encoded CIGs faithfully reflects a CPG encoder's intention can be performed [111]. GLEE acts as a CDSS by (1)

---

<sup>35</sup> Protégé's inferencing engine is called *Pellet*.

interpreting guideline knowledge encoded in the GLIF3 format, (2) integrating it with patient data, and (3) generating recommendations tailored to individual patients. This process can be observed in Figure 14. In addition to clinical decision support, GLEE aims to be used for quality assurance, guideline development, and medical education as well.

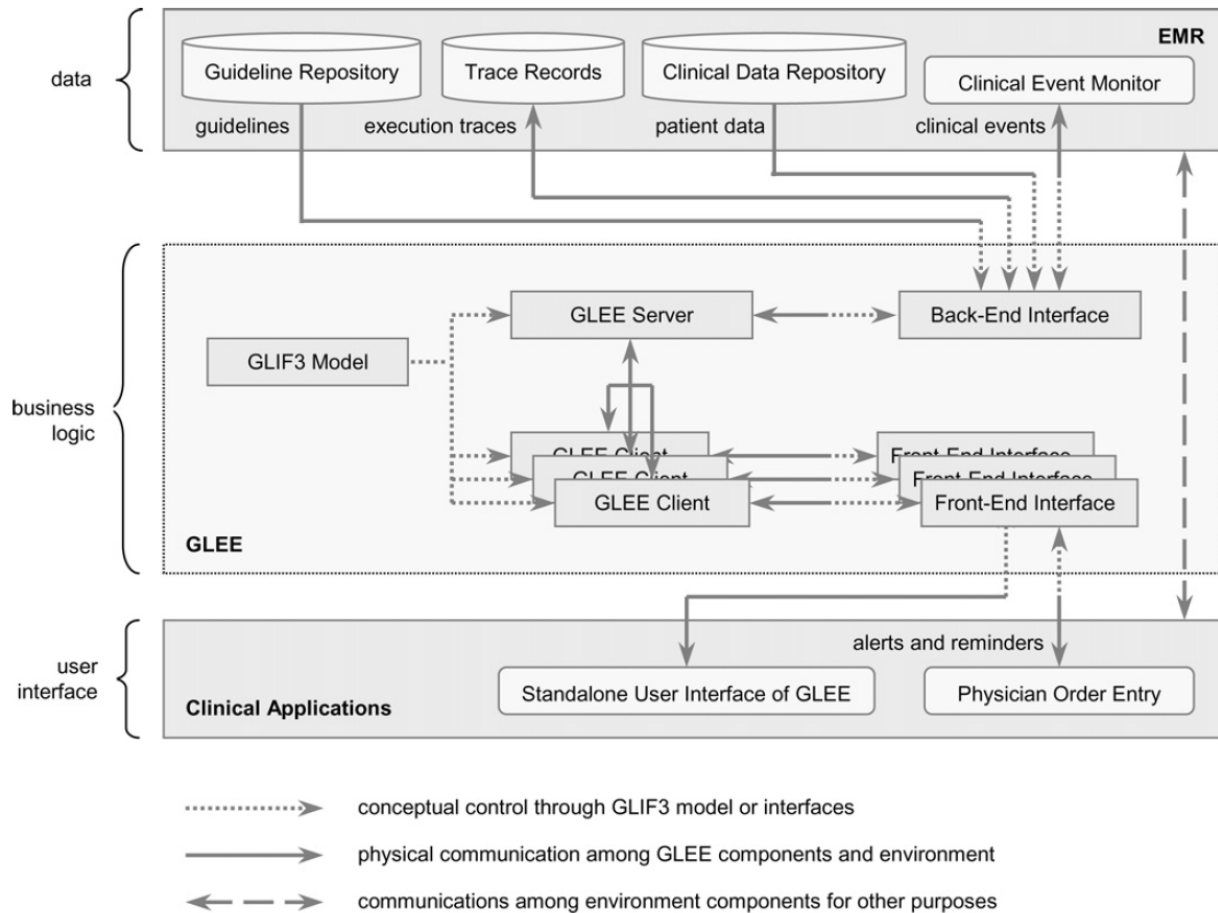


Figure 14 - The internal structure of GLEE (of GLIF3), and its interactions with the environment [116]

GLEE is built as middleware that is intended to be integrated with the CISs at a local institution through predefined interfaces to its EMRs and clinical applications. Figure 14 shows the GLEE middleware integrated into a host CIS. The architecture can be classified into three conceptual layers: (1) data to support execution, which includes repositories for clinical data and GLIF3 models, (2) the core components implementing the execution logic (i.e. business logic), and (3) the interfaces both to users and the host environment (UIs and data interfaces).

The communication between GLEE and the EMR back-end enables GLEE to access various resources in the local environment, such as retrieval of patient data, and monitoring of clinical events. The

communication between GLEE and associated clinical applications at the front-end enables the integration of the decision support services provided by GLEE, such as alerts and reminders, within a clinician’s workflow. The architecture allows the simultaneous instantiation of multiple GLEE clients. It also features a standalone GUI (Figure 15) for simulating the process of guideline execution on individual patients; however, it is used only for development and demonstration purposes. GLEE is currently implemented in Java [116].

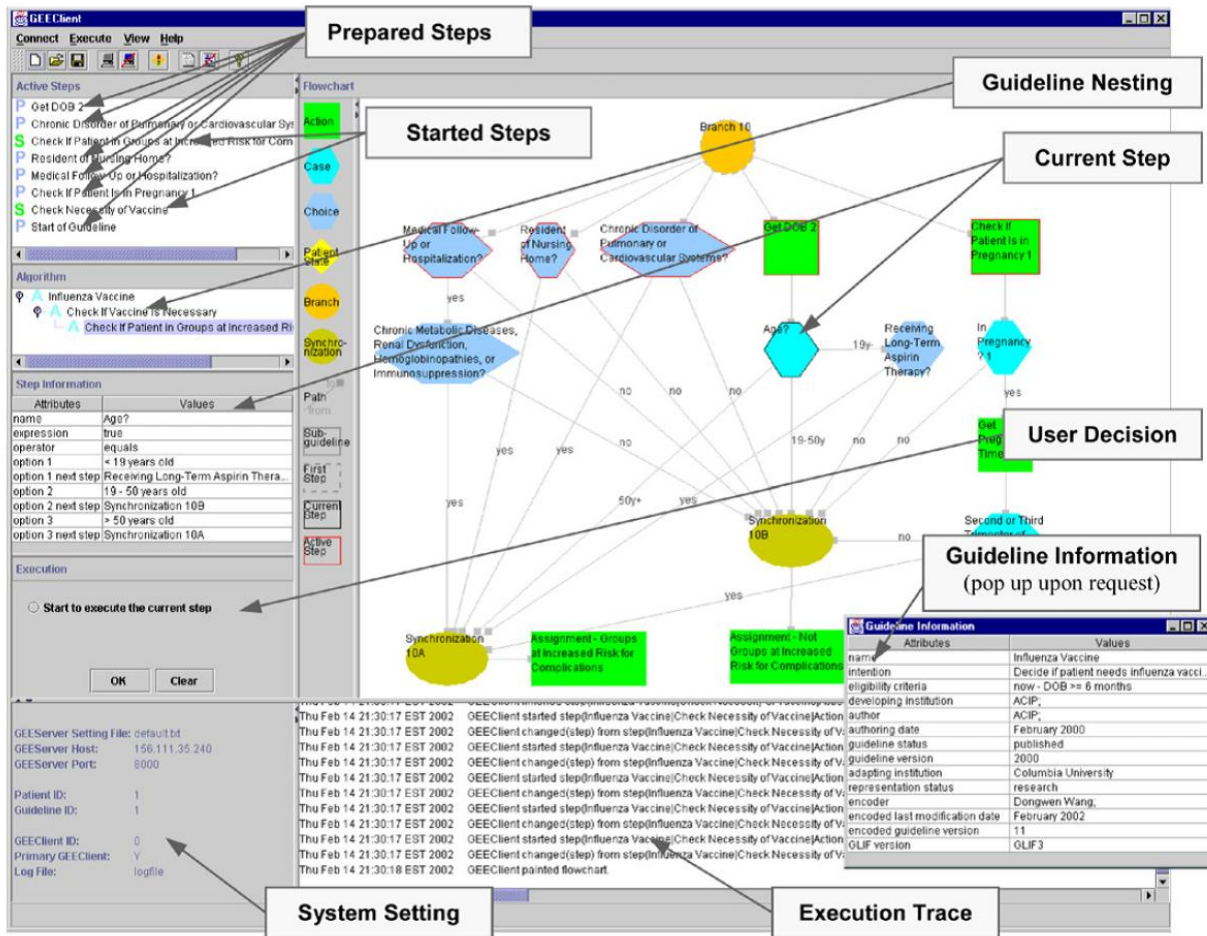


Figure 15 - Screenshot of GLEE’s standalone GUI during development and testing (client side) [116]

Figure 15 demonstrates GLEE’s standalone GUI, on which the algorithm of a GLIF3 guideline for influenza vaccination is shown as a flowchart at the upper-right portion of the screen. The upper-left portion of the screen provides information on the list of active steps, the hierarchy of algorithms, and detailed information on the currently highlighted step. The lower part of the screen shows the setting of the current client and the execution trace. The pop-up window provides maintenance information of the guideline.

## Advantages and limitations

Some of the advantages of GLIF are:

- GLIF allows for the formal specification of CIG with an associated modeling environment (Protégé) and execution engine (GLEE).
- GLIF incorporates GELLO<sup>36</sup>, a now standard formal expression language for specifying decision criteria and patient states.
- GLIF uses a layered patient data model to enable GLIF3 steps to refer to patient data items defined by a controlled terminology that includes standard medical vocabularies (such as UMLS<sup>37</sup>), as well as standard data models for medical data (such as HL7 RIM, or vMR).
- In GLIF2, the attributes of the main constructs were defined as text strings that could not be parsed. This prevented the resulting guidelines to be used in clinical inferencing during execution. To address this problem, GLIF3 extended the GLIF2 specification with several new constructs, and requires a more formal definition of decision criteria, action specifications and patient data [107].
- Just like the Arden Syntax it facilitates a *ternary logic* (i.e. true, false, unknown) to support limited uncertainty [32].

Some of the limitations of GLIF include:

- According to multiple sources [34,82,111], GLIF supports sharing of computer-interpretable clinical guidelines across different medical institutions and system platforms. Although authors stress the importance of sharing, there are no good published examples.
- GLIF authoring is currently bound to the Protégé environment, which might be a limiting factor when it comes to clinical adoption.

---

<sup>36</sup> Before GELLO was adopted, GLIF had been using an expression language called GEL, which was based on the Arden Syntax's logic grammar.

<sup>37</sup> More on UMLS can be found in the "Other important medical formalisms, frameworks" section.



## **Asbru**

The *Asbru* modeling language was created as part of the Asgaard framework in the Asgaard/Asbru project [31,33], the aim of which was to provide an architecture that supports the design and the execution of skeletal plans by a human executing agent other than the original plan designer.

### Development

The Asgaard project was introduced in 1998 [121] and has been an active area of research since. It involves collaborators from many places, including the Vienna University of Technology, Stanford, the Ben Gurion University and the Vrije Universiteit from Amsterdam [82,121].

### Use

According to [121] and [122], Asbru has been used in a fair amount of clinical applications (e.g. diabetes, jaundice, breast cancer and neonatal intensive care), all of which are only prototype applications. Asbru sources for some of these examples can be found in [123].

### Syntax and semantics

Asbru is a time-oriented and intention-based “skeletal plan” representation language that is used for the specification of clinical protocols. Asbru’s (skeletal) plans facilitate reuse by capturing only the essence of domain-specific procedural knowledge, thus leaving room for execution-time flexibility in the achievement of particular intentions. In a plan, there are concepts defined for (1) characterizing plan attributes such as intentions, conditions, and effects, (2) ordering of plans, and (3) defining temporal dimensions of states and plans. Uncertainty in temporal scope and parameters can be expressed by bounding intervals [82]. Asbru plans are written in XML, the schema specification is given in [124]. An overview of the high-level concepts and their relationships represented as a UML class diagram – taken from [124] – can be seen in Figure 16. An explanation of the diagram follows.

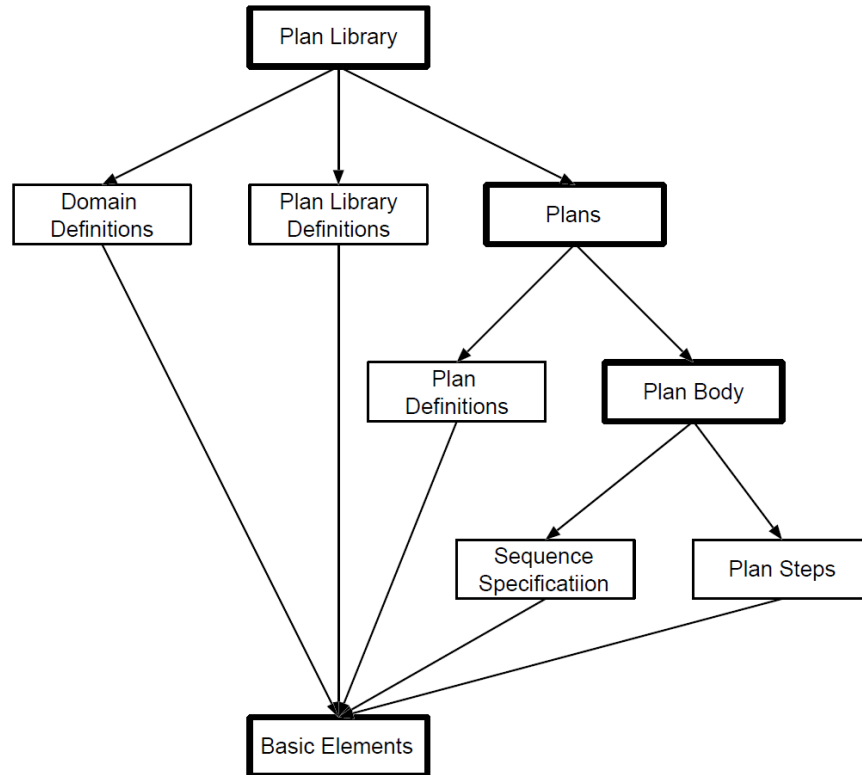


Figure 16 - Overall structure of a plan library in Asbru [124]

In Asbru, *plans* are contained in a *plan library* together with *plan library* and *domain definitions*, both of which help to separate declarative data abstractions from the procedural hierarchy of plans. Plan library definitions contain a set of reusable, domain-independent parameter definitions valid in the whole plan library, including functions, variables, constants, types and time definitions. Domain definitions provide an interface between the plan library and various environments in which the plans will be executed. In other words, they specify entities of the real world, which the plan library accesses.

A **plan** specifies a set of actions to be taken to reach a certain goal. The actions to be taken are specified in the *plan body* while the goal is given with the help of intentions and effects. Plans also have a return value and a set of conditions, which control their execution. A brief description of these elements is provided based on [124]:

- **Arguments** are values passed from the invoking or calling plan (called *parent*) to the invoked or called plan (called *child*).
- **Preferences** describe the costs, resource constraints, and responsible actor.

- **Intentions** are high-level goals of the plan, an annotation specified by the designer independently of the plan body. Intentions are temporal-pattern constraints, represented by temporal patterns of actions and states that should be maintained, achieved or avoided.
- **Conditions** guard the transitions among the states of a plan. For each condition, it can be defined whether the change in plan state occurs automatically or whether user confirmation is required. In addition, it can be allowed to the user to induce the change on plan state even if the condition is not fulfilled.
- **Effects** describe the relationship between plan arguments and measurable parameters by means of mathematical functions or in a qualitative way. A probability of occurrence can be denoted.
- The **plan body** contains set of plans to be executed in a particular way. It also specifies which of the child plans have to be completed successfully in order to terminate (complete) the parent plan successfully. A plan body can be one of the following:
  - *Subplan* is a set of plan steps performed in sequence, in parallel, in any order (any order, and unordered).
  - *Cyclical-plan* is a plan, which can be repeated multiple times.
  - *Single-step* is a single step of plan execution, consisting of either a plan activation, a variable assignment or the setting of the context.
  - *Refer-to* is a reference (link) to the plan body of another plan.
  - *To-be-defined* is a special tag declaring that this plan is not executable. It is an abstract pattern, which other plans can inherit, filling in the missing plan body.
  - *User-performed* is another special tag indicating that this plan is executed through some action by the user, for which reason it is not modeled in the system.

### Execution

A partial definition of the semantics of Asbru has been given using Structured Operational Semantics [125], however, according to [42], that effort does not provide sufficient information to permit others to implement tools. Taken from [124] and [33], the informal description of how Asbru plans are executed is described below.

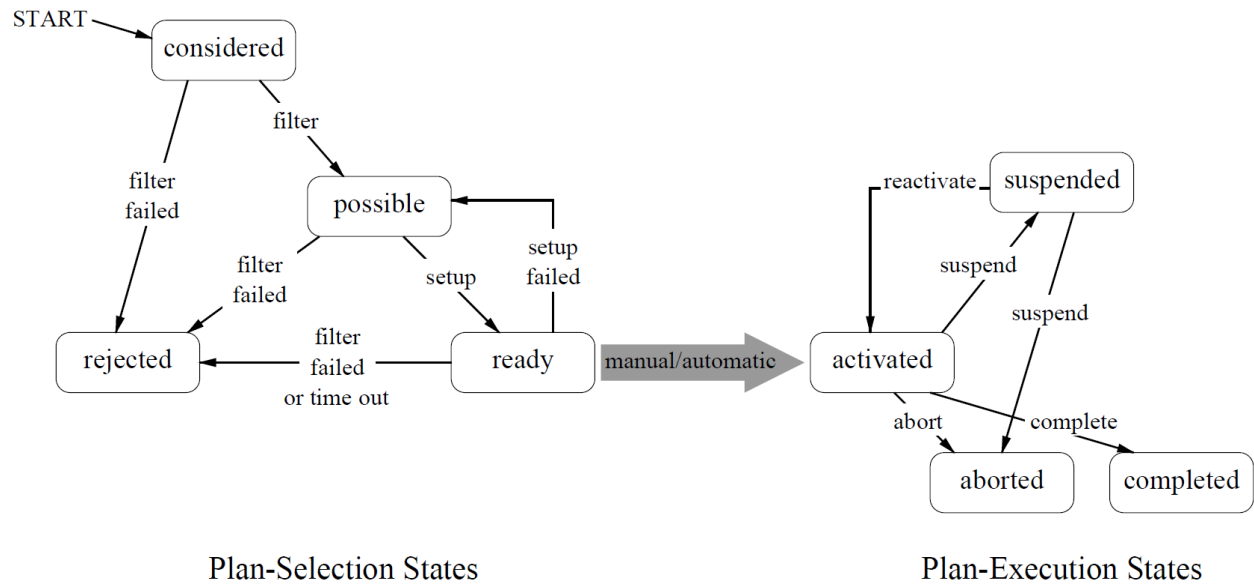


Figure 17 - State machine of an Asbru plan [124]

Generic library plans are executed by the execution interpreter (i.e. EE), which implements the state machine from Figure 17. According to the figure, plans have states that determine whether the plan is applicable and whether a plan instance can be created. Each plan is initially considered after which conditions control the transitions among the mutually exclusive states of the plan.

At execution time, if a plan was in *ready* state, it is instantiated. The decomposition of a plan into its sub-plans is always attempted, unless the plan is not found in the guideline library, in which case the plan is interpreted as non-decomposable. Non-decomposable plans might be decomposable into more primitive actions at a particular clinical site. A non-decomposable plan is executed by the user or by an external call to a computer program. Every (decomposable) sub-plan has the same structure. Thus, a sequential plan can include several potentially decomposable concurrent or cyclical plans.

### Implementation

#### Modeling and visualization environments

There are several tools that support the authoring and visualizing of Asbru guidelines. These include AsbruView [126–128], Delt/A [129,130], and CareVis [131,132] and the DeGeL framework [133–135].

AsbruView is a tool to make Asbru accessible to physicians, and to give any user an overview of a plan hierarchy, since Asbru plans cannot be understood by physicians with no or little training in formal methods. AsbruView's two main views, the topological view (TopoView) and temporal view (TempView),

are based on visual metaphors that make the underlying concepts easier to grasp. Figure 18 – from [82] – presents a patient-diagnosis scenario model in TopoView.

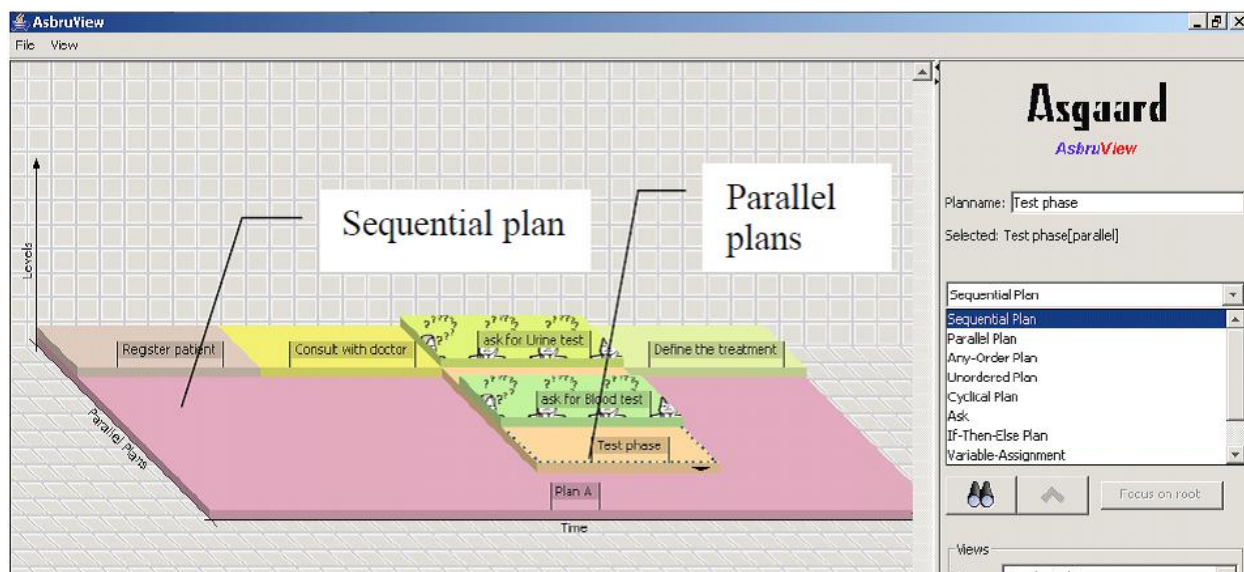


Figure 18 - Patient-diagnosis scenario model in AsbruView [82]

DeGeL, the Digital electronic Guideline Library, is a hybrid, multifaceted representation language and computerized, Web-based set of tools for storage, authoring, retrieval and enactment of hybrid Asbru guidelines<sup>38</sup>. DeGeL supports the gradual conversion<sup>38</sup> of clinical guidelines from text to fully formal, machine-readable Asbru representations [121]. Intermediate steps include the translation of text-based guidelines to structured text, which is an XML file, segmented and labeled by Asbru semantic tags. The DeGeL project has developed a large set of tools to support the development and implementation of guideline applications. Figure 19 illustrates how these components fit together.

---

<sup>38</sup> Hybrid representations of clinical guidelines include any combination of free-text, semi-structured text, semi-formal representation, and machine-comprehensible formats in a chosen target guideline ontology (in this case Asbru).

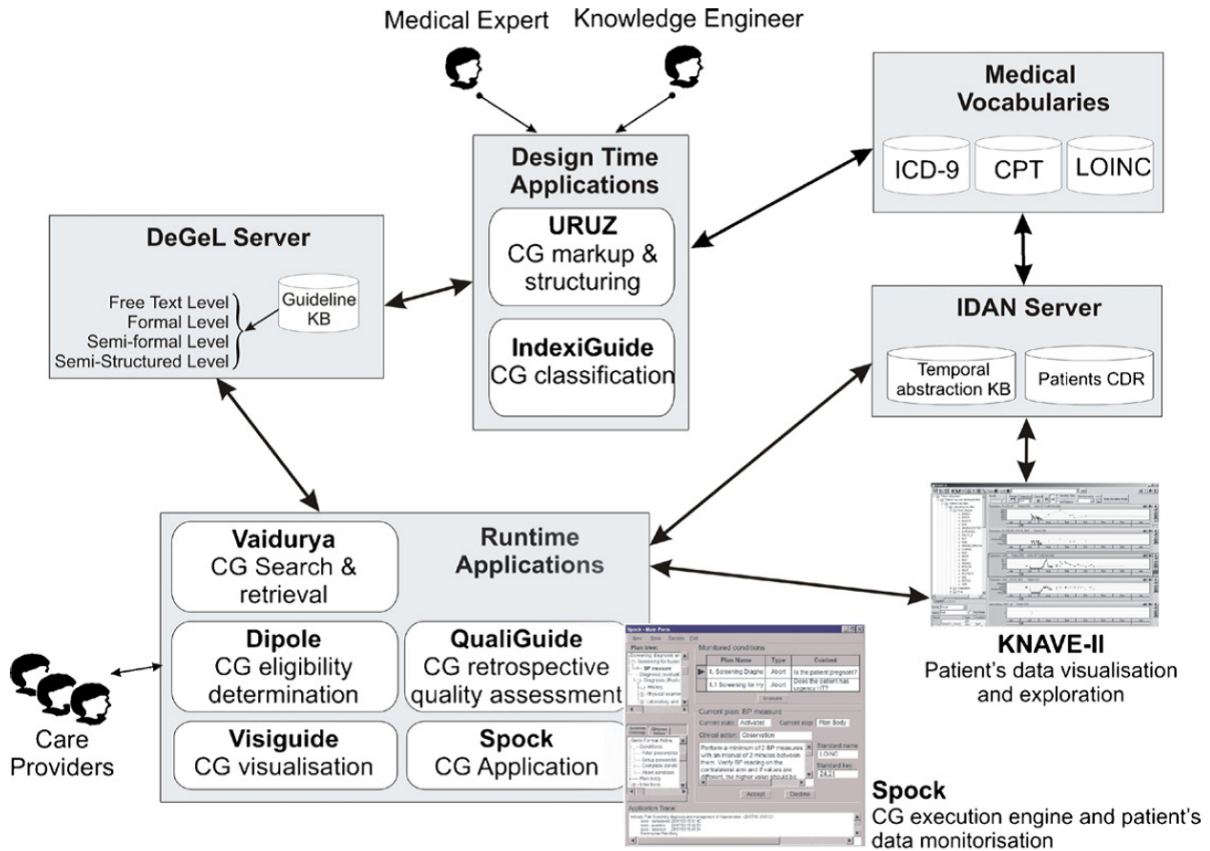


Figure 19 - DeGeL architecture for Asbru guidelines [26]

Support is provided for a wide range of tasks central to guideline-based care:

- During guideline specification:
  - verification of the guideline process specification (syntax)
  - validation of the guideline against its goals (semantics)
- During guideline execution:
  - determination of patient eligibility and guideline applicability
  - visualization of one or more potentially applicable guidelines
  - application (execution) of the guideline
  - quality assessment of providers' actions
  - modification of guideline or provider plans
  - evaluation of guideline effectiveness.

DELT/A – the Document Exploration and Linking Tool (with Addons) – formerly known as Guideline Markup Tool (GMT) was developed to provide a relatively easy way to translate free text into Asbru. It achieves this by displaying both the original text and the translation, and showing the user which parts of the Asbru code correspond to which elements of the original text [121].

CareVis is an interactive visualization tool designed to support (Asbru) protocol-based care. CareVis provides views for the complex underlying data structure of treatment plans and patient data. It is designed for use, for example, during guideline execution.

Another visualization tool [136] uses flowchart-like representation (similar to what Oracle BPEL Process Manager [137] uses) to conceptualize Asbru guidelines. Unfortunately, this tool only allows clinicians to interpret Asbru guidelines, not to edit them.

Execution engine: AsbruRTM

Asbru’s original execution environment is called the Asbru Run Time Modules (AsbruRTM) [138–140] and it is written in Java. In order to translate the XML-based Asbru plans to Java classes, the EE facilitates Castor, an open source data-binding framework for Java, which generates a Java object model out of Asbru’s XML Schema. AsbruRTM consists of three main modules: the data-abstraction unit, the monitoring unit, and the execution unit. These can be seen in Figure 20.

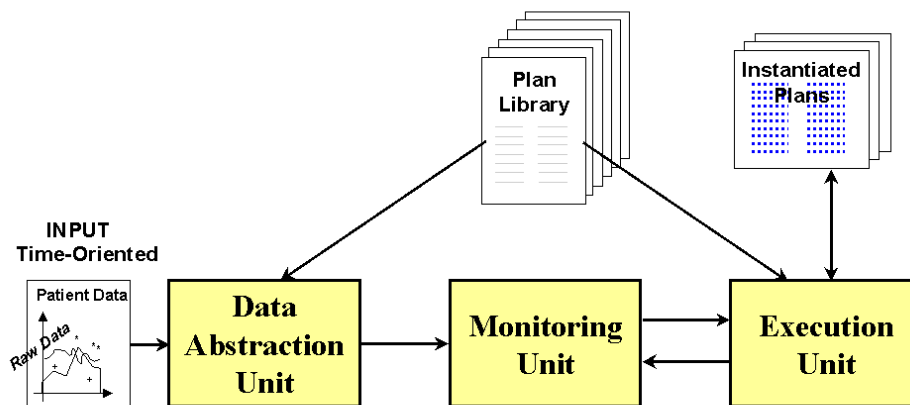


Figure 20 - The Asbru engine [138]

The role of the data abstraction unit is to feed the incoming data to the monitoring unit, by performing the transformation of information on two distinct data source types<sup>39</sup>: high-frequency domains (e.g. sensory measurements), and low-frequency domains (e.g. manually entered daily measurements). The resulting formatted data is received and stored in a list of observed parameter propositions by the monitoring module. How long each observation is valid is specified by the data abstraction unit. The monitoring module receives another list, which is specified by the execution module. This list contains temporal patterns, called the monitored parameter propositions. The monitoring module then periodically compares these two lists and if two elements match, the execution module is notified.

Mapping of plans and actual situations is accomplished by the execution module on three distinct layers:

1. Plan Synchronization: All plans in Asbru have time-annotated conditions for their start, successful completion and failure. These annotations provide flexible means for denoting temporal constraints on the duration of a plan. The execution of a plan lasts until its goal conditions are satisfied or a failure is reported.
2. Plan Adaptation: Plan adaptation is implemented as plan suspension and plan replacement. Plans stay active until they are either completed, aborted, or suspended. An alternative plan can be activated when the original one fails.
3. Replanning: Replanning can occur only if a plan fails or there is an explicit user request for it. If a top-level plan fails, the execution module looks for plans (in the plan library), which either have the same intentions, or have effects which remove the reason for the failure of the current plan. Additional user requests can be specified by either selecting a plan to be executed, an intention, or a goal to be achieved.

#### Execution engine: Spock

The DeGeL framework includes another EE engine for Asbru, called Spock [142,143]. The goal of the system is to assist providers to “apply guidelines over extended time periods in an intermittent fashion at the point of care”. What Spock implements is an extension to the original state transition model of an Asbru plan (Figure 17) with the ability to automatically process information coming from an EMR

---

<sup>39</sup> Extensions to this unit have been proposed in [141].



system; however, Spock focuses mainly on execution of semi-structured (hybrid) Asbru guidelines. Their extended execution model is defined with the help of UML Statecharts and can be seen in Figure 21.

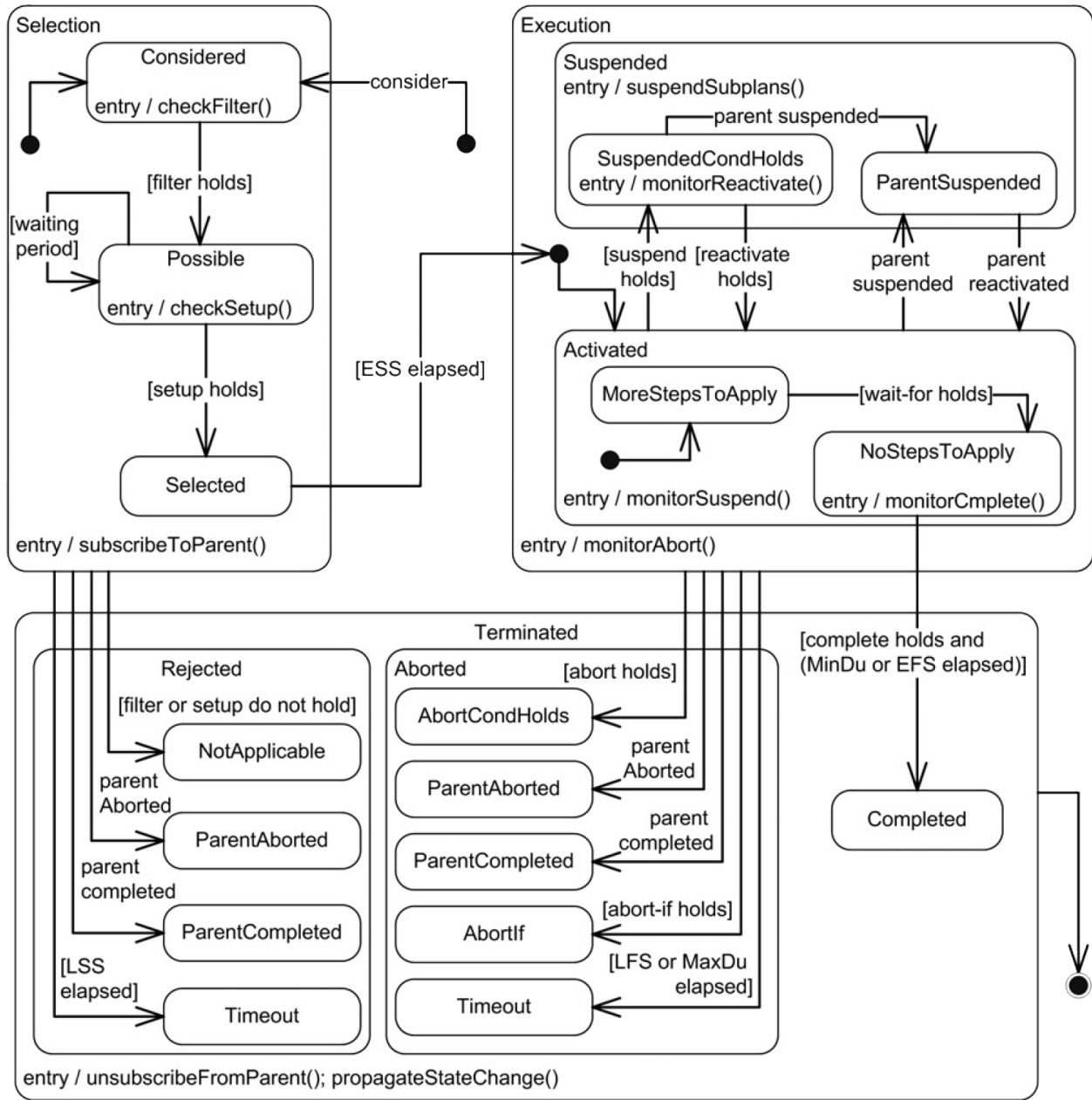


Figure 21 - Extended state machine of an Asbru plan in Spock [143]

### Advantages and limitations

Some of the advantages of Asbru are:

- Asbru uses sophisticated temporal structures, which allow the representation of uncertainty in start time, end time, and duration of time interval.
- Asbru attempts to provide support for the reuse of guidelines by providing multiple abstraction layers. Examples for this include (1) providing concepts for capturing domain definitions separately from guideline logic, and (2) providing abstract plans serving as placeholders for the implementation of location-specific tasks.

Some of the limitations of Asbru are:

- Asbru is a highly complex language, but seemingly makes no use of medical standard terminologies.
- As far as one can tell from available literature [139,144], Asbru's execution environment - is fairly rudimentary, as it only processes a subset of Asbru – called Asbru Light – and it lacks a configurable interface for interacting with guidelines being executed.

### Guideline modeling languages not selected for evaluation

There are many other guideline-modeling methodologies that this thesis does not discuss. Some of the most well-known ones include *SAGE*, *SpEM*, *EON*, *GEM*, *GUIDE*, and *PRODIGY*, none of which possess any significant features that were not covered by previously discussed approaches.

### Other important medical formalisms, frameworks and organizations

As it was discussed earlier in this thesis, **information exchange and reuse of knowledge are important factors** in the interoperability of CIG-based CDSS. Without trying to provide a complete picture, this section introduces a couple of medical formalisms, frameworks and organizations, mentioned in previous sections<sup>40</sup>, that support just that. A more thorough analysis of prominent EMR standards can

---

<sup>40</sup> Sections talking about these formalisms are the "CIG execution - Patient management based on CIGs", the "Components of a CIG-based CDSS", and the "Information exchange" sections.

be found in [145], where standards are evaluated in four major areas, namely (1) in their level support for interoperability, (2) functionality, (3) extensibility and complementarity and (4) market relevance.

**Information exchange** in this case means that CIG-based CDSS need to communicate with other supporting CIS components (such as EMR and CPOE systems) in order to make decisions that are based on external information, and to propagate their results into other systems that want to consume it. Another critical requirement for these systems is the ability to **reuse knowledge**. This means that the effort needs to be minimized when facilitating knowledge in one system captured in another.

Standards could help with both of the aforementioned problems, but as there are no universally accepted CIG modeling languages today, other solutions are needed. According to [119,146], the two more easily sharable components are the data models and the expression languages. This means that sharing of knowledge can be achieved on multiple levels, using

1. **shared (medical) concept model** (i.e. vocabularies), to identify concepts of interest
2. **shared data models** (i.e. structures), to exchange data with concepts identified using vocabularies (e.g. patient information model)
3. **shared expression languages**, to communicate data-dependent condition expressions and calculation methods (as opposed to sharing the calculated data)

Each level builds on the next, thus being more complex.

### **Data models**

Standardizing the references to patient data in CIGs allows the prevention of the rewriting of the data references at the time of local adoption. The idea is that there needs to be an unambiguous one-time mapping between standard and local models, which then allows for an automated translation at the time of local adoption.

Candidates for data models include:

- vMR
- RIM

### Virtual Medical Record

The vMR specifications – maintained by HL7 – is described by [147] as “an interface that allows an information system to obtain access to, create, and or modify clinical statements about a given person. In its simplest form, the VMR could simply be viewed as an association between a person and that list of clinical statements.” It is intended to be a computationally independent model of a person’s EMR, which allows it to serve as a simplified and optimized input and output data structure for CDSS. vMR maps associated data elements from standards-based data structures to vMR data structures, producing vMR schemas and associated mappings.

### Reference Information Model

The RIM is an ANSI approved component of the HL7 Version 3 standards, which was developed with the aim to support all healthcare workflows. It provides a shared object-oriented model of the HL7 clinical data (domains) and defines the life cycle of messages exchanged in interactions within workflows. “RIM expresses the data content needed in a specific clinical or administrative context and provides an explicit representation of the semantic and lexical connections that exist between the information carried in the fields of HL7 messages.” [148]

### **Vocabularies**

Standard data models by themselves are not enough; they need to build on standard vocabularies to be truly effective [119]. This is achieved by data models using references to concepts defined by standard (medical) vocabularies. Concepts in standard vocabularies are generally organized in some sort of structure, most of the time in some sort of taxonomy, or in an ever more generic structure, an ontology. Examples include standards such as CPT<sup>41</sup>, ICD-9 and 10<sup>42</sup>, LOINC<sup>43</sup>, SNOMED-CT<sup>44</sup>, and so on.

Even if data models were built using local concepts, a one-time mapping is needed between local concepts and standard vocabularies in order for interoperability to be achieved.

---

<sup>41</sup> Current Procedural Terminology (CPT) [149]

<sup>42</sup> International Statistical Classification of Diseases and Related Health Problems version 9, 10 (ICD-9, ICD-10) [150]

<sup>43</sup> Logical Observation Identifiers Names and Codes (LOINC) [151]

<sup>44</sup> Systematized Nomenclature of Medicine - Clinical Terms (SNOMED CT) [152]

## Expression languages

Expression languages, such as ones introduced in previous sections (e.g. Arden Syntax's MLM query language and GLIF's recently standardized GELLO) can ease interoperability if standardized. With their help, a higher efficiency can be achieved when exchanging information.

Let us look at an example scenario, where one system needs the minimum and the maximum temperature values of a patient over the past two days, but temperature values with minute-based granularity are stored in another system. If there are no expressions used, the receiving system needs to get all data values from the other system for the past two days and find the minimum and maximum values itself. If there is a language allowing the construction of expressions in one system and consumed by another, the system receiving the request can calculate the minimum and maximum values and transmit them instead.

## Unified Medical Language System

Created in 1986, the *Unified Medical Language System (UMLS)* [153] is a collection of many controlled vocabularies in the biomedical sciences. UMLS is an important medical ontology that ties various other ontologies together. Its main components are:

- the **Metathesaurus**, which is the core database of the UMLS. It is a collection of concepts and terms from various controlled vocabularies, and their relationships,
- the **Semantic Network**, which is a set of categories and relationships that are being used to classify and relate the entries in the Metathesaurus,
- the **SPECIALIST Lexicon**, which is a database of lexicographic information to be used in natural language processing,
- a set of **supporting software tools**.

Knowledge sources in UMLS are multi-purpose by design, which means that they are not optimized for any one particular application, but instead can be applied in a large number of systems performing a wide range of functions. A set of supporting software tools were created in order to assist developers in using the UMLS knowledge sources for particular purposes.

The reason why an *ontology of ontologies*, like UMLS, is important is that it can serve as the common language to which concepts used in different CIGs could be translated to. This translator then can be used to translate requests between CIG-based CDSS and other servicing systems, as well as to transfer CIG specifications between two CIGs-based CDSS.

### **Health Level Seven**

Health Level Seven [154] is a non-profit organization involved in the development of international health care standards. The work of the HL7 group is important, because they oversee many of the standards, on which CIG-based CDSS need to build in order to better support interoperability. These include:

- conceptual standards (e.g. HL7 RIM),
- document standards (e.g. HL7 CDA),
- application standards (e.g. HL7 CCOW),
- and messaging standards (e.g. HL7 v2.x and v3.0).

“Messaging standards are particularly important because they define how information is packaged and communicated from one party to another. Such standards set the language, structure and data types required for seamless integration from one system to another” [155]. As mentioned earlier HL7 oversees the development of the Arden Syntax and GELLO as well.

## CHAPTER III.

### EVALUATION OF OPEN QUESTIONS IN CIG LANGUAGE DESIGN

While previous sections provided evaluations respective of each approach individually, this section provides an evaluation summary on formalisms and frameworks. The current evaluation can be considered as complementary to several comparisons, most of which are listed below. The section starts with a description of earlier evaluation efforts, which is then followed by a discussion including the author's own assessment and findings.

#### **Why is the evaluation of these approaches difficult?**

The section "Evaluation criteria for guideline-based clinical information systems" contains evaluation criteria in order to make the evaluation process easier. However, a proper assessment remains a difficult challenge, as the terminology used in these languages is inconsistent, the semantics of the control-flow of some of the languages is incompletely and informally defined, and the approaches used by the languages for guideline modeling are heterogeneous. [82]

#### **Earlier comparison efforts**

There have been multiple attempts to evaluate and compare CIG formalisms and their execution. In the following, some of the well-known attempts are listed in a chronological order. Major findings are also listed; however, it needs to be noted that many of these findings are already incorporated throughout this thesis.

In one of the earlier comparisons [45], Perry reviews representative knowledge bases and knowledge-based systems in medicine current in 1990 (including Physician Data Query (PDQ), Hepatitis Knowledge Base (HKB), MYCIN, CASNET, PIP, and INTERNIST-1). Most of these approaches are covered in the "Early CDSSs" section.

In an article from 1994 [156], authors showcase a performance analysis of the diagnostic capabilities of systems current at that time (Dxplain, Iliad, Meditel, and QMR). The results from the article indicate that the evaluated systems "should be used by physicians who can identify and use the relevant information and ignore the irrelevant information" provided these systems.

In an article from 2000 [157], authors provide a categorized review of guideline representation approaches, followed by a comparison of their features and capabilities. Their evaluation summary without further explanation can be seen in Table 7.

**Table 7 - Comparison of features of six CIG formalisms [157]**

Models	GLIF	EON	Asbru	PROforma	Prodigy	Prestige	GEM
Algorithmic	Yes	Yes	Yes	Not primarily	Yes	Not primarily	Not Primarily
Sub-guideline Support	Yes	Yes	Yes	Yes	Yes	Yes	No
Supports Decision Criteria	Yes	Yes	Yes (temporal)	Yes	Yes	Yes	Yes
Intentions and Goals Support	Possible through sub-guidelines	Yes	Yes	Yes	No	No	No
Ranking of Options Supported	Yes	Yes	No	Yes	Yes	Yes	Yes
Temporal Abstractions Supported	No	Yes	Yes	No	No	No	No
Explicitly Models Patient Preferences	No	No	No	No	No	No	No

In an article from 2003 [146], which can be viewed as a follow-up to [157], the authors studied “similarities and differences between CIGs in order to identify issues that must be resolved before a consensus on a set of common components can be developed”. Authors compared structural semantics of CIG languages using example models. Comparison was performed using eight major dimensions:

- **Organization of guideline plan components**, including plan components and capability for hierarchical decomposition (nesting)
- **Specification of goals/intentions** (using expression languages)
- **Model of guideline actions**, including the structuring and refinement of medical actions, expressivity of temporal constraints, and communication with host systems
- **Decision models**, including the expression of mutually exclusive branching of guideline control flow (i.e. switch), preferences for alternative candidates of non-deterministic decisions (i.e. choice alternatives), and authorizing user and system (i.e. manual and automatic) decisions
- **Expression/criterion languages** used to specify decision criteria, including presence criteria, template-based criteria, first-order logic criteria, temporal criteria, context-dependent expression, etc.



- **Data interpretations/abstractions**, including the expression of temporal abstractions and patterns (i.e. trends), abstract terms (derived other terms), terminology abstractions via classification hierarchies
- **Representation of a medical concept model and its use** describes how languages reference and access medical concepts (e.g. referencing variable names, explicit function calls, or using standardized terminology concepts)
- **Patient information model** is concerned with representing patient data and its mapping to institutional EMR data models

Even though the authors found consensus in many aspects (including plan organization, expression language, conceptual medical record model, medical concept model, and data abstractions), results indicated that a common CIG language was far from feasible as there were many fundamental differences among the languages evaluated (including in underlying decision models, goal representation, use of scenarios, and structured medical actions).

A 2007 article [78] surveys free and open source (FOS) technologies for patient-centric, guideline-based care, and discusses trends and future directions of their role in clinical decision support. The article provides a high-level overview of the availability and functionalities of then existing and emerging FOS guideline modeling tools as well as FOS supporting technologies for implementing terminology, data interchange, and EMR standards. Authors reported active and growing trends of deploying FOS enabling technologies and suggested some possible future directions. While a variety of sophisticated, formal CPG representations and languages are available free, concerns are raised as “most of the modeling and editing tools are proprietary, and have limited acceptance and usage in practice due to the complexity of the modeling tasks”.

Another 2007 article [82], which can be considered as complementary to previous comparisons [146,158–161], examines and compares the expressive power of CIG modeling languages using pattern-based analysis. Authors compared the control-flow component<sup>45</sup> of special-purpose CIG languages by evaluating their degree of support for expressing a predefined set<sup>46</sup> of *control-flow patterns*<sup>47</sup> used

---

<sup>45</sup> Also referred to as *decision models* [146].

<sup>46</sup> A comprehensive description of the full set of control-flow patterns, 43 in total, can be found in [162].

normally to evaluate expressiveness of general workflows. The evaluation of support for a common set of operators not only allows putting one CIG language in contrast with another, but it allows one to evaluate the differences – from the control-flow perspective – between general process modeling languages offered by workflow management systems and modeling languages used to design clinical guidelines. Some of the more interesting findings of the article are summarized below:

- Even though the members of the computerized guidelines community have emphasized how important it is to support flexibility in guideline formalisms, the examined guideline modeling languages (Asbru, EON, GLIF3.5, and PROforma) showed only limited additional flexibility not present in business process modeling languages. Only two new patterns<sup>48</sup> were not encountered in business process models.
- Additionally, only half of the workflow patterns elicited from business process modeling languages are supported by CIG languages. Many of the missing patterns relate to flexibility of process execution, which may be useful for clinical guideline applications.
- Because CIG languages do not offer substantially more control-flow constructs than business process modeling languages, it is suggested for the medical community to “rethink the use of more general formalisms and tools, which have formal foundation and have been widely tested and used in industry, for expressing control flow of guideline models”.
- In addition to the set of constructs discussed in this article, to support the interoperability of CIGs the use of configurable modeling constructs – found in business process formalisms – is proposed.

In an article from 2008 [26], authors review and compare eight systems that allow some kind of enactment of clinical guidelines using an EE. Several aspects related to execution are inspected. These include CIG semantics, CIG management, and interfacing of host CIS and EE. The article identifies a common feature related to the internal structure of the analyzed systems, according to which most of

---

<sup>47</sup> Also known as *workflow patterns*.

<sup>48</sup> The two patterns are *deferred multi-choice* and *forced trigger*. *Deferred multi-choice* is “a capability to defer the selection of multiple options by a user until the user decides that no more options will be selected (for instance, selecting several medicines from the recommended ones for the treatment of the patient). The functionality of the deferred multi-choice has been encountered in GLIF3.5/Protege-2000, EON/Protege-2000 and PROforma/Tallis”. *Forced trigger* is present in scenarios “where any internal or external event triggers the execution of a task even if the task precondition was not satisfied at the moment of triggering. The functionality of the forced trigger has been encountered in PROforma/Tallis”. [82]

the evaluated systems have three main layers. These are (1) a layer with the patient's related data, (2) an intermediate layer with the execution engine, and (3) a layer containing a set of interfaces to connect the execution engine with external systems. Another, rather disappointing, result of the evaluation is that, as far as we know, none of the systems is actually in daily use in any medical center.

A 2009 article [32] presents SpEM, a generic framework for CIG management together with a case study. The SpEM approach is based on the ECA rule paradigm and active databases. In order to display the novel capabilities of the framework a comparison of existing formalisms is provided, with special attention paid to the support for the manipulation (including "performing operations and issuing queries on aspects of the guideline knowledge and information") and "the exploitation of database features for managing information and knowledge".

The last article mentioned here is from 2009 [163]. It is not a comparative study in the classical sense, but an important extension to the work presented in [82]. The work showcased is on the formal analysis of the expressiveness and on the verification of structural, behavioral and temporal properties of clinical workflows. With the help of Colored Petri nets (CPNs) the authors show how the chosen guideline language, PROforma, may be mapped to a formal (CPN) representation, which allow them to provide formal proofs on which workflow patterns (out of the 43) are supported by PROforma. It is argued (and with an example it is demonstrated) that [163] provides a more rigorous analysis with different results than the ad-hoc method presented in [82].

Table 8 - CIG formalisms compared in published studies

Articles	[157]	[146]	[78]	[82]	[26]	[32]	[163]
Arden Syntax [79–81]	Discussed		Discussed		Compared		
PRODIGY [164,165]	Compared	Compared				Compared	
Prestige	Compared					Compared	
PROforma [97–99]	Compared	Compared			Compared (Arezzo, HeCaSe2)	Compared	Analyzed
Asbru [31,33]	Compared	Compared	Discussed	Compared	Compared (DeGeL)	Compared	
EON [166,167]	Compared	Compared	Discussed	Compared		Compared	
GLIF [106,107]	Compared	Compared	Discussed	Compared	Compared (GLEE)	Compared	
GEODE-CM	Discussed						
GEM [168,169]	Compared		Discussed				
GUIDE	Compared				Compared		
HELEN [170]	Discussed						
SAGE [171,172]	Discussed			Compared			
Stepper	Discussed						
GLARE [173,174]					Compared	Compared	
NewGuide						Compared	
SpEM [175]					Compared	Compared	
GASTON [176,177]						Compared	
SIEGFRIED						Compared	

### Discussion

This section aims to provide a discussion regarding commonalities and differences found in evaluated approaches. This discussion covers the two main components of frameworks, namely the CIG formalism and the software suite.

## Discussion on CIG formalisms

### DSML

All examined languages are custom built DSMLs<sup>49</sup>, specifically created to capture CIGs. Their syntax and semantics are formally defined, but with various levels of degree. Nonetheless, some abstract form of execution model is provided for all of them. Interestingly, though all languages recognize the importance of interoperability (e.g. GLIF even was supposed to act as an interchange format), we have not come across any form of formal mapping that would allow automated translation between the examined CIG formalisms.

### Components

The languages differ slightly as to what they consider the elements of a CIG to be, and how its objective is expressed. However, all allow CIGs to contain, among other things, decisions, actions, and nesting (i.e. hierarchical decomposition). Furthermore, all contain expression languages that represent criteria, which influence decisions and control plan execution (e.g. to express conditions that determine whether a task might be started or terminated) [36]. In summary, the types of information typically stored in CIG models include the following independent components:

- **descriptive data**, which contains a (machine-readable) unique identifier for the CIG and various other metadata such as author, source, version, and purpose. It also may state whether the CIG is a specialization of another<sup>50</sup>.
- **plan, or control logic**, which is the core of the CIG. It defines the permitted or mandated set of (clinical) actions and situations. Restrictions on structure and content of the CIG instances are specified as a combination of an orchestrated set of allowed actions and constraint rules. Expression languages are considered to be an integral part of the control logic and they are generally applied in combination with the rest of the control constructs (i.e. as an extension).

---

<sup>49</sup> Even though they use concepts from medicine, medical professionals still need to invest both time and energy to learn them. Evaluating whether one language is more intuitive than the other is not in the scope of this thesis.

<sup>50</sup> This can be achieved with various mechanisms, including using traditional versioning techniques, or inheritance operators of an ontology.

They are used to express parts of the plan and thus they achieve a certain subset of the functionality.

- **concepts with semantic ties**, which are the basic (atomic) building blocks of the CIG (e.g. medications). Definitions for the concepts are given with the help of ontologies provided by the language or the CIG, or with the help of semantic ties to controlled (i.e. standardized) vocabularies.

### Terminology and information exchange

Inputs (such as data points, which an executing CIG instance bases its decisions upon) and outputs (produced by the CIG-based system) must be communicated through an interface. Unless manual information exchange<sup>51</sup> is an acceptable model, this translates into a communication using some form of shared terminology. Accordingly, all, but one formalism rely on some sort of medical terminology both to represent (atomic) concepts and to help define an interface to the host CISs. The only exception is PROforma.

Other important aspects of information exchange, such as the ones described in the “Information exchange” subsection of “Information technology aspect: System integration” (e.g. specifying the mechanism of access, addressing security, managing availability, and specifying quality) are generally omitted in the explanation provided by all examined formalism. Without these specifics, a CIG can only be implemented on a relatively abstract level. This issue is further discussed in P2.2.

### Expression language

To manage complexity, each language facilitates a custom expression language of its own. The definition for the expression language is provided in terms of syntax and semantics, the latter of which provides a definition for calculating the value of expressions, ideally without causing any *side effects*<sup>52</sup> [42]. An expression language may enable the implementation of information exchange (e.g. data querying in Arden Syntax), and methods for constructing derived data points with filtering and aggregation using

---

<sup>51</sup> The manual information exchange model simply means that the system is used as a standalone tool. Accordingly, providers manually need to enter input data in order to receive recommendations, which they need to process.

<sup>52</sup> Being *side effect free* here means that using (the operators of) the expression language causes no changes to the state of the guideline.

data values and logical operators defined by the (expression) language. They can also express various constraints (including conditions for goals, failure, an priorities over alternatives, etc.). For example, other than Arden Syntax's explicit placeholder, called "*purpose*", the goals of plans are implicitly defined together with the rest of the language<sup>53</sup>.

### Describing the control logic

All four languages were designed to describe CIGs, but we found that there are two distinct styles in terms of representing control logic, namely (1) the rule-based approach, which builds on ECA rules, and (2) the workflow-based approach, which uses *task network models* (TNM)<sup>54</sup>. While the Arden Syntax<sup>55</sup> uses the former, most of the other languages, including the rest of the formalisms examined in this thesis: PROforma, GLIF, and Asbru, use the latter.

#### Rule-based

As described in earlier sections, *rule-based systems* rely on storing and executing ECA rules. These rules implicitly define a control flow. There are many different rule-based system implementations, but their execution logic follows a logic similar to the two methods described in the following. (1) In execution time for systems that implement *forward chaining*, all (enabled) rules are considered *active* (i.e. runnable). The execution environment is in charge of listening for events (both internal and ones coming from the environment). If for a rule, its triggering event(s) is (are) observed, its conditions (associated with the triggering event) will be evaluated. If the conditions are satisfied, the actions defined in the rule are performed. As a result of the triggered actions, newly enabled rules are activated. These systems need a strategy for managing the situation where multiple (potentially conflicting) rules get selected for execution. Strategies for conflict resolution can involve preset priorities for rules or user-based resolution methods. (2) The opposite of forward chaining is *backward chaining*, where the system is working iteratively backward from a list of predefined goals by trying to search the applicable set of rules in order to find ones that would result in one of the goals.

---

<sup>53</sup> This is especially true for Asbru's natural language-based "intentions" implemented in the "logic" block.

<sup>54</sup> Definition for TNM is provided below.

<sup>55</sup> PRODIGY, which is not examined here, also uses the rule-based approach [146].

### Workflow-based

The other type of approach in representing the execution logic in CIGs is using *workflows*. In the literature, there are many names and many definitions, but essentially, they all mean the same thing. The main building block of this approach is often referred to as a *plan* [36,146], as in "an orderly arrangement of parts of an overall design or objective"<sup>56</sup>. However, we refrain from using this term, as this would incorrectly imply that a collection of rules in a rule-based system would not be able to describe a medical plan. The term "*task network models*" is also used to describe guideline-modeling formats based on this approach. According to the definition, TNM languages "provide modeling primitives specifically designed for representing complex, multistep clinical guidelines and for describing temporal and other relationships between component tasks" [146].

Based on definitions provided by [36,82,146,157], *workflow-based approaches* can be viewed as a paradigm, where multi-step guidelines are modeled as coordinated sets of interacting tasks in an explicit control flow. A task in the CIG can be considered as a logical unit of work that is carried out as a whole. They represent medical actions such as decisions and procedures. Coordinated by the control flow, tasks are carried out in sequence or in parallel over a period of time using typical (i.e. basic) control flow patterns, including sequencing, parallel split, synchronization, merge, and exclusive choice. Components can be composed into iterative, cyclic and hierarchical structures. In addition, models support expression of temporal (i.e. scheduling) constraints as well as various other logical preconditions on their components.

Albeit building on the same approach, workflow-based languages differ in terms of expressivity. These differences are due to variance in (1) supported component (both data and action) types, in (2) the expressivity of the expression language, and (3) the support for various control flow operators (i.e. patterns). As described in the previous section, comparison of (3) is what the authors of [82] proposed, using workflow patterns defined by the business process-modeling community. While all basic patterns are directly supported by the languages examined in [82], there is a greater variance in their support for more advanced ones (e.g. multiple choice and multiple merge operators).

The typical and natural visual representation associated with this approach is a flow diagram. The flow diagram is a directed graph where a node can either be a starting point, one of the tasks, or one of the

---

<sup>56</sup> Definition of a "plan" according to the Merriam-Webster dictionary.



ending points. In the graph, the arrows between the tasks represent scheduling constraints. Similarly to UML's activity diagrams, they prevent one task from being activated until another has finished. Scheduling constraints, which are easy to visualize, express necessary, but not sufficient conditions for the activation of tasks. However, each task has a set of logical preconditions that are evaluated after its scheduling constraints are satisfied, and the task is only activated if both its scheduling constraints and its preconditions are satisfied [36].

### **Discussion on software suites**

Architectural requirements and a common framework have already been described in previous sections. Figure 3 in "Components of a CIG-based CDSS" summarizes what are the essential components and how they should interact. With various levels of detail, all examined formalisms implement the needed components, including a CIG modeling environment, an execution engine, a communication layer, a test environment, and finally a patient management UI.

Unfortunately, according to [26] the discussed systems are only used as prototypes and are not part of actual live CIS systems.<sup>57</sup>

### **Open problems**

This section discusses problems not fully addressed by any of the introduced frameworks. Provided by various authors, partial solutions – if found – are given for the listed problems.

#### **P1. Execution semantics and expressivity**

##### ***P1.1. Seeking a common representation for all the different types of CIGs***

In the section "Selecting a suitable modeling language" the question of whether it is possible or not to define a single DSML to represent all guidelines was raised. The answer of the authors of [70] to this question is *not currently*. To explain why, they list five major obstacles in their article:

1. **As of now, there is no consensus in the CIG community on the most effective applications for computer-based guidelines.** Consequently, it is difficult to identify functional requirements, and

---

<sup>57</sup> Arden and GLIF are claimed to be (see Arden/Use and GLIF/Use).

thus what conceptual models and development environments are most likely to be important. This is reflected by the variety of modeling approaches in existence.

2. **The guideline authoring process is influenced by the specification of location and use case-specific functional requirements.** On one hand, guidelines can be developed generically, which leaves their adaptation to the implementers of specific applications. On the other hand, they can be developed specifically, which hinders reuse. In the latter case, to support reuse, authoring tools need to better provide some sort of separation of local and generic components.
3. **Dissemination of CIG models needs to be supported by both the modeling framework and the anticipated delivery environment (i.e. platform).** If there is a one-to-one mapping between the models created and the ones consumed, that is the implementation is within a closed environment, then a (proprietary) dissemination format can be agreed-upon straightforwardly. If the mapping is many-to-many, meaning that the guidelines are developed generically for multiple purposes, or for multiple target environments, then a common (standardized) format needs to be adopted.
4. **Despite their interest in providing CIG-based support, implementers of clinical information systems are more interested in adopting already existing systems and methods than investing significant resources in development and/or evaluation such systems.** Accordingly, current approaches would need to provide readily available tools for the incorporation of guideline-based applications, including for “customization of the guideline logic, adapting to workflow requirements, mapping of data elements and actions, and integrating user interfaces” [70].
5. **Update of adapted CIGs is an unsolved problem.** More on this can be found in a dedicated problem point (see P2.9).

*P1.2. Workflow versus rule-based approaches*

As listed in the “Clinical Practice Guidelines” section, CIGs have a broad range of uses. However, in summary they provide the means for two things: (1) identifying clinical situations and (2) managing clinical processes associated with predefined clinical situations. For CIGs that put the emphasis on (1), and implement monitoring of many potential clinical conditions and provide alerts/reminders based on identified conditions, the critical functionality involves identifying methods for defining conditions, automating the exchange and processing of data involved, and generating, filtering, and propagating events. Whereas for CIGs that concentrate on (2), the emphasis needs to be on being able to express,

interpret, and navigate through a potentially complex graph representing dependency and causality among observations and actions.

For example, an alert on an identified potential diagnosis and a list of associated recommended actions would be an expected message generated by a CIG of (1). As for systematic instructions, including orchestration based on the order of how things should be performed, or help with resource allocation (e.g. who is supposed to what, is the appropriate equipment available) and scheduling (i.e. when is an action need to be performed) would be expected from a CIG of (2).

It is arguable, but generally speaking (1) maps well to the rule-based paradigm, whereas (2) to the workflow-based paradigm. The advantages and limitations of the two approaches, detailed below, should help explain why. The benefit of using a workflow-based approach is that, unlike rule-based systems, they represent the control flow explicitly. This means that they can explicitly model alternative pathways (i.e. sequences of tasks) as a control flow and they provide tools for visual representation of plans and the organization of tasks within them, thus they can be much easier to define and comprehend. However, their use poses significant disadvantages. As they are designed to describe the flow of execution in the form of a sequence of operations, for all non-orthogonal (i.e. non-parallel) tasks there **always needs to be a complete order defined**. If permitted these tasks should be split to parallel branches, otherwise operators related to ordering, such as “execute-in-any-order”, should be introduced. However, this solution is far less scalable than using decision tables for example. While experimenting with workflow-based CIG languages, we also found them to be **difficult to use for describing interrupts** (i.e. events signaling the state changes of the environment in an even-driven system). A typical group of examples, which illustrates this problem, includes cases where a sequence of tasks is defined based on the priorities of the problems they would address. In these cases, while the ordering can faithfully represent how physicians address problems, the execution flow is too rigid in the sense that it would not go back to treat a high priority problem once it has already advanced to addressing lower priority one. Potential alterations include adding (many supplemental) arrows to cover all possible cases (i.e. all possible event combinations), or splitting up the original workflow to independent workflows, both of which made our model much less comprehensible.

Rule-based representations can seemingly overcome this problem. In a rule-based world, all constraints from the previously mentioned examples are represented as part of ECA rules. However, a large set of (seemingly disjoint, but implicitly dependent) rules cause comprehension and maintenance problems. Verification is also difficult, as rules are relatively low level and they are captured and evaluated

independently of each other. Yet another problem is the unhandled non-determinism among evaluated and concurrently enabled (forward chaining) rules.

While the authors of [157] state that “CPGs are typically composed of sequences of steps that have been arduously designed as a whole, instead of strings of single-step rules that are secondarily chained together” it does not mean that they can easily be represented by workflows. Furthermore, it needs to be noted that formalisms, which use rule-based specification (e.g. Asbru) can also be adapted to represent guideline knowledge that unfold over time.

**We believe that an ideal solution needs to support both paradigms.** This would allow for expressing the general sequence of logic in terms of a branching workflow, but also allow exceptions to be phrased as a set of rules to alter the behavior defined by the workflow. In this case, exceptions define cases either not represented by the workflow or cases that are, but need to be overridden (by the exceptions).

### *P1.3. Lack of formality and explicitly defined execution semantics*

Defining explicit formal execution semantics is important for all CIG languages. Reasons can be found in the previous sections, including “Benefits of Computer Interpretable Guidelines” and “Formal semantics”. These reasons can be summarized as follows:

- Formal semantics help clarify what CIGs – defined with the help of the language – are trying to achieve.
- It allows the implementation of software tools. More specifically, it provides a specification for expected behavior, both in simulation and in real-life execution environments.
- It also provides means for analysis, including the comparison of various formalisms through the assessment of what representations can and cannot capture.
- The use of a formal, well-defined representation also promotes maintainability and reusability of the software, as the temporal structure and coordination of the tasks will be captured explicitly. This is in sharp contrast to traditional approaches where this information is hidden in the code.

All examined approaches have formalized semantics; however, each of them only provides a high-level, abstract execution model, which is not sufficiently detailed for building an execution engine that could

be integrated into a live CIS. Part of the problem is that no formalism is able to properly (i.e. separately, formally, and explicitly) represent and incorporate the behavior of the host CIS.

*P1.4. Avoiding duplication of functionality between the CIG and the expression language*

The expression languages in all examined cases are part of the language. However, based on their description (provided in the “Expression language” section), it can be seen that expression languages can overlap with other parts of the CIG language in terms of functionality. For example, if conditions can refer to time, a condition (of the expression language) can potentially describe a situation that contradicts the sequencing phrased by a sequencing operator (of CIG language). Alternatively, if a condition can refer to the occurrence of an event<sup>58</sup> it can express sequencing itself.

*P1.5. Temporal reasoning*

Temporal reasoning, specifically the representation and evaluation of temporal expressions is an essential part of CIGs. In CIGs, temporal expressions typically are used to express constraints on possible execution sequences, or assertions that need to be validated during execution. For their implementation, three important questions need to be evaluated:

1. **What is the expressivity of the temporal expression language?** In other words, what kind of temporal concepts are being modeled and how can they be used?
2. **What are the questions expected to be answered based on the expressed constraints?** In other words, are the limitations caused by the complexity of satisfiability problem in the chosen logic acceptable? Is, for example, simplification of the phrased constraints, or checking for contradictions expected?
3. **What is the expected evaluation strategy for the constraints?** Is it pre-processed (i.e. design-time scheduling), or lazy (i.e. evaluated at runtime)? Is it cyclical (i.e. checked with a certain frequency), or on-demand (i.e. event-based)?

Temporal constraints in CIGs often express both *qualitative temporal requirements* (e.g. invariance, precedence) and *quantitative temporal requirements* (i.e. “hard real-time constraints, which put timing

---

<sup>58</sup> An expression language can employ time-related expressions similar to what Allen’s interval algebra [178] defines.

deadlines on the behavior of the system”)<sup>59</sup>. Examples of the latter include constraints that refer to *absolute time*, (i.e. metric time, e.g. drug A has to be administered before 10 am), and *relative time*, meaning that there is an event ordering annotated with timing expressed relative to an event or state change of the protocol (e.g. drug A has to be administered no later than 3 hours after drug B was).

In summary, the logic should be expressive enough to formalize a wide class of time-related constraints, but checking compliance – satisfiability, in logical terms – should be decidable and efficient enough for practical use [180]. However, this topic is not discussed by any of the examined approaches.

#### *P1.6. Component-based guideline modeling*

The use of CIG formalisms promises better reuse of CIG components, thus greater flexibility, scalability and higher quality when constructing new guidelines. In this regard, CIG development shares many common attributes with component-based software development [181]. For this reason, it is important to leverage existing accepted solutions introduced in component-based software development where it is possible. The following subsections discuss some of the necessary features of component-based CIG development.

##### *Composition of protocols*

The composition of protocols is another in the list of topics, which has been neglected in the literature. Composition here means the combination of two or more CIGs in order to produce a new one, with the assumption that the new CIG will combine the logic of its component CIGs without any potential interference between components. *Interference* can be interpreted as unwanted interaction between two component protocols (e.g. contradictions, or contraindications). The benefit of protocol composition is that it supports model reuse. However, its implementation is non-trivial, because interference handling, and the automatic update of the used components are tasks needed to be worked out.

The idea of composition was brought up in [80], because composition in Arden Syntax’s rule-based paradigm is a natural concept. The hierarchical nesting of MLMs allows the reuse of components of existing MLMs, however, Arden’s building blocks (i.e. subcomponents) have a fixed predefined context

---

<sup>59</sup> For example, qualitative temporal requirements can be expressed with Allen’s interval algebra [178], while quantitative ones can be with MFOTL [179].

(i.e. a set of criteria describing when they are applicable). The disadvantage of using this type of protocol composition is that it does not allow nested sub-protocols to be modified by their parents.

We believe that there are three types of sub-protocol inclusions (other than a plain copy of one protocol's elements to another, where no conceptual link is preserved between the two copies):

- Protocols viewed as being **atomic (or black box) systems**, in which case the included (sub)protocol is not modifiable (similar to the concept of libraries in software engineering, or to the concept of shortcuts used in many operating systems). Here the composition is rigid, but straightforward (e.g. Arden Syntax).
- Protocols provide an **interface for manipulation**, in which case protocols provide an interface for any potential host to manipulate its internal structure and/or behavior. Examples for this would be to allow the host through predefined methods to suppress or override alerts, or set parameters of the sub-protocol (e.g. redefine the temperature threshold for fever). This allows a component identifying a patient with high blood pressure to have different triggering thresholds in two different disease treatment cases.
- Protocols are **copied with preserving a conceptual link to source**, in which case the contents of the (source/sub)protocol is copied into the host protocol while preserving a conceptual link to the source for managing updates of the copy based on changes made to its source. This approach allows the complete override of each individual element (of the copy), however automatic update management is only possible if domain modelers understand the chosen conflict resolution methods (e.g. source overrides changes in the copy, updates only happen if there were no changes made to the copy, etc.).

In the latter two cases, support for handling protocol-protocol interaction needs to be provided. This will allow the analysis of the effect and the resolution of potential conflicts of multiple guideline instances that are being executed on one patient (e.g. finding contradicting suggestions).

#### *Decomposition of protocols*

The opposite of the previous point, decomposition of complex protocols into independently (re)usable subprotocols is also a challenging problem. Deciding where it makes sense to split a protocol up into subprotocols is highly dependent of the protocol's use and can only be decided by domain experts.

However, if the composition of protocols is supported, creating methods for dividing protocols should be targeted to allow maximizing the reuse of captured knowledge.

### *Version control*

Problems described in the previous two points are both related to versioning. Versioning in the case of CIGs deals with tracking the evolution of a protocol. It entails creating and maintaining an accessible version history through committed user changes.

In component-based software development there are multiple versioning schemes provided [182–184], but there is no consensus in which proposed method would be the best method. In order to decide which one of the many proposed solutions (if any) fit the CIG development, one needs to evaluate the types of changes that would trigger a version change in a CIG's lifecycle. Some of the most important aspects of CIG versioning that influence this decision are listed below:

- Type of the change
  - Continuous development: Development of a complex guideline takes a lot of time, during which multiple iterative version might accumulate.
  - Correction of a discovered error: Like any other piece of software, CIGs are hardly perfect, when being created. Analysis, testing or everyday use can reveal these errors, which then need to be addressed.
  - Creation of new protocols by composition or decomposition: These methods were previously discussed. As an example, depending on the chosen method, their behavior can be similar to the use of software libraries or to complete source fork<sup>60</sup>.
  - Parallel development: If multiple users need to work on a set of CIGs at the same time the creation of parallel branches and their merge need to be supported.
- Degree of the change: Projects and local workflows can require the differentiation between minor changes, which do not affect end users, and major changes, which do (e.g. official releases).

---

<sup>60</sup> The process of *forking* in software engineering produces a distinct piece of software by create a complete copy of the original source code of software package so that development on it can be done independent of the original package.



- Functional and non-functional versions: In a collaborative environment even non-functional (i.e. non-executable) versions of CIG have value, as they could be passed around between various domain experts, each of whom can make changes in order to complete the guideline.
- Local adaptation<sup>61</sup>
  - HCO-specific version
  - Personalized version: Some providers could require the customization of a CIG (e.g. adjusting certain thresholds).

As opposed to versioning of CIGs, versioning can be interpreted in a broader sense as well. In a larger project, basically all captured CIG components that can be considered as standalone, reusable components should be version tracked.

#### *P1.7. Representing uncertainty*

Uncertainty is inherent to medicine, as medical processes often exert complex behavior in which outcomes are non-linear. Because of this, the representation of uncertainty is crucial in CIGs. Despite this requirement, and the fact that various techniques have been proposed and evaluated before [185,186], none of the examined approaches provide native support for them (e.g. probabilistic decision models, fuzzy logic).

We found that there are two (somewhat overlapping) use cases where CIG representations could benefit from representing uncertainty: (1) selection of the proper solution from alternatives and (2) representing numerical parameters. These topics are discussed in the following subsections.

##### *Selection of the proper solution from alternatives*

Expressing uncertainty is important when offering competing alternative treatment options for a patient with a particular configuration of clinical indicators. This requisite is effectively illustrated by the example presented by Figure 22.

---

<sup>61</sup> This topic is discussed in more detail in the following sections.

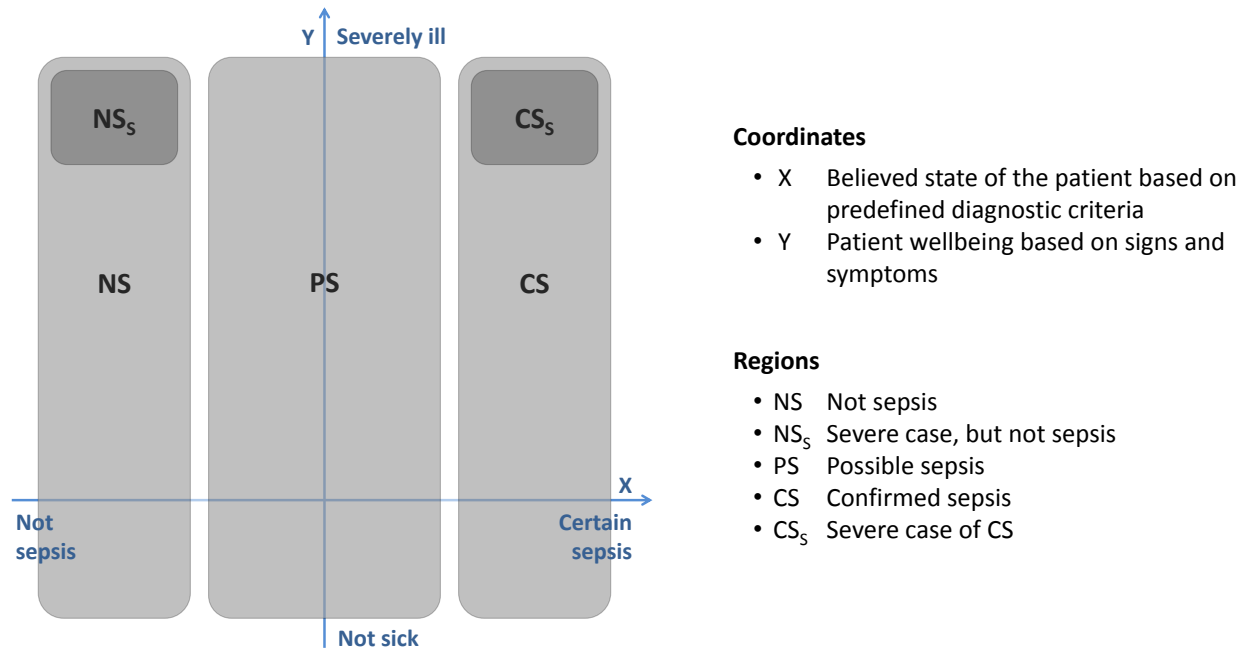


Figure 22 - Deciding on protocol applicability (Directed versus empiric therapy)

Figure 22 shows a two axis Cartesian coordinate system, which should help clinicians decide what treatments to choose given that there is some suspicion that a patient could have sepsis (e.g. after a notification from an automatic *health indicator*<sup>62</sup> monitoring system). The X-axis represents the confidence level of a provider in terms of sepsis based on the available *objective data*<sup>63</sup> of the patient. The Y-axis shows the general wellbeing of the patient in terms of the same objective data. The explanation of the grey regions of the figure is the following:

- Not sepsis (NS): here clinicians should decline starting up the sepsis protocol.
- Severe case of NS (NS<sub>s</sub>): this special case of NS requires some intervention (other than the sepsis protocol), because the patient is very sick.

<sup>62</sup> *Health indicator* is a patient related data point that is indicative of their health status (e.g. signs and symptoms).

<sup>63</sup> *Objective data* in the health-care setting “can be defined as data that is factual, unbiased, and unchanged by personal feelings or interpretations”, examples include lab test values, sensory data, medication history. [187]

- Possible sepsis (PS): here *empiric therapy* is advised, according to which clinicians should treat for all potential diagnoses (including sepsis, as well as others), but only do procedures that are not conflicting (i.e. do not contradict each other) and are not harmful to the patient (i.e. do not perform an action which would have a highly negative effect on the patient in case the assumed diagnosis was not right).
- Confirmed sepsis (CS): here *directed therapy* is advised, which means devising specific treatments for specific diseases [188]. This, in the case of sepsis, translates to the execution of a CIG-defined sepsis protocol.
- Severe case of CS (CS<sub>S</sub>): in this special case of CS, the CIG should enable otherwise sequential steps in parallel in order to compensate for the severity of the case (e.g. execution of the fluid challenge and the early goal-directed therapy, both of which are components of the sepsis treatment CIG).

As the above example illustrates, expressing uncertainty in selection of the proper solution from alternatives is vital. We found that methods for ranking the alternatives can be one of the following options:

- represented **implicitly**, meaning that the order is built into the solution (e.g. try solution A first, afterwards evaluate parameter P, if  $P < \text{threshold } T$ , try solution B),
- represented **explicitly**, with a help of relative or absolute priorities, or with a help of scoring values, which attach probabilities to alternatives, where values can be based on statistical results, or
- it can be **left to the end-user**, which is a form of (intended) non-determinism that can (and should) always be resolved by the expert users. This choice, the existence of the need to make a decision over alternative solutions, indicates a missing (i.e. incomplete) criterion. The missing piece of information can be absent due to various reasons.
  - First, it could be that the criterion cannot be represented, because (a) the (expression) language is not sufficiently expressive, or because (b) the information cannot be described or obtained automatically (such as the “septic look” of a patient, a typical condition identified by physicians when physically examining a patient).

- Second, it could be a deliberate decision for requiring user decision (for example if there is no study supporting decision, or there is too much risk involved).

### Representing numerical parameters

In all evaluated systems, numerical parameters (e.g. thresholds of a condition for triggering alerts) are discretized values, even though the fact that the definition of these values are somewhat arbitrary. The case of hypotension detection demonstrates this issue well. Hypotension is generally defined as systolic blood pressure less than 90 mm Hg<sup>64</sup> or diastolic less than 60 mm Hg. Does this mean that it is acceptable to not bother looking at patients who have a diastolic blood pressure of 61 mm Hg (and systolic over 91)? A solution building on fuzzy logic might be able to address this issue (see Figure 23), but of course, it would have its own disadvantages, including the fact that there is usually no research data to fill in the gap (between discrete values). For example, the effects of 250mg and 500mg doses of Fulvestrant are well studied, but there are no studies done for a 355mg dose.

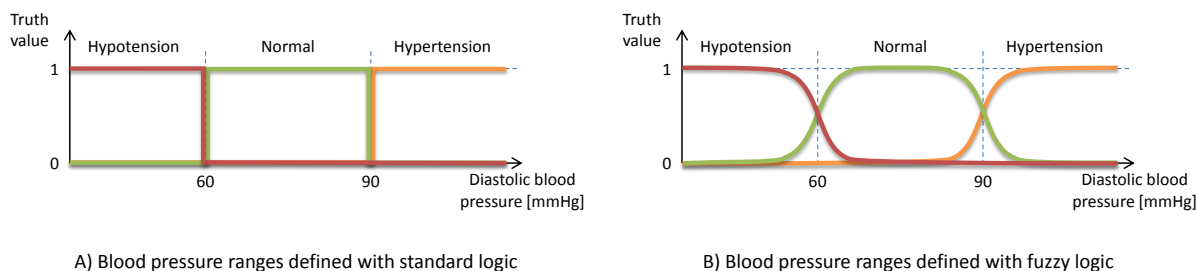


Figure 23 - Standard versus fuzzy logic for representing uncertainty

In [189], the author proposes extensions to Asbru and PROforma to include the ability to capture various “grading systems”, which would allow for better automation of the selection among alternatives. These grading systems base their decision on the following groups of information:

- Levels of Evidence (LoEs)
- Strengths of Recommendations (SoRs)
- Trade-off between benefits and harms
- Cost of actions

<sup>64</sup> millimeters of mercury

*P1.8. Representing configurable input parameters*

In an ideal world, clinical conditions either would be independent (i.e. orthogonal) problems, or would be decomposable to independent problems. If this was true, guidelines could be defined to address each problem independently. However, clinical conditions are often highly dependent on each other. Unfortunately, not all dependencies are fully understood, only suspected. Some of them might not be discovered at all. The ones that are well-understood can be divided into two groups: explicit (e.g. chronic hypertension may lead to hypertensive retinopathy) and implicit dependencies (e.g. treatments of both problem  $P_1$  and  $P_2$  involves X dose of medication M, however if both  $P_1$  and  $P_2$  are present it does not mean that 2X of M should be given for a patient). These kinds of dependencies cause a state space explosion in the space of unique problem combinations for which dedicated treatments needs to be defined. In the worst case, CIGs would need to be defined for each element of the superset of problems, which would be extremely difficult to manage. Thus, it makes sense to provide clinicians with flexible CIGs, which they can tailor to the specific needs of the patient. Providing a solution specific to the patient requires adjustable thresholds and alternative treatment options. Examples for configurable parameters can be seen in Figure 24. These parameters represent intended non-determinism that is resolved at either before or in runtime.

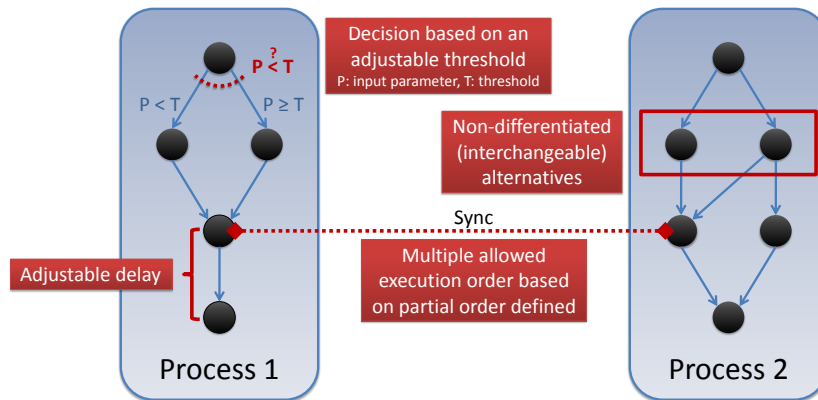


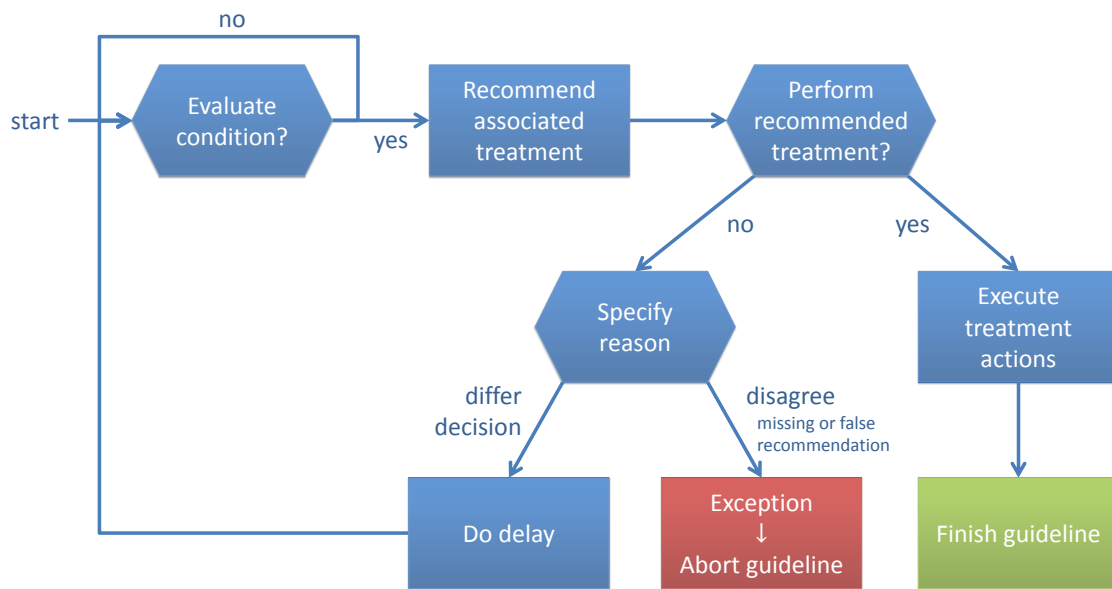
Figure 24 - Forms of non-determinism in CIGs

Nonetheless, to track protocol compliance it is important to cleanly identify which parts of guidelines are configurable and which are not<sup>65</sup>.

<sup>65</sup> Clinicians may always choose to opt out from a CIG if they do not find any of the predefined options appropriate.

**P1.9. Exception handling and state-space coverage problems**

During the execution of a guideline, the actual treatment in real life implemented by a clinician might diverge from the scenario-based gold standard, which is captured by the executing CIG (Figure 25). This dissonance indicates (1) potential problems in the encoding of decision logic and/or (2) the quality of (patient) data built upon<sup>66</sup>. In cases when only the former is true (i.e. (1)), there are three possible scenarios: either (a) there is an inaccuracy in the CIG, which needs to be corrected, (b) the wrong CIG was selected for execution, or (c) the state-space is not sufficiently covered, which means that the particular (sub)case simply was not captured. All cases of (1) should be considered as exceptions, and as such, have to be gracefully handled by the execution environment. Examples for a couple of graceful methods are discussed in the following three sections. Cases involving (2) indicate issues with data acquisition, which is not in the scope of this work, however, the related topic of rollback is discussed in P1.12.



**Figure 25 - Simplified decision support flowchart**

**P1.10. Grey tracking**

In the context of CIGs, *grey tracking* is the ability of an execution environment to follow the actions of a user even if they do not precisely line up with what the guideline suggests. Furthermore, a grey tracking enabled EE should be able to recognize points in the (off-course) treatment trajectory where the

<sup>66</sup> Here we assume that overrides of CIG suggestions made by clinicians are conscious decisions and not mistakes.

treatment does comply (again) with the CIG (Figure 26). At those points it should allow the user to opt back into CIG execution, and if there were no compliance constraints violated, mark the execution trace as compliant (with the guideline). A simple example for this is the case when a physician wants to perform the same actions defined in a CIG, but in a different order. In a more complex example, a physician performs only a subset of the recommended actions and adds actions that are not present in the CIG.

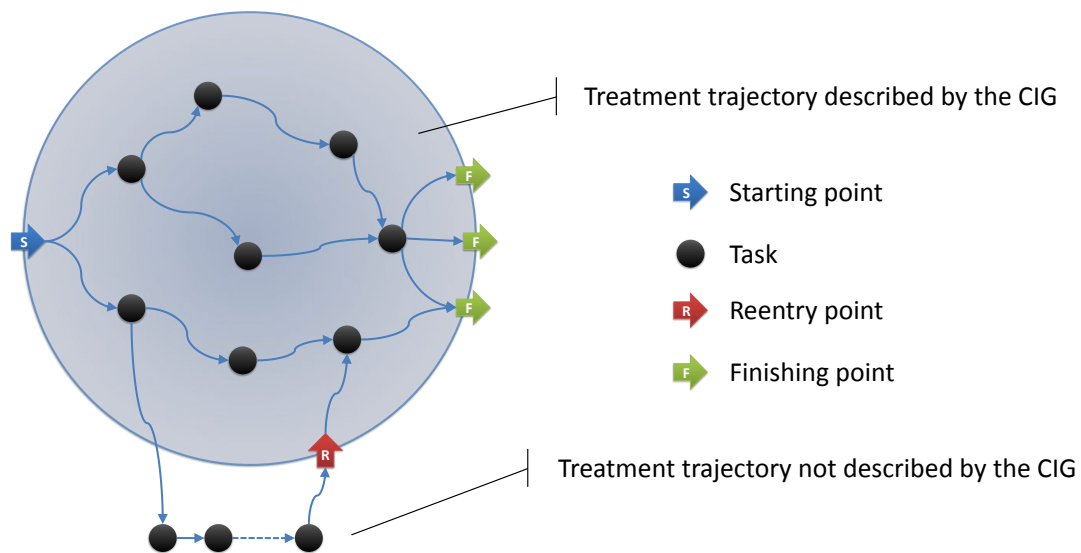


Figure 26 - Grey tracking (illustrated for a TNM-based CIG)

The questions need to be answered for implementing grey tracking include: “What counts as still within the protocol?” and “How can we get back (i.e. reenter) to the once abandoned treatment protocol?”

#### P1.11. Protocol updates “mid-flight”

One of the potential requirements of a scalable CIG model management solution is the ability to apply updates for protocols *mid-flight*. Such an update mechanism would allow the exchange of a protocol to another one during execution. Mid-flight updates can be deemed clinically important if swapping one guideline for another promises better outcomes, or if there is a critical update to the guideline currently in use that needs to be applied.

This is a particularly complicated problem, as the consistency of the treatment needs to be guaranteed during the transition. Preserving the consistency here means that execution traces need to be clinically valid and should not be confusing for experts to follow.

There are multiple options for performing such an update, but not all can guarantee consistency. Discussing this problem in detail is out of the scope of this thesis; however, we illustrate it a consistency

violation with a simple example: Taking protocols that have running instances at the time of an update and simply stopping and replacing them by their updated version could cause undesired effects. There can be a case, where a problem that was continuously treated with medication ( $M$ ) by the original version of the protocol ( $P$ ), is not covered by the updated version ( $P'$ ). Accordingly, the administration of  $M$  will not be recommended by  $P'$ , which means that in the case when  $M$  is a drug that cannot abruptly stopped, the sequence of legal actions will generate a medical error.

*P1.12. Defining rollback strategies (i.e. undo)*

Another complicated problem left untouched is defining the logic for dealing with the effects of false data<sup>67</sup> that has only been deemed as such after it has been processed by the system. In cases, where the unwanted changes can be undone without side effects, the problem translates into implementing a rollback feature (i.e. undo). Ideally, this case would include complicated problems, such as rolling back a set of transactions internal to the EE (e.g. based on a (false) decision to start something a whole set of events are triggered). It would also include rolling back transactions that occurred between the EE and other CIS systems, which is only possible if all affected systems support an undo capability. In cases where changes are permanent (e.g. an injection was already administered), the introduced problem will require the execution of appropriate counter measures, which also should be a part of an ideal system. Additionally, not just the implementation of the mentioned features, but also the suitable representation of mistakes and appropriate remedies needs to be worked out.

*P1.13. Look-ahead feature (i.e. simulation of the “what if” scenario)*

Validation by simulation is an important component of testing protocols, as it allows experts to test how CIGs (or more precisely CIG-based systems) react to a certain situation. For simulation, usually series of these situations – described by timed events and data points – are composed into scenarios (i.e. test cases).

However, there are cases when this type of simulation is not flexible enough. For example, someone might want to look ahead in the trajectory of a running CIG to understand what the protocol would do next, but without actually taking those steps. This type of interactive simulation could be used to serve teaching purposes, to understand the next steps, or it could be used for helping in the preparation for

---

<sup>67</sup> False data can include mistakes made by users selecting the wrong set of actions, and automatic sensors reporting incorrect values.



various prognosis theories (e.g., what if I select treatment A, but the patient's conditions do not improve even after an hour). Solving this problem involves a sophisticated harmonization of live execution and simulation, which includes understanding how far can steps be taken, while still preserving the ability to roll-back and how can fictitious (e.g. prognosticated) data be separated from live ones.

*P1.14. Time control in simulation*

The architecture – including the internal structure – of the simulation environment is not required to be an exact replica of the live system, however as seen in the examined approaches, they are usually the same. This could be a problem if this common architecture does not support the use of arbitrary time sources (i.e. custom clocks). Custom clocks are needed for interactive simulations where time often needs to be sped up, slowed down, or even reversed to better understand what is happening in execution time.

*P1.15. Lack of verification of correctness of CIGs*

As described earlier, verification means making sure that the implementation conforms to the specification. In the case of CIGs, verification entails support for phrasing a range of properties (e.g. goals and intentions<sup>68</sup>, including properties related safety, security and privacy) independently from CIGs that can then be used to check whether or not they hold for the CIGs in a particular environment.

None of the examined approaches do actual verification, although potential solutions, like [163], have been proposed. When approaches talk about verification, they refer either to validation, or to syntax checking.

It was also mentioned in [163] that verification is only possible if there are concepts for formally describing the specification against which CIG models can be tested. This means that CIG languages need to include support for modeling properties, possibly defined as constraints. More on this can be found in Chapter VIII.

---

<sup>68</sup> Verification of goals and intentions are usually performed by evaluating expressions (defined with the help of expression languages) over the decomposed solution.

## P2. Local adaptation

### P2.1. Lack of standardized terminology

To maximize portability, CIGs need to build on standardized vocabularies and data models. However, currently there are many vocabularies and data models (e.g. SNOMED, CPT, ICD-9, LOINC and HL7-RIM) being maintained by many different groups with various interests, which make the selection difficult. The good news is that with the help of the mappings provided by a common ontology layer, such as UMLS, the adaptation can be much more straightforward.

### P2.2. Communication protocol for managing interactions with the host system

Interfacing a CIG-based system with a potential host system require more than agreeing on a common vocabulary and a data model. For seamless integration, defining the communication protocol<sup>69</sup> with the host system is critical. This problem is complicated by the fact that the host system often involves not one, but a suite of different CIS systems, which a CIG-based system might be required to interact with.

Ideally, all participating systems would support (at least) one common communication protocol. This would require all participating systems to define an information exchange interface<sup>70</sup>, which would allow them to be arbitrarily connected. However, as there are no standard CIG formalisms, there are no standardized communications protocols in existence either. Thus, if someone was trying to implement CIG-based clinical care, they would have to also design their own protocol and build the interfaces implementing the protocol for the participating CIS components.

Two related issues are discussed in the following two points.

### P2.3. Feedback mechanisms in a communication protocol

One of the issues related to managing the interactions with the host system is defining the feedback mechanism of the host for actions that were “ordered” by the CIG system. This feedback mechanism

---

<sup>69</sup> A *communication protocol* between a CIG-based system and its host system(s) describes a mechanism for their interaction. It provides a formal definition for digital message formats and the rules for exchanging those messages (i.e. syntax, semantics, and synchronization of communication). The definition includes descriptions for data abstractions, behavioral models (e.g. message handling), addressing, etc. Furthermore, a protocol may implement other related functions, such as authentication, error detection and correction, flow control, etc.

<sup>70</sup> An *information exchange interface* is similar to a service interface. It defines its host’s address, the required authentication method, input and output message types, their sampling rate, max throughput, etc.

allows a CIG-based system to make sure that the actions it expects the host system to perform are completed.

We see two options possible for closing the loop: (1) white and (2) black box methods. In the **white box** method (1), the chain of systems serving the request provide explicit feedback messages, which allow the tracking of each request's state. As an example, let us consider the series of steps leading to a successful ordering of a medication in a typical CIS system suite with the white box method. After a CPOE system receives a request from the CIG system for ordering a 500mg dose of drug  $D_A$  in an injection form, it sends an explicit acknowledgement (or denial) for the particular request. In the next step, the medication order (typically) gets routed to a *pharmacy information system* (PIS) and in turn to an *electronic nursing medication administration records system* (eMAR) [190]. Both of these systems would also report back, the PIS about the successful preparation and the eMAR on the administration of the drug. Such series of status reports (which are sent either directly to the CIS system, or indirectly with a series of messages propagated through the involved systems), makes the CIG system aware whether everything goes according to plan, or not.

In the other method (2), the host systems form a **black box** system, where the CIG system needs to listen and observe events of the environment confirming that the requested action was completed. For example, if a clinician ordered drug  $D_A$  to be administered through the CIG system, the CIG system will need to listen to a shared channel for drug  $D_A$ 's administration to close the loop. This "reverse engineering" has its advantages and limitations. As a benefit, it does not require the host systems to be modified, as long as they provide a method for observing the results (e.g. method for requesting the complete history of administered drugs, or providing a "real-time" channel where administration events can be observed). Additionally, they allow a more scalable integration with other systems and protocols, as they recognize "similar" (i.e. matching type of) actions. An example for this would be recognizing and skipping a particular procedure defined by the CIG for a patient, if another clinician already just performed it. Conversely, the black box method does not provide support for identifying where a particular request is in terms of completion and if assumed lost (e.g. cancelled) or stuck (e.g. waiting for approval) where that might be. Another disadvantage is that while the information recorded in downstream transaction systems can be visualized in multiple ways in the electronic health record, it is not visually linked to the original order. This makes it difficult for both humans and systems to understand the state of the order, comparing the intended plan with what the nurse actually administered to the patient. Furthermore, the decision support logic that assisted the provider in

creating the order, such as parameter based dose calculations, are not carried over with the order sentence to the transaction system. This context is lost once the order is finalized, but is essential information for downstream healthcare professionals to verify the appropriateness of the order.

#### *P2.4. Data provisioning*

None of the publications indicate that authors have considered their CIG-based systems to be data providers for other systems, only as consumer systems. In order for a CIG-based system to act as a data source to other systems, the CIG formalism should support the definition of what the system can be a source for. For example, in the case of the treatment of a particular disease, the diagnosis of the disease confirmed in the protocol could be a piece of information that another system, such as a dashboard indicating the status of multiple patients across multiple diseases, would want to consume. Abstract states, such as the severity level of a condition, could be another example.

#### *P2.5. Avoiding the duplication of functionality of the host systems*

When integrating a CIG-based system into a new environment or when adopting a CIG, it is extremely important to identify which functions will be implemented by which (system) component. This includes addressing the problem of overlapping (i.e. duplicated) system functionality. As an example, the CIG, consequently the CIG-based system, might implement some functionality related to patient safety already addressed by the CPOE system in place. There is a great chance for this, as CPOE systems usually implement the following safety measures before accepting the order for a drug:

- General drug interactions
  - Drug dosing: including one time maximum, lifetime maximum, minimum to be effective, etc. dose and dose rate checks
  - Drug-drug interaction: interaction with other already administered drugs
  - Drug-food interaction
  - Other recommended related (needed or extending) drugs, such as pre- and post-meds
  - Other (potentially better) options achieving the same, including checking the duplication of functionality (e.g. checking the existence of multiple drugs from the same class)
- Drug-patient condition interaction
  - Drug-disease interaction: potential negative effects to other medical conditions
  - Drug-allergy interaction (i.e. adverse reaction)
  - Pregnancy and lactation interaction

- Drug-patient parameter interaction
  - Drug-genome interaction: genetic contraindications
  - Age related interaction (e.g. for geriatric, pediatric patient populations)
- HCO's workflow-specific restrictions (e.g. complying with safety, privacy, and quality of service rules).

*P2.6. Adapting configurable parameters*

As discussed before, CIGs may contain configurable parameters, including thresholds and alternatives, to allow clinicians to tailor the execution of CIGs to a particular patient. At integration time, not only data and action items need to be mapped to local resources, but these options as well. Their mapping is complicated by the fact that they might need to be altered (restricted, or broadened) to fit resource and policy constraints.

A parallel problem is the configuration of local tools that are facilitated by CIG in execution time. The general case of this problem is discussed in P2.2. Similarly to this problem however, all the typical components of a CIG-based CDSS (other than the EE) can be considered as “facilitated” systems. In this case, the CIGs can potentially be required to include configuration for those systems as well. An example for this would be storing information regarding the UI, such as what font to use, or how to display warning messages associated with guideline steps if there are multiple options for it.

*P2.7. Separation of concerns: Lack of ability to provide multiple visual representations*

Separation of concerns is a principle often applied when designing DSMLs. Not surprisingly, all examined guideline formalisms employ it to manage the complexity inherent to CIGs. One of the typical techniques used in formalisms is to separate guideline knowledge into groups (or aspects) related to

- maintenance, which captures meta-information on the CIG model (e.g. version, author, and creation date)
- medical context, which captures the medical context of the CIG (e.g. purpose, keywords, and references to external sources (such as EMR items facilitated, or cited research articles))
- plan knowledge, which capture the set of (medical) actions to reach a certain goal.

This separation, however, does not address two problems, namely (1) control over visualization (including information hiding) and (2) support for local adaptation<sup>71</sup>. Both of these problems are related to the lack of (or minimal) support in the examined languages to create specifications (in their respective abstraction layers) on top of the existing CIG definitions.

This approach in the case of (1) would allow knowledge experts to have control over what information was shown (from what is captured in CIG) and how it was manifested (e.g. there are control mechanisms in the formalism for deciding whether a vital sign should be charted as values in a table or as a graph). It also means, that the same guideline definition could be used to present different views (of the model) for the different HCO personnel (e.g. physicians, nurses, billing personnel; see Figure 27). However, so far none of the formalisms have proposed to support abstractions specifically designed to allow modelers to configure the visual manifestation of (the elements of) a CIG.

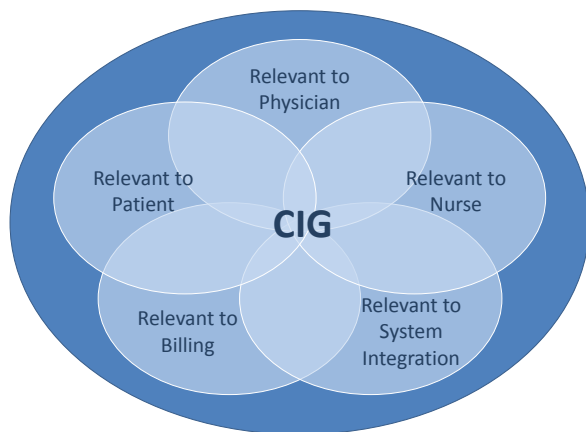


Figure 27 - User aspects of a CIG

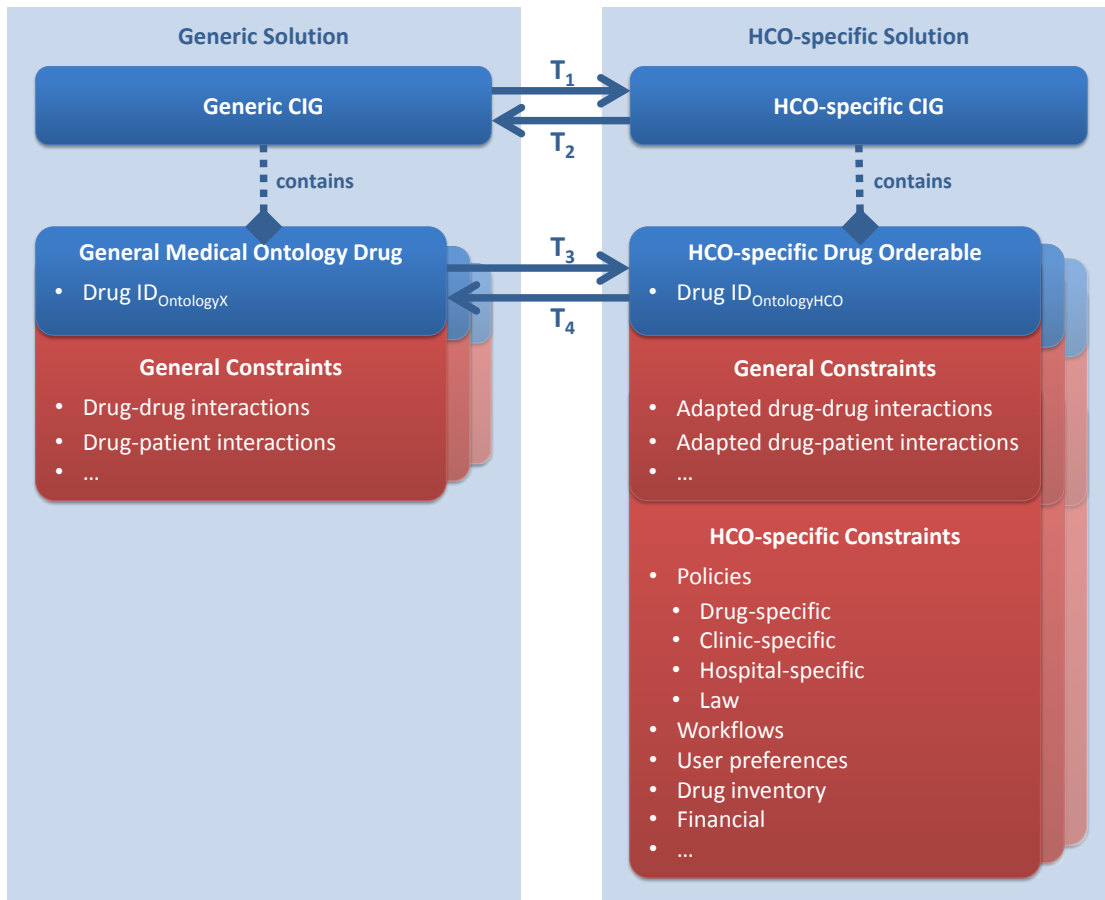
#### P2.8. Separation of concerns: Portability

The other problem mentioned in the previous section is the lack of support for local adaptation of existing CIGs. Formalisms simply do not provide scalable solutions for the generalization of HCO-specific and specialization of generic protocols. We believe that these steps are essential for adapting CIGs for environments that are different from what they were created in. Furthermore, they allow for the creation of tools that can (semi)automatically incorporate CIG-based applications into the existing ecosystems of HCOs.

---

<sup>71</sup> Problems deriving from the unsatisfied requirements defined in point 3) and 4) of the “Modeling requirements” section.

Figure 28 illustrates some of the typical issues with guideline portability. This simplified figure shows two versions of a given CIG: a generic version (seen on the left side), which is built using general drug concepts ( $Drug_{General}$ ), and a HCO-specific version (seen on the right side), which is built using drug orderables available in the provider’s CPOE ( $Drug_{HCO}$ ). In the figure, each arrow marked with a “T” indicates a transformation:  $T_1$  and  $T_2$  are transformations between the two versions of the guideline, and  $T_3$  and  $T_4$  are example transformations between two versions of a typical action item of guidelines, medications.



1.  $Drug_{General} = Drug\ ID_{Ontology\ X} + Constraints_{General}$
2.  $Drug_{HCO} = Drug\ ID_{Ontology\ HCO} + Constraints_{General}' + Constraints_{HCO}$
3.  $|Drugs_{General}| \gg |Drugs_{HCO}|$

---

4.  $Drugs_{General} \neq Drugs_{HCO}$

Figure 28 - CIG portability

$T_2$  denotes the generalization of a HCO-specific CIG. This is a difficult task, as CIGs developed to work at a specific HCO often contain logic captured with the HCO’s local resources and constraints in mind. They

often contain built-in assumptions regarding the host CIS infrastructure, workflows, security policies, etc.

$T_1$  denotes the specialization of a generic CIG, which is another concern. HCOs need to be able to customize CIGs in order to comply with local policy and workflow requirements, map abstract CIG elements (data elements and actions) to local ones, and integrate UIs [70], ideally without altering the original plan.

In order to illustrate some of these challenges we included the description of the transformations for medications.  $T_3$  denotes the adoption process of an HCO when creating or updating their local CIS's action items to include drug definitions specified by a given ontology (e.g. FDB MedKnowledge [191]). During this process, existing drug concepts are paired using a common concept id ( $DrugID_{CommonOntology} = DrugID_{OntologyHCO} = DrugID_{OntologyX}$ ), and existing local knowledge is updated based on the (new) information available in the generic ontology. General constraints ( $Constraints_{General}$ ), such as known drug-drug interactions and associated warnings, are tailored to fit the location's abilities and needs ( $Constraints_{General}'$ ) and location-specific constraints ( $Constraints_{HCO}$ ) are added. For drugs that cannot be automatically paired (e.g. there are drugs described by the generic CIG that are not available at the adopting HCO) clinically valid substitutions are needed. Additionally, such drug concepts may trigger the creation of new orderables, but only if appropriate authorities decide to extend the HCO's existing set of action items to include them.

$T_4$  denotes the inverse process of  $T_3$ , during which HCO-specific concepts are stripped from locally specified constraints (e.g. splitting up bundled a group of medications to individual medication items).

### P2.9. Management of CIG updates

Management of CIG updates is generally an unsolved problem. It relates to the previously mentioned problem of portability. "Local implementations will have modified authoritative guidelines to adapt to local constraints, and will have done considerable work to interface them to their platforms". "As recommendations of authoritative guidelines change, local implementers must be notified of these changes, and must modify their local versions. Maintaining version control and revision history is critical to this task, and means of automating or facilitating the changes required will be needed." [70]



### P2.10. Representation of patient state: Management of information abstraction

Clinicians often build complex and comprehensive diagnoses from simple indicators, such as from observed signs, and reported symptoms and objective data. However, this type of abstract reasoning is not easy to represent with current formalisms. In the following, three specific examples are shown to illustrate why:

#### *Example 1: Aggregating different sources*

Formalisms regularly express the execution of tasks conditional upon patient (health) indicators, for example on cardiac output (CO): `If CO < 4.0 L/min then do TreatmentActionX`. The problem with these definitions is that they are often ambiguous, as it is in the case of CO thresholds: There are a large number of clinical methods for the measurement of CO, ranging from an invasive direct intracardiac catheterisation to a non-invasive estimation using measurements of the arterial pulse [192]. Thus, if different techniques or (types of) devices are used, measurement results may vary in terms of precision. Another related problem is that the field of medicine often uses different thresholds for defining normal value ranges for different types of measurements. A simple example for this phenomenon is the normal values defined for the human body temperature based on where the measurements were taken at [193]. Yet another problem is resolving another issue related to the context of the measurement: Relative to food consumption when was the blood sugar level measured, or was the weight measured with or without clothes on? Last, but not least, measurements can be made using various measurement units, which means that values can only be compared if they are converted to a common unit. These problems make the aggregation of the information sources and the derivation of the abstract information difficult. Regardless of these problems, formalisms need to support the ability to express abstraction rules to allow clinicians to define derivation formulas (e.g. always use the most reliable source available, use the calculated average of all available sources, if the preferred source type A is not available use source type B with using a given conversion method).

#### *Example 2: Defining composite indicators*

The execution of CIGs relies on the availability of health indicators, which in some cases can be automatically retrieved from an EMR. This is usually not the case with *abstract indicators*<sup>72</sup>. One part of abstract indicators can only be defined informally. If not present formally in the EMR, their evaluation

---

<sup>72</sup> *Abstract indicator* is a patient health indicator, which requires a clinician to determine a diagnosis. It can also be interpreted as a not directly measurable patient health indicator, or one, which cannot be directly received from the patients EMR.

requires users to provide the information manually to allow the EE to continue its execution. An example for this type of abstract indicator is a clinical condition called sepsis, which can be defined as observed *systemic inflammatory response syndrome* (SIRS) secondary to an infection [25]. In the case of sepsis, without culture data it would be extremely difficult to automatically infer the presence of an infection from other indicators, especially that the treatment often needs to be started even if there just a suspicion on the presence of an infection.

Another difficulty in dealing with abstract indicators is that their definition can be relative to a disease, to a patient and even to a case. Automatic interpretation of such definitions is only possible for the other type of abstract indicators, called the *composite (or derived) indicators*. Composite indicators, such as hypotension<sup>73</sup>, can be expressed from the logical composition of other (formally defined) indicators. E.g.:<sup>74</sup>

$$\text{Hypotension} = [ (\text{MAP} < 65 \text{ mmHg}) \text{ OR } (\text{SBP} < 90 \text{ mmHg}) \text{ OR } (\text{DBP} < 60 \text{ mmHg}) ],$$

where MAP stands for mean arterial pressure, SBP for systolic blood pressure, and DBP for diastolic blood pressure. Ideally, CIGs define the context of the treatment (i.e. when is the specified treatment applicable), which includes the definition for abstract (and complex) indicators.

If in a CIG the same definition of a composite indicator is used in more than a handful of places (e.g. in guard conditions of various specific treatment actions) it makes sense to define the indicator centrally as a reusable concept. This centralized definition of indicators (e.g. patient condition) has multiple benefits, including allowing users to comprehend the CIG algorithm faster, and making update process (for all instances of these conditions) less error prone. However, it can also mislead experts, who make incorrect assumptions on what the definition might be. Still, we believe that formalisms (including expression languages) should support this feature.

### *Example 3: Identifying the source of indicators*

Formalisms do not support the definition of the source of the electronically available parameters. If they did, they might be able to discover inconsistencies in composite indicator definitions. For example, in

---

<sup>73</sup> abnormally low blood pressure

<sup>74</sup> Other examples include calculating the aggregate daily activity level of an elderly patient from a series of momentary measurements of an accelerometer, and identifying trends based on multiple measurements of an elevated heart rate over time.

the above-mentioned definition of hypotension the inclusion of mean arterial pressure might only make sense if it was a value obtained from something different than averaging the same systolic and diastolic blood pressures. Another point in favor of being able to identify the source is that while CIGs blindly trust the information provided to them, clinicians have tremendous background knowledge, which allows them to second-guess any of the information presented to them. Thus, clinicians would greatly benefit from CIGs that allow them to see where and how the presented information was obtained.

### **P3. Clinical adoption and implementation**

#### *P3.1. Readability of models*

“Great importance has to be attached to the notational convenience of these languages, by which we mean the ease with which they can be read, written, and understood by the potential users of a system, and to the domain experts” [36]. Notational convenience, as well as usability of the associated tools need to be properly evaluated.

#### *P3.2. Handling transactions with a bundle of actions*

If communication protocols (discussed under P2.2) are not intelligent enough, the exchange of composite messages could cause problems during the execution of a CIG. For example, drug interaction checking, including checking for required supporting medications, may be handled by the CPOE system. Furthermore, let us assume that there is a rule for drug A, saying that drug B always needs to be given to the patient before A can be administered. In such a case, if a CIG splits up the ordering of A and B into multiple transactions, there will be a warning if the request for A is processed first.

#### *P3.3. Interactive simulation*

The simulation of CIGs has multiple purposes. It can be used (1) to implement the look-ahead feature (see P1.13). It can be used for (2) the validation of the correctness of the logic captured by the CIGs. It can also be used (3a) to educate clinicians about a particular guideline, or (3b) to test their performance in facilitating them. The common aspect of cases 2 and 3 is that they typically rely on using predefined scenarios (i.e. test cases). Scenarios are defined by controlled data sets, in which data elements are specified in terms of time relative to the start of the scenario. These data elements act as inputs for the CIG in simulation time.

However, there is a significant difference between use cases 2 and 3. Whereas for defining scenarios for the validation of CIGs (case 2) it is perfectly acceptable to use a scenario with a linear sequence of inputs

paired with the expected behavior, for defining scenarios for 3, a more sophisticated technique is required. This is because with 3, and especially with 3b, human interaction is expected during simulation for which the scenario needs to be prepared. An interactive simulation requires a patient model to be captured, which in simulation time is able to “respond” and generate relevant physiological data based on the actions clinicians take. Depending on the purpose of the simulation, a patient model defined as a part of a scenario might be realized by something as simple as defining sub-scenarios and branching logic connecting them. A complicated case might require an expert to perform a simulated reaction of the patient, in other words create the relevant data (e.g. generate vital and lab response) in coordination with the simulated treatment. The point is that all significant sub-cases in a scenario need to have relevant, explicitly defined input data associated with them.

*P3.4. Simulation-specific view*

Both in the case of interactive simulation and in the case of building possible future treatment scenarios for an actual patient visualization of future data is essential. This is particularly true if there are multiple possible options to represent. Thus, it would be important to work out how future data should be released to the execution system and how it should be represented on a UI.

## CHAPTER IV.

### OVERVIEW OF THE SEPSIS PROJECT

The previous chapters surveyed CIG-based CDSSs with an emphasis on the CIG languages and the software architectures of each approach. As the survey demonstrates, there have been many CIG-based CDSSs proposed, with great variance in terms of their scope. Their evaluation allowed for the construction of a general requirements list for constructing these systems. However, it is apparent that none of the examined solutions come close to fully addressing all of the points. Some of these shortcomings are relevant to all approaches and are summarized in the “Open problems” section of Chapter III.

The goal of our research is the *systematic development* of a prototype CDSS system using *model-integrated*<sup>75</sup> development techniques. Our hypothesis is that emerging tools and methods, such as the *Model-Integrated Computing*<sup>76</sup> (MIC) tool suite, have significant impact on the development process, capabilities and acceptance of CDSS and has the potential to lead to a new generation of decision support and patient management tools. Accomplishing this goal requires an application context that is justified and realistic from a medical point of view, its complexity challenges the state-of-the-art in computer science and provides opportunity for real-life evaluation in the clinical environment. We achieved this purpose by establishing a collaborative research effort between the *Institute for Software Integrated Systems* (ISIS) and the *Vanderbilt University Medical Center* (VUMC) on a project titled the *Sepsis Treatment Enhanced through Electronic Protocolization* (STEEP). In this project, clinicians from VUMC, computer scientists from ISIS and the VUMC Informatics Center worked together to provide a CDSS for managing the complicated but well-studied problem of the identification and treatment of sepsis. The partnership provided the interdisciplinary team an invaluable opportunity to investigate model-based treatment management problems, and analyze/understand related technology directions.

---

<sup>75</sup> A *model-integrated* (i.e. model-based) system facilitates abstractions (i.e. models) to define certain properties of the system. In the case of a model-integrated patient management system, models can be used to represent required input parameters, users, who will access the system, and guidelines, which configure the behavior of the system.

<sup>76</sup> *Model-integrated computing* is explained in more detail later in this chapter.

The overall project had significant size and complexity. It included several efforts that were outside of the contributions of this thesis (such as a clinical trial for STEEP supported by an NIH grant, the development of a production version of the system and the integration of STEEP with the HIS infrastructure of VUMC). In this chapter, we provide a brief overview of the overall effort and highlight our specific areas of contributions. The specific research results will be detailed in chapters V-VI.

### **Medical Context: Sepsis Management**

To maximize the potential impact of our prototype system and to help establish design requirements for a CIG-based CDSS, we sought a clinical paradigm that was common, clinically and economically important and had readily available and accepted evidence-based treatment guidelines. We found the detection and management of septic patients to be an ideal candidate for our intervention for the following reasons.

The sepsis syndrome results from a robust host reaction to an infection and is characterized by a systemic inflammatory response, frequently with hypotension and multiple organ failure. The disease process is very common, occurs with a worldwide distribution, and can impact patients of any sex, race, or age (about 750,000 cases occur in the US annually [194], and about 30 percent of septic patients die from the disease [195]).

The management of sepsis is a complex and extremely time and information intensive process, performed in *intensive care units* (ICUs) and emergency departments. Accordingly, severely septic patients consume many hospital resources, requiring on average 7–10 days in ICUs and 3–5 weeks total hospital length of stay. Sepsis-related expenditures are estimated to approach US\$17 billion annually in the US alone [196].

Given the large scope of this clinical problem, it is not surprising that many treatment strategies have been proposed and investigated. The *Surviving Sepsis Campaign* (SSC), led by experts from numerous professional organizations, seeks to improve the diagnosis, management and clinical outcomes. The SSC has published a comprehensive set of treatment guidelines based on graded clinical evidence [197]. The guidelines are widely considered to represent the state of the art in sepsis management, but they will evolve over time. Also, they must be customized to individual patient needs, and their correct application has important quality and cost implications in sepsis care. The SSC guidelines are complex and require multiple time-sensitive interventions based on dynamic patient variables. Correct and timely

implementation of the guidelines requires continuous assimilation and interpretation of numerous pieces of patient data.

The STEEP project incorporated an extensive medical effort conducted by a team of attending physicians and fellows from the VUMC. The primary results of their contributions are the STEEP guideline models, user interface and usability evaluations and in-depth guidance to the technical work from clinical point of view.

### **Functional Architecture of STEEP**

The functional architecture is shown in Figure 29. The architecture is composed of a suite of existing VUMC CISs (colored yellow) and our contributions (colored blue), namely the *CIG Modeling Environment (GME)*, the *CIG Repository*, the *Execution Engine* and the *Treatment Management Console*. The GME Modeling Environment supports the graphical editing of STEEP models formally captured in a newly developed modeling language, called the *Clinical Protocol Modeling Language (CPML)*. The Modeling Environment is not part of the system execution; it provides an interface for creating, modifying and validating the Protocol Models.

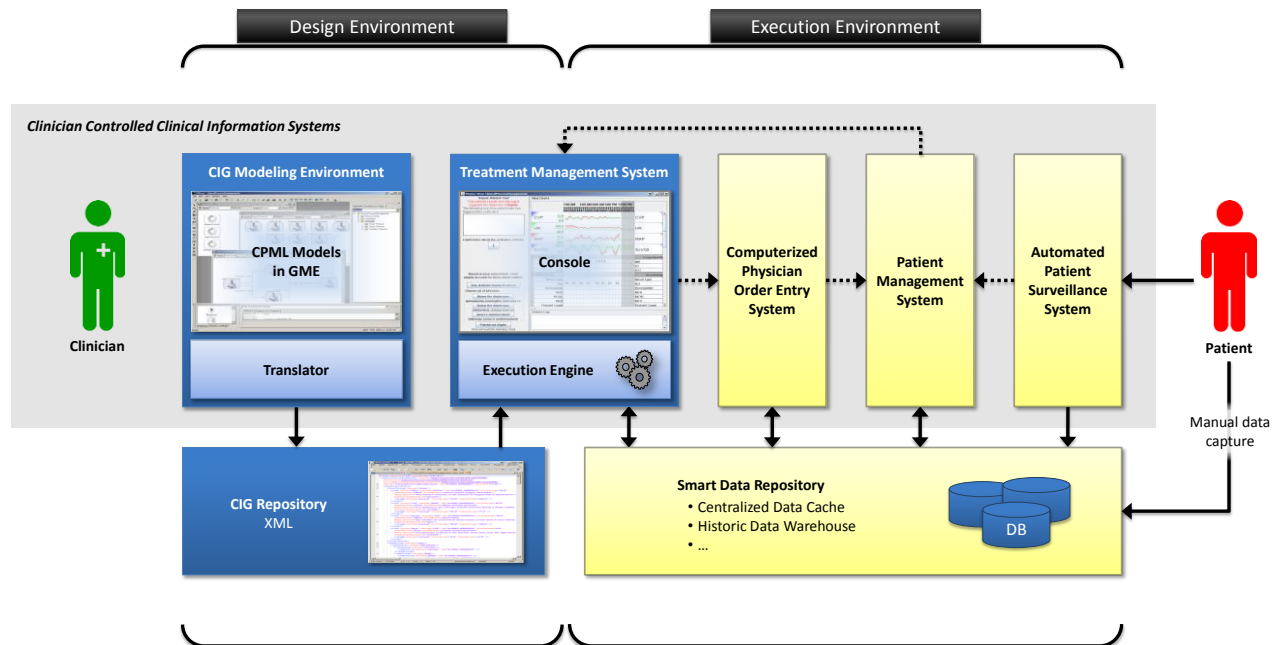


Figure 29 - Functional Architecture of STEEP

The execution of the patient management system is supported by the rest of the functional components in Figure 29. The execution includes two phases:

1. Pre-diagnosis surveillance and alerting
  - a. Identify patients potentially having sepsis
  - b. Prompt clinical teams
2. Treatment management
  - a. Once pending diagnosis confirmed, provide continuous real-time process management recommendations based on live patient data
  - b. Process confirmed orders

#### **Pre-diagnosis surveillance and alerting**

The detailed steps are the following. The *Automated Patient Surveillance System* in STEEP monitors real-time patient data streams (coming from the *Smart Data Repository*) and, using specific laboratory and vital signs criteria, identifies patients with possible sepsis. These monitored abnormalities are quite sensitive for the diagnosis of sepsis, but lack specificity without clinical input and contextual interpretation. Therefore, patients with an “alert status” need a healthcare team’s assessment. These patients are identified first by a visual cue on an ICU patient management dashboard (part of the *Patient Management System*). If this alert is not addressed in a timely manner, an electronic notification via text page is sent to appropriate team members. If there is a reasonable suspicion that the abnormal physiological parameters are due to infection, the physician activates decision support.

#### **Treatment management**

Following a confirmed diagnosis by a clinician, the *STEER Execution Engine* is instantiated for the patient and starts running the treatment management process by executing the protocol model. The executable version of the protocol models is an .xml file generated from the formal, GME-based Protocol Model. The Execution Engine also interacts with the *Treatment Management Console*, a GUI that physicians use to assess the treated patient’s health status, get decision support from evidence-based guidelines on the screen, and actuate their decisions. Recommendations are generated by continuously interpreting the protocol as new information from the environment (including the patient and the clinicians) arrives.



The interaction between the Physician and the system is facilitated by the Treatment Management Console by means of two panels: the Monitoring Panel and the Advisory Panel. The Monitoring Panel presents a timeline where categorized patient health information can be viewed in time in context with the actions of the therapy provided to the patient. The Advisory Panel presents the set of actions recommended by the protocol.

STEEP is integrated with the CIS infrastructure of VUMC via interfaces to the Patient Management Dashboard (StarPanel) and to the Clinical Information System (StarChart). Real-time patient data are received by the Surveillance Tool to detect sepsis symptoms using a set of rules. Additional real-time patient data including physiological data, and a range of treatment data are stored and continuously update in the Clinical Information System. (Further details about these components will be provided in Chapter V.)

The model-integrated design approach is reflected in the context of the functional architecture in the following ways:

1. The protocol models fully determine the behavior of the STEEP patient management system. The Execution Engine serves as a model interpreter that generates responses to received input from the user interface and the CIS interface according to the state of the patient and the state of the treatment process.
2. The Treatment Management Console is configured/customized via models captured as an integrated modeling aspect in CPML. This solution enables the design of a dynamic user interface that can be adjusted according to various treatments and treatment phases.
3. The STEEP Execution Engine is integrated with the CIS via model-configured data and information model interfaces. Configuration of the data interface requires the specification of real-time data streams carrying patient vitals to the STEEP Monitoring Panel. The information model interface links modeling concepts used in STEEP for describing medications, labs, and other orderables to the terminology used in VUMC CIS-s, such as the CPOE.

Our contribution to the STEEP architecture includes the extension of the model-based approach from the narrow protocol modeling and execution to the broader model-integrated design approach that involves the implementation of previously described components (2) and (3). Details of this contribution are described in Chapter V.

## Modeling Language and Model Development

The use of the model-integrated approach in the STEEP architecture redefines the overall development challenge from pure software development to a model and software development process. One of the primary promises of model-integrated computing is that most (or a large part) of the overall system complexity can be expressed using formal models that are easier to create, modify, validate, verify and reuse. Accordingly, the traditional software development and system integration tasks are significantly decreased and their results are becoming highly reusable in a wide range of domains. The core part of our work focuses on the modeling process in STEEP.

We decomposed the model development process into the following tasks:

1. Design of an integrated suite of DSMLs for representing treatment protocols, component configuration information and system integration information. The language suite must provide sufficiently rich abstractions for the targeted domains allowing the economical representation of knowledge components.
2. Define *structural semantics*<sup>77</sup> for the DSMLs that enables the use of static model verification techniques for checking well-formedness rules.
3. Define the *execution semantics*<sup>78</sup> for the DSML components by specifying the transformation of CIG models (protocols) onto an execution platform that ensures protocol enactment and provides interoperability with a host CIS infrastructure.
4. Define the *behavioral semantics*<sup>79</sup> for the validation and verification of protocols by specifying the transformation of the CIG models into the input language of a CIG implementation analysis tool suite.

---

<sup>77</sup> *Structural semantics* of modeling languages define the set of well-formed models. The Model-Integrated Computing tool suite uses metamodels to express static semantics.

<sup>78</sup> From here on, we have to differentiate *execution semantics* from *behavioral semantics*. *Execution semantics* is considered as the definition of the complete system behavior (i.e. how and when the various constructs of a DSML should produce a program behavior). Considering the example of a CDSS that is integrated into its host CIS using SOA techniques, the execution semantics describes the complete low-level message exchange protocol governing the communication including the socket configuration, message rate, error handling, etc.

<sup>79</sup> *Behavioral semantics* can be considered as a subset of what the execution semantics defines: behavioral semantics only considers system behavior that is in the focus of interest and abstracts out the rest. In this thesis the behavioral semantics is used to analyze certain safety properties of implemented CIGs, while for example underlying reliable communication with the host CIS is assumed.

5. Develop models of specific treatment protocols (i.e. CIGs) using the modeling language defined in step 1. These models are a formal representation of guidelines that can drive the management of clinical (treatment) processes.
6. Translate and analyze safety properties of built CIGs. The precise semantic foundation of the MIC modeling infrastructure and related tools enable validation and verification of the models against a range of safety, privacy and security related criteria defined as constraints or policies.

The main research contributions of this thesis are in Tasks 1, 2, 4, and 6 with additional contributions to the execution of Tasks 3 and 5. Detailed discussion of the research results are presented in Chapter VI.

### **Software design and implementation**

While the design and implementation of the STEEP software components were not a direct part of our research, they progressed in tight coordination with the model development process. Implementation of the software architecture included the following tasks:

1. Treatment management console development with specific considerations on a well-structured user interface that helps rapid understanding of the patient state and treatment state by clinicians. User interface design was strongly influenced by both domain experts and a VUMC human-computer interaction (HCI) expert.
2. Design of the STEEP Execution Engine that runs simultaneous instances of the protocol. The Treatment Management Console and the Execution Engine were implemented using the client-server architecture.
3. Configurable integration components and interfaces to the Clinical Information Systems including EMR, Patient Management Dashboard and Surveillance tool.

## Evaluation

Systematic evaluation of STEEP has been a complex, multi-faceted effort. The main phases of the evaluation include:

- **Model evaluation:** The goal of the model evaluation is to decide if the protocol models reflect the current state-of-knowledge in sepsis treatment. Model evaluation is performed by an integrated physician and computer science team working on the STEEP project. After the development of the required models, an initial evaluation confirmed their correctness. Subsequent evaluations are now part of the workflow that supports the necessary ongoing continuous updates of implemented guidelines.
- **System evaluation:** The goal of the system evaluation is to determine if the STEEP system is suitable for use in ICUs. The system evaluation is performed by the VUMC quality control personnel. As a result of the successful evaluation, STEEP was integrated into the live production environments in VUMC's MICU and SICU.
- **Clinical evaluation:** The goal of the clinical evaluation is to determine improvement of outcomes in patients with sepsis. Clinical evaluation is ongoing by a VUMC team under grant support from NIH.

Our research contributed to the model and system evaluation processes. The language design and development effort was tightly integrated with the model evaluation and resulted in several significant updates to the CPML language to improve expressiveness and provide better structuring for the protocol model. The model-integrated system architecture has profound implication on the system evaluation. We could successfully argue, the model updates have not changed the software, therefore the time and cost of the quality control process was increased. We will discuss this issue further in Chapter IX.

## CHAPTER V.

### A MODEL-INTEGRATED IMPLEMENTATION ARCHITECTURE FOR STEEP

The model-integrated development of STEEP is related to the *Model-Driven Architecture* (MDA) [198] software development approach. In MDA, system functionality is specified using *Platform Independent Models* (PIMs) expressed in some DSML. The PIMs are automatically translated into executable *Platform Specific Models* (PSMs) that are usually specified as a combination of some platform modeling language corresponding to a composition platform (e.g. Corba) and some general purpose programming language (e.g. Java or C++).

However, while MDA based software development traditionally starts with selecting some standard PIM modeling language suites and a standard target platform, in STEEP we followed a different approach. The model-integrated architecture development starts with separating models and associated modeling languages that express the changing, rapidly evolving and complex aspects of system operation from the model-based components of the execution platform that interpret the models. This followed by the integration of the system into an underlying information infrastructure that provides a stable, reusable infrastructure not only for STEEP, but, by changing the models and modeling languages, for other guideline driven patient management systems as well.

The fundamental challenge in model-integrated architecture design is the identification of the models, modeling languages and model-based components, their connections and their interfaces to operators and to the given CIS infrastructure. As in most architecture design, the design decisions have major impact on the cost of the system implementation, various aspects of system performance, testability and future acceptance. In this Chapter, we provide a summary of the STEEP architecture design with emphasis on specific contributions. Evaluation of the STEEP architecture decisions is discussed in Chapter IX.

#### **Implementation architecture overview**

The functional architecture of STEEP in Figure 29 is implemented by the *Clinical Process Management Architecture* (CPMA) shown in Figure 30. Design of the CPMA was driven by the needs of integrating STEEP into VUMC's existing CIS infrastructure.

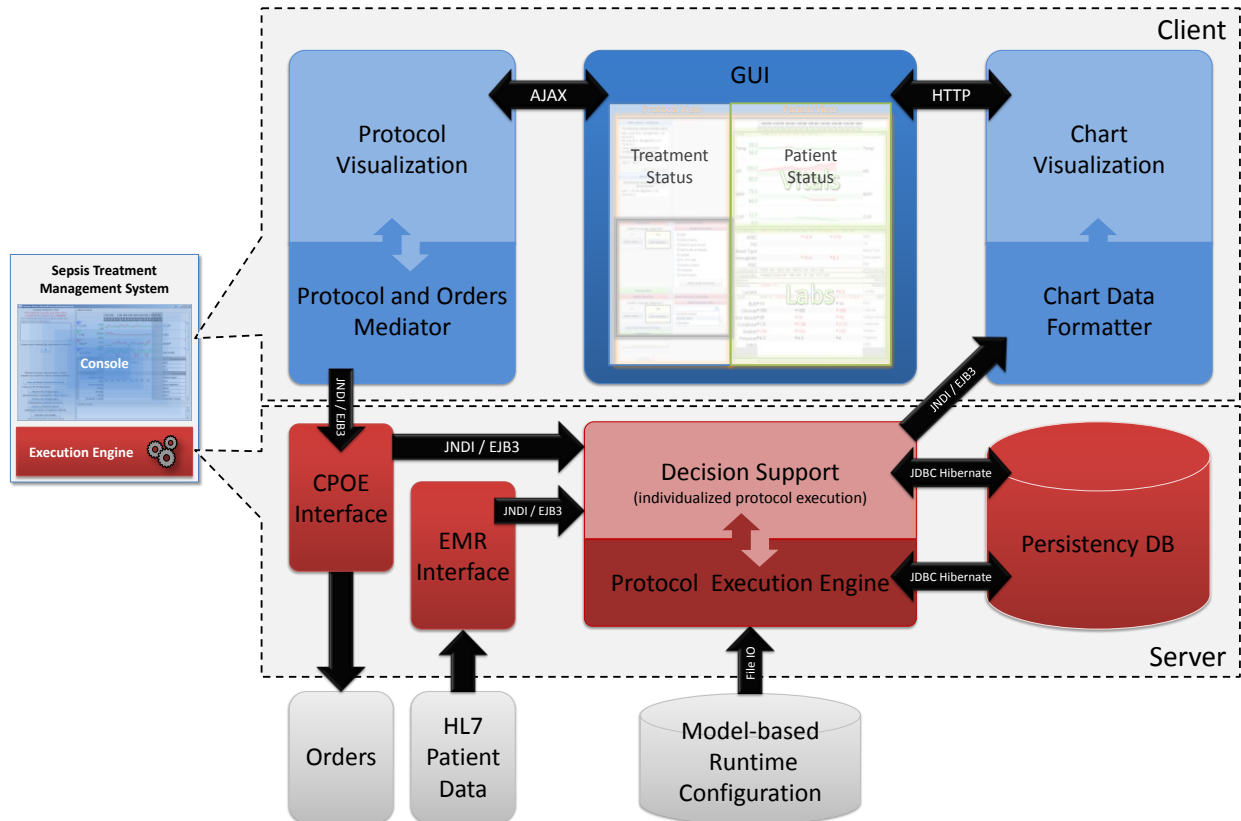


Figure 30 - CPMA: The implementation architecture for STEEP

The main components of the STEEP CPMA, The Treatment Management Console and the Execution Engine have been implemented as a client-server architecture. The client side runs in standard browsers available at clinical workstations (CWS) dispersed in the ICUs, communicates treatment and patient status and interacts with authorized ICU personnel. The server side includes the integrated Decision Support and Protocol Execution Engine (briefly Execution Engine), the Persistency Database for storing detailed treatment data for each patient and interfaces to the EMR and the CPOE systems of VUMC. The implementation technology of STEEP utilized Java component technology (JNDI/EJB3), JDBC Hybernate for Persistency Database, and AJAX for client-server interaction and HTTP for client-side visualization (GWT). Since STEEP needs to serve a large number of patients simultaneously, the server side is multi-threaded with serialized access to the Persistency database.

### Treatment Management Console

The Treatment Management Console facilitates the interaction between the ICU personnel and the system by means of two panels: the *Monitoring Panel* and the *Advisory Panel* (see Figure 31).

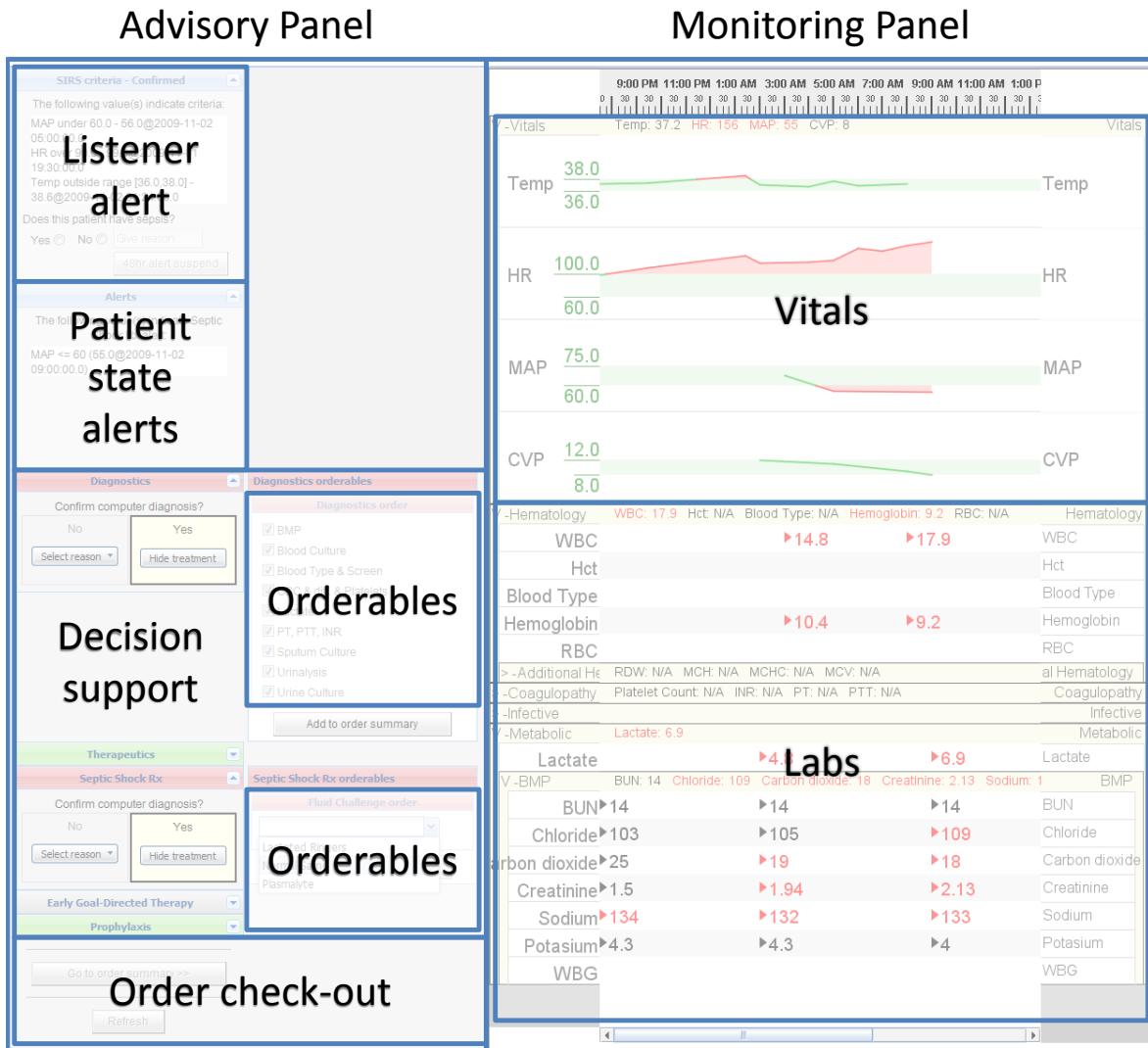


Figure 31 - STEEP GUI: Structure of the Treatment Management Console

The monitoring panel presents a timeline for viewing categorized patient health information in context with the therapeutic actions provided to the patient. Displaying cause and effect relations involves linking patient data and treatments so that the effect of one on the other can be seen; this is what we refer to as the action-reaction concept. The protocol models define this information (both displayed indicators and available treatment actions). In effect, they transform the generic GUI to a protocol-specific interface. The timeline runs from the past, when the treatment started, to the current time. Health indicators, fed to the system as a stream of data, include vital signs, such as temperature, blood pressure, heart rate and central venous pressure. Laboratory test results, like the white blood cell count, are updated on the screen when the information becomes available. The panel also shows the actions of

the treatment that were, or are scheduled to be, provided to the patient (e.g. the start of a normal saline (NS) treatment). All displayed data is temporally aligned in the same columns.

The Monitoring Panel is divided up into two fields:

1. Vitals, which represent locally and usually frequently measured physiological indicators of the patient (e.g. temperature, heart rate, mean arterial pressure and central venous pressure data).
2. Labs, which represent infrequently and/or remotely measured physiological information (e.g. white blood cell count, international normalized ratio and lactate levels)

The other panel, the advisory panel, helps clinicians make a formal diagnosis by using the built-in logic, available action controls and diagnostic information for external systems. Its main components are the following:

1. Listener Alerts displaying the status of the Surveillance Tool, including a list of the various levels of alerts triggered over the patient and the associated objective data,
2. Patient State Alerts displaying the progression of the disease with the help of higher-level diagnostic information (e.g. sepsis severity level),
3. Decision Support functions displaying categorized treatments for various problems that at the lowest level include recommending specific tests, medications and procedures.

Figure 32 shows an actual instance of the state of the Treatment Management Console. The monitoring panel shows a day worth of measurements for four vitals (temperature, heart rate, mean arterial pressure and central venous pressure), marked red when out of the predefined normal ranges. The advisory panel shows that as part of the treatment the user confirmed alerts from the surveillance tool and currently being presented with the next recommendation step, called “Diagnostics”, marked with yellow. As part of this step only the “Diagnostic orderables” component needs to be addressed, which means the ordering of various laboratory test.



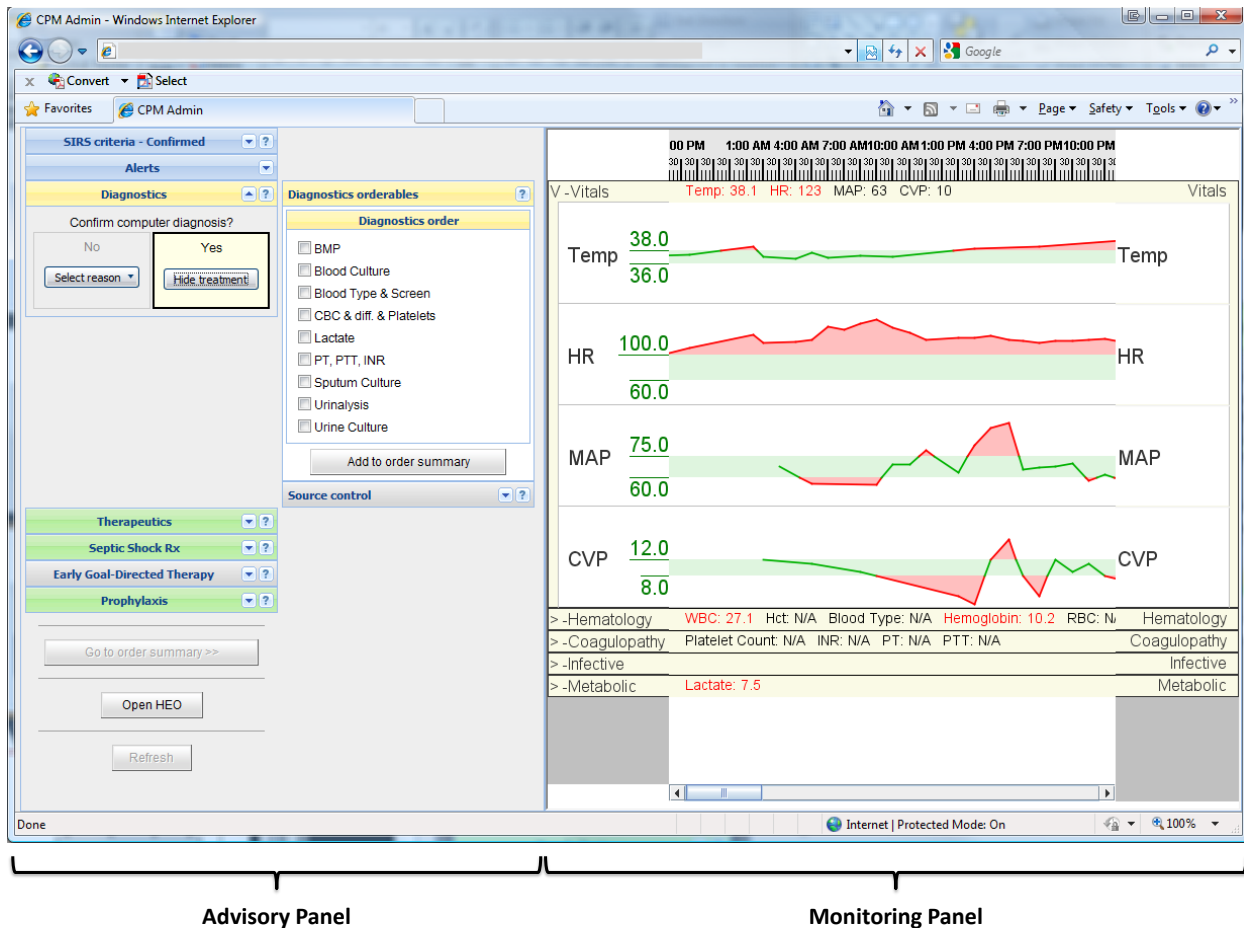


Figure 32 - STEEP GUI: The Treatment Management Console

### System integration interfaces

Development of advanced CIS is an inherently experimental process. It requires an in-depth evaluation of the clinical environment to understand impact on the quality of care. Unfortunately, creating credible experiments in the form of carefully controlled clinical trials is a complex task, because of the need for integrating the experimental system with heterogeneous health IT infrastructure, such as the EMR, CPOE, authentication systems, audit logs and whiteboards. Integration in real life environments can be prohibitively expensive without a systematic approach (e.g. facilitation of existing standards) that helps rapid system integration.

Due to its complexity, addressing and discussing all aspects of the integration challenge in detail is out of the scope of this thesis. The contributions of this thesis are restricted to those integration issues that directly influence the modeling language design.

## **EMR Interface**

To receive information from EMRs, we connected to VUMC's centralized EMR repository system (Core Cache) using the messaging component of HL7. Their centralized system allowed access to vital sign measurements using SNOMED identifiers as well as to lab test results using VUMC's proprietary coding scheme (Chisl code). As for getting access to completed orders, STEEP needed to subscribe to a non-standard data feed with another proprietary coding scheme (Svc code), which was provided by an order logging system (TDQ).

## **CPOE Interface**

Similar to the data receiving interface, communication towards VUMC's CPOE (Horizon Expert Order) needed to be implemented. This was a much more complicated problem, because VUMC's CPOE was not ready for accepting orders from external systems. A close collaboration of its design team and STEEP collaborators allowed the design of a proprietary interface through which both partial and complete orders could be placed (using the Portobello proprietary coding scheme).

In the aforementioned cases, the challenge in terms of language design was to represent interface elements for both EMR and CPOE systems to enable the expected communication, while making sure that guidelines built using these components were executable without the VUMC-specific systems in place.

## **Contributions**

Development of the STEEP implementation architecture and completing the system implementation has been a significant effort by the ISIS and VUMC teams. The current status of the project is described in Appendix B. The STEEP system implementation has been primary completed by Andras Nadas, ISIS, while the CPOE and EMR integration was accomplished by the VUMC team led by Dan Albert. Our work contributed to the effort in the following areas:

1. Leading the integrated modeling language development required for the model-based configuration/customization of the Treatment Management Console and the System Integration Interfaces
2. Contribution to the GUI design and selection of the configurable features.

3. Contribution to the design of the Execution Engine that runs simultaneous instances of the protocol.
4. Contribution to the design of the proprietary CPOE communication interface.

In summary, the main research contribution of this thesis to the implementation architecture and implementation tasks is decomposing the architecture into components and interfaces that can be configured from models, integrating these models into the overall modeling language suite and enabling (in a limited sense) model-based system integration.

## CHAPTER VI.

### MODELING LANGUAGE AND MODEL DEVELOPMENT

As in all model-integrated system development, modeling languages play a central role. In STEEP, modeling languages provide the abstractions that will be used for defining all essential aspects of managing septic patients and integrating STEEP in the CIS infrastructure. There are a number of scientific challenges in identifying these abstractions:

1. **Shared conceptualization:** DSMLs include concepts that describe domains using a vocabulary. Freedom in selecting the terms in vocabularies is restricted, because the knowledge expressed by STEEP models is integral part of a complex “knowledge context” used in a live, evolving clinical environment. In fact, conceptualization selected in the STEEP modeling language suite needs to be shared by other related clinical domains: order management, treatments, drugs, EMR and others. Since these domains evolve independently, each with a different life cycle, the conceptualization used in CPML needs to be structured into independent vocabularies and the vocabularies need to be composed with the modeling languages. This is a hard challenge both scientifically and technically. Our approach was based on the iterative development of alternative conceptualizations, continued evaluation with medical and informatics experts and the rapid revisions of modeling environments and models.
2. **Modeling language specification:** Terms of the vocabularies are used as concepts in DSMLs. However, DSMLs are richer than the superset of concepts: they define modeling domains that include the set of well-formed models. The formal specification of domains is the structural semantics of DSMLs. In MIC, the structural semantics is defined using metamodels. Defining the precise structural semantics for the suite of modeling languages in STEEP constitutes a second scientific challenge in the language development.
3. **Model specification:** Models defining treatment protocols, system configurations and the integration of the STEEP execution platform to the CIS infrastructure are one of the “end products” of the model-integrated development approach. Their simplicity, expressivity, safety and unambiguity are essential indicators of the quality of the DSMLs developed.

In STEEP, the modeling language CPML has been defined as a suite of connected DSMLs. To utilize existing work, our approach draws from and extends the general design principles of existing model-

based CIG solutions summarized in Chapters II and III. In this chapter, we focus on some of the key challenges of the construction of CIG-based CDSS including (1) **finding the appropriate level of abstractions for the highly reactive intensive care patient management**, and (2) **providing the ability for domain experts to construct HCO-specific CIGs without weaving the knowledge specific to a HCO with medical knowledge**, thus allowing for less laborious customization of CIGs for adaptation at other HCOs. Evaluation of the modeling language development results are summarized in Chapter IX.

### **Modeling language development approach**

In STEEP, the CPML modeling language supports the construction of three types of models:

1. treatment protocols,
2. component configurations and
3. integration specifications.

The reason for considering these models together is that they are interrelated: models of component configurations and integration specifications are “model-level interfaces” between the STEEP patient management system and other clinical information systems and users.

In the model-integrated framework, the first step of the development process is the specification of DSMLs for the essential subdomains and their interactions. However, there is a fundamental question regarding the selected approach to language design. Specifically, **should we adopt an existing language or proceed with developing a new one?**

As proposed in [70], many of the problems (such as the generalization of HCO-specific protocols, or avoiding the duplication of functionality of the host CIS) could be addressed in a more cohesive manner if a common shared representation were to be mutually developed by modelers and implementers. However, the same source concluded that – because the various current conceptual models for treatment protocols are sufficiently different – **it is unrealistic to build a single common modeling language** that incorporates all of their abstractions.

This pressure towards fragmentation is the basis for the well-known dichotomy between domain specificity and reusability: utility of CIG modeling languages increases by adopting domain specific

abstractions, but at the same time domain specificity, which translates to decreased scope<sup>80</sup>, results in decreased reusability of models and tools across domains. Currently, there are two common approaches to resolving this dichotomy:

1. Development of standardized modeling languages that are sufficiently broad in scope to span different domains with the hope that modeling, verification and synthesis tools will appear (eventually) and adopters will accumulate assets that in turn will establish a self-sustaining infrastructure and market. Based on our review of the state-of-the-art in CIG modeling languages, this clearly has not happened yet. While there are numerous examples of industry- or academy-driven attempts for standardization, there is no single emerging standard that may count on general acceptance. Among the many reasons are the extreme heterogeneity of the CIG domains, the lack of formally specified semantics for the modeling languages that would enable their use in different domains, and the general lack of support tools that would make a difference. Even more interestingly, this approach brought quite limited success in model-based engineering as well, where generic modeling languages, such as UML, AADL, SysML and many others have highly limited domain penetration - even more than a decade after their introduction.
2. Adoption of a modeling language that is used by a tool (or tool suite) providing some usability for the domain, even if the abstractions are not fully appropriate and the tool suite capabilities have major gaps. This leads to - what Alberto Sangiovanni-Vincentelli calls – the “tyranny of tools” in design automation, when the available tools, and not the design problems dictate the abstractions that designers should use in problem solving [199]. This approach typically fails in heterogeneous domains that require integration of tools – and with them – modeling languages. The consequence is the familiar “islands of automation” infrastructure that sporadically covers the needs of a domain.

Lately, the introduction and increased use of metaprogrammable tools, such as the MIC tool suite [200], have started changing this situation. They enable the use of domain specific languages that are the least complicated and most relevant to a domain, without sacrificing precision and advanced tool support.

---

<sup>80</sup> i.e. its ability to represent other problems

## Design of CPML

These findings were further exacerbated by our intention to take advantages of model-based methods not only for treatment modeling, but also for experimenting with various semantics, and customizing and integrating the system in a host environment. As none of these features were supported by the examined approaches, we decided to create a new modeling language, called CPML.

The precise specification of CPML proved to be a hard problem due to the following challenges. First, operational protocols, policies and treatment guidelines of healthcare organizations are rarely phrased in a mathematically sound, unambiguous manner. Second, the protocols that describe the medical processes constituting a treatment, their triggering conditions and their coordination methods need to be considered as guidelines and not rigid workflows that must be enacted always the same way. This requirement is essential for the design of the execution semantics of models.

### **DSML design**

A DSML defines a domain as the set of all structurally well-formed models [201]. A specific model is a “point” in the domain. A well-formed model is a model that satisfies all the constraints imposed on its construction. Formally, a domain is

1.  $\gamma$ : A set of concepts from which models are built
2.  $R_\gamma$ : A set of possible model realizations
3.  $C$ : A set of constraints over  $R_\gamma$ .

The model realizations represent all allowed ways that models can be built from the available primitives. The set of well-formed models in a domain  $D$  is the set of all models that satisfy the constraints. This set construction is written as

$$D(\gamma, C) = \{r \in R_\gamma \mid r \models C\}$$

where the notation  $r \models C$  can be read as “ $r$  satisfies constraints  $C$ ”. Domains may carry meaning beyond their structure. This meaning (such as behavior) is expressed as a mapping of models in one domain to models in another domain with existing behavioral semantics. This mapping is called interpretation ( $\llbracket \cdot \rrbracket$ ):

$$\llbracket \cdot \rrbracket: R_\gamma \rightarrow R_{\gamma'}$$

Every domain has at least one interpretation, which is the structural interpretation; this is called the *structural semantics*. A domain may have other interpretations as well, expressed as a family of mappings  $(\llbracket \cdot \rrbracket_i)_{i \in I}$ . The interpretations, together with the behavioral semantics of the target domain, define the behavioral semantics of the domain. Based on these notions of domains and interpretations, a DSML  $L$  is defined as a 4-tuple comprised of its domain and a set of interpretations.

$$L = (\gamma, R_\gamma, C, (\llbracket \cdot \rrbracket_i)_{i \in I})$$

Based on these definitions, the specification of CPML includes the following steps:

1. Specification of the set of concepts  $\gamma$  providing the vocabulary of CPML
2. Specification of the  $D(\gamma, C)$  domain of CPML. This specification is provided using *metamodels*<sup>81</sup> and the *metamodeling language of the MIC tool suite*. Once complete, these metamodels are used for the automated customization of the metaprogrammable tools, such as the *Generic Modeling Environment* (GME) and model-management tools, such as the *Universal Data Model* (UDM).
3. Specification of  $(\llbracket \cdot \rrbracket_i)_{i \in I}$  interpretations for CPML as model transformations for defining semantics as required.

Before discussing these steps in detail, below we summarize early attempts that helped in formulating the direction of our research.

### Early attempts

The formal specification of CPML has proved to be difficult for two main reasons:

1. Health care organizations rarely phrase operational protocols, policies, and treatment guidelines in a mathematically sound, unambiguous manner. While the medical knowledge available is rich, it is highly context dependent and provides room for different interpretations.
2. Healthcare practitioners must consider the protocols that describe the medical processes constituting a treatment, their triggering conditions, and their coordination as guidelines—not rigid workflows that must be enacted the same way every time.

---

<sup>81</sup> Thus, the DSML (definition) can be referred to as the metamodel.



Due to these challenges, the language development took several iterations. In our first attempt, the language explicitly represented treatment trajectories as a connected, directed, bipartite graph structure. The nodes were either decision points with predefined multiple possible outcomes or actions representing treatment steps. The advantages of this approach were that it followed the formalization efforts presented in the available medical literature (e.g. see Figure 33) and that it was simple enough. However, this approach did not prove to be efficient for expressing complex treatments because of the exponentially large number of potential trajectories generated by the many concurrent and interacting treatment processes.

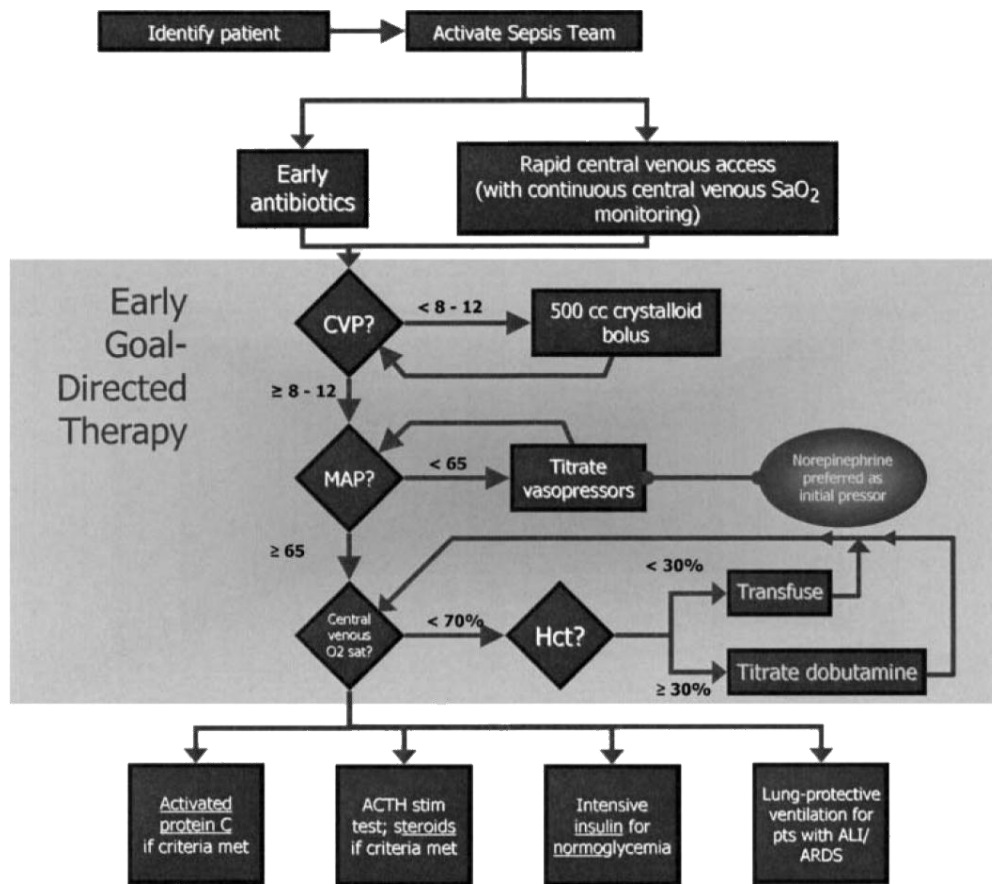


Figure 33 - The Multiple Urgent Sepsis Treatments (MUST) protocol [22]

Figure 34 shows an example for an early attempt to represent the treatment protocol using simple workflows. The analysis allowed us to discover that treatment steps can be grouped together into

*bundles*<sup>82</sup> that are typically executed together. However, it soon became clear that execution of these bundles can overlap, influence each other, and may result in a wide variety of treatment trajectories that would be impossible to capture.

---

<sup>82</sup> Bundles are “interventions related to a disease process that, when executed together, result in better outcomes than when implemented individually” [202].

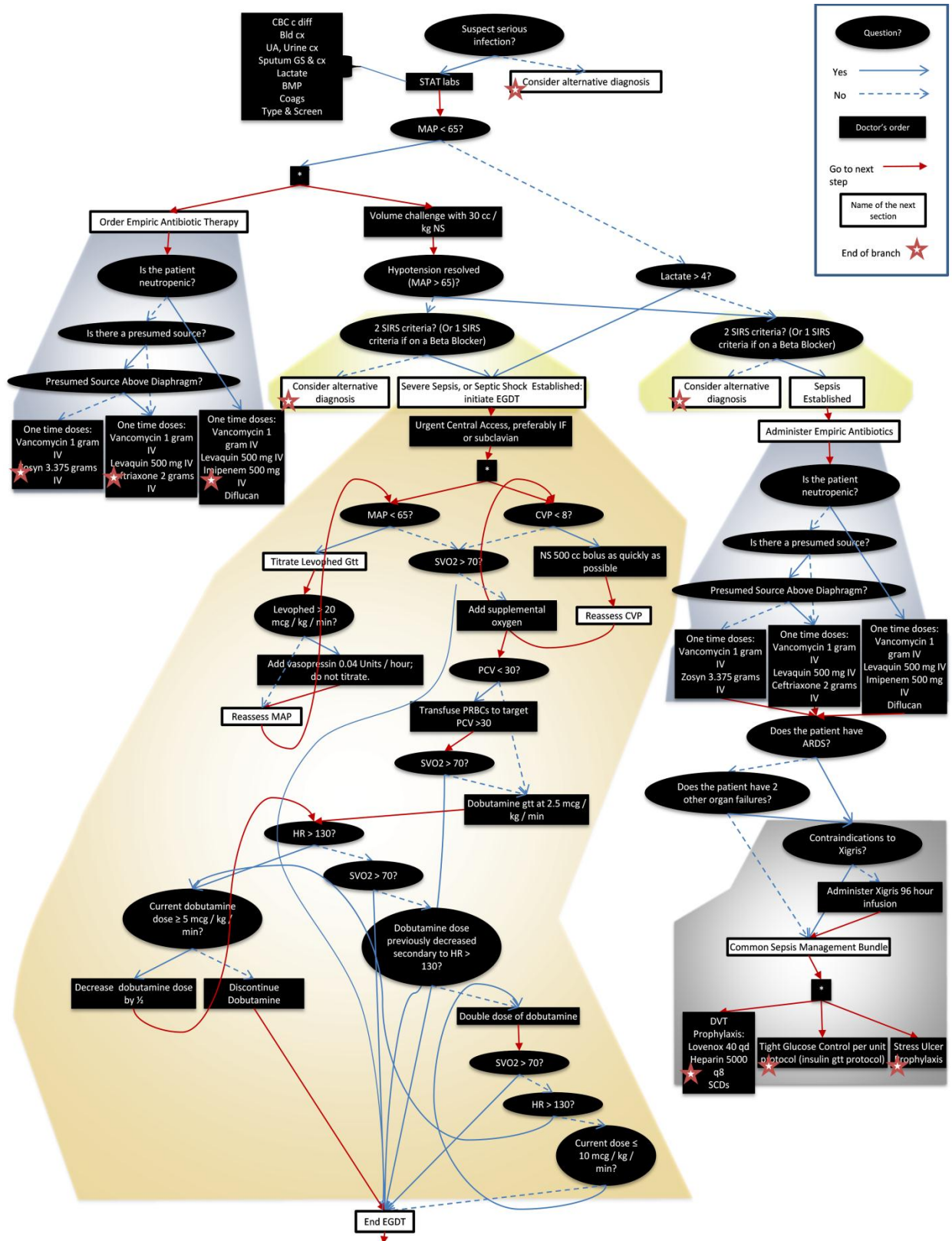


Figure 34 - Initial version of the VUMC sepsis treatment protocol

Another basic question in designing the protocol language was if it should primarily reflect the state of the treatment process, the state of the patient or both. Discussions and experiments clarified that most of the language complexity and expressiveness should be linked to the appropriate modeling and tracking of the treatment process. The emphasis was on the need for describing the patient state with a relatively low resolution (i.e. objective data) primarily to allow the physicians to formulate their own opinion; however, a high-resolution description was also necessary to enable the automatic construction of diagnoses.

These considerations led us to the latest iteration of the CPML semantics: the protocol modeling language describes treatment steps that are grouped together under the concept of *treatment processes*. Processes are concurrent, asynchronous and can interact with each other via *events*. In order to capture the decision logic concisely, processes can be organized in a hierarchical manner. Processes listen to events happening around them and only start running if their triggering conditions are satisfied. Coordination of processes is done with the help of events (and related messages). The behavioral semantics of the selected process model corresponds to the well-known Communicating Sequential Process (CSP) model [203]. The major advantages of the CSP approach is the possibility of using hierarchies and defining segments of a complex protocols independently from each other (processes composition in CSP). This semantic form proved to be more intuitive to the physicians as well, because it is closer to the way they think of the different sub-problems.

### **CPML vocabulary**

There are two very different approaches to specifying vocabulary for DSMLs. In many engineering communities, the selection of terms for DSMLs is frequently considered as a large informal and ad-hoc task, where language designers have a lot of “freedom”. This view is justified by the perception that the semantics that really matter are behavioral (i.e. dynamic). Behavioral semantics is usually defined transformationally, so the vocabulary selected for constructing a DSML does not matter (it is basically just “syntactic sugar”). The primary technology used for defining modeling languages is metamodeling. Informatics communities (e.g. those working on Web service technologies) follow a remarkably different interpretation of semantics. They consider the use of different vocabularies as the primary source of semantic heterogeneity and identify semantic integration with mapping and translating terms across different communities. The technology background for developing sharable models is known as an

ontology. Ontologies provide “a formal, explicit specification of shared conceptualization” [204], therefore the emphasis in semantics is not behavior, but conceptualization.

STEEP is designed for and integrated into a clinical environment where both behavior and conceptualization are essential. As a patient management system, STEEP needs to model complex, concurrent treatment processes that receive and generate events and trace complex treatment trajectories. As a CIS, STEEP needs to be integrated with other major CISs, where shared conceptualization is essential. These considerations have led us to structure the CPML vocabulary into three major conceptual categories shown in Table 9:

- The **Protocol Modeling** category includes concepts that model the treatment processes. These concepts will be structured into CPML sublanguages that focus on behavior (see the Protocol modeling section).
- The **Medical Knowledge Modeling** category includes information models, typically described as ontologies.
- The **Model Management and Support** category includes modeling concepts that further extend models captured by the previous two categories (e.g. meta information, GUI configuration) required for.

A complete description of CPML is beyond the scope of this thesis; however, in this section we present the abstractions of the language<sup>83</sup> that are important from the point of modeling the sepsis guideline (see Table 9).

---

<sup>83</sup> Because CPML is under continuous development, from here on we refer to version 3.12, if it is otherwise not stated. Version 3.12 is the one being used for both the currently running STEEP environment and the analysis environment presented in the following sections.

Table 9 - CPML vocabulary

Category	Abstraction	Description
Medical Knowledge Modeling	Medical Library	Components of the medical library serve as a knowledge base for the rest of the language. It includes medical vocabularies for diagnoses, symptoms, conditions, vital signs, laboratory test values, medications, procedures and non-medical actions. Concepts in the Medical Library are used to create Orderables that are associated with Activities in the modeled Protocols.
	Orderables	Orderables implement both simple and complex medical actions that are scheduled as Activities in the Protocol. Orderables form a shared vocabulary with the CPOE.
Protocol Modeling	Activity	Activities represent atomic actions in a Protocol. A physician initiates these items during the treatment process. Activities are tied back to items defined in the Medical Knowledge vocabularies with the help of Orderables.
	Protocol	Protocols are parameterized care plans involving patient and other monitored parameters, constraints, explicit event and data-based coordination, activities and metadata (e.g. reference to clinical sources, versioning, authors).
	Process	A process represents a coordinated group of activities used in Protocols. They help to decompose the treatment protocol and to categorize the treatment steps. They are concurrent, asynchronous and can interact with each other via Events.
	Event	Events are components used in Processes. Events refer to the significant status changes (e.g. activation, starting and completion) of executable components, such as Protocols, Processes and Activities. They help to establish coordination by creating dependencies among the mentioned runnable components.
	Explicit Coordination Primitives	Coordination primitives help in expressing the desired behavior. As opposed to implicit coordination (e.g. constraints), explicit coordination primitives define rules that will determine the flow of the execution. Explicit coordination primitives include the Activation and Step connections, control flow operators (such as Branch, Fork and Merge) and logical operators (such as ActivitySelectionSet and LayeredSelectionSet).
	Expression Language	The Expression Language provides a definition for calculating the value of expressions without causing any side effects, and provides methods for constructing derived data points with filtering and aggregation using data values and logical operators defined by the (expression) language. In addition, it implements an implicit request for information exchange (i.e. external data points represented in expressions will be requested by the engine). Finally, it is used to express various constraints (i.e. implicit process coordination), including conditions for goals, failure, and priorities over alternative treatment options.  Our expression language is defined with the help of ANTLR [205], which specifies context-free grammars expressed using extended Backus-Naur form (EBNF) [206]. In defined expressions, operations are identified by ANTLR's built-in parser and are mapped to functions implemented in by Java. Further discussion of our expression language is out of the scope of this thesis.
Model Management and Support	Metadata	Metadata refers to meta information related to protocols and their building blocks.
	GUI Configuration	Constructs of the GUI Configuration provides the ability to control how certain components are represented on the STEEP GUI.
	Physical Quantity	Physical Quantities were only introduced in a later version of the language. They define physical entities, including basic (e.g. mass and length), derived (e.g. surface area), and enumerated (e.g. gender). They are captured as an ontology of physical units extended with a system of unit conversation equations.

## Medical knowledge modeling

Medical Knowledge is represented as ontologies shared across CIS in the institution. Its content can be automatically imported from standardized data sources, or manually created. Integrability of STEEP demanded that we adopt existing ontologies to the fullest extent or, if not exist, model them such that they can be reused in other CIS developments. To achieve this, we studied and adopted parts of the UMLS [153] and standards (such as SNOMED-CT and FDB) and built mappings between constructs used in VUMC CISs (Core Cache, TDQ and Horizon Expert Order) using their medication notation (Chisl, Svc and Portobello codes).

Ontologies of Medical Knowledge are implemented in GME as model libraries. This approach is essential, since the vocabulary terms in these libraries need to be updated on a regular basis and had to be referenced in the protocol models. To make the implementation feasible, we defined a language for these libraries as GME metamodels and created the model libraries first manually (by recreating the structure of required elements) then later automatically (by importing their content from the used standards). Since the overall size of these libraries is exceedingly large, we describe only the basic components of these model libraries without providing all details.

### Medical Library

The medical library is the lowest layer of medical knowledge modeling, its components serve as a knowledge base for the rest of the language. Its main components, which can be organized into a taxonomy using an arbitrary layer of grouping elements (as seen on Figure 35), allow the definition of medical terms that describe the status of a patient (diagnosis, symptom, condition, vital sign and lab value), and action items (both non-medical and medical ones, such as medication and procedure).

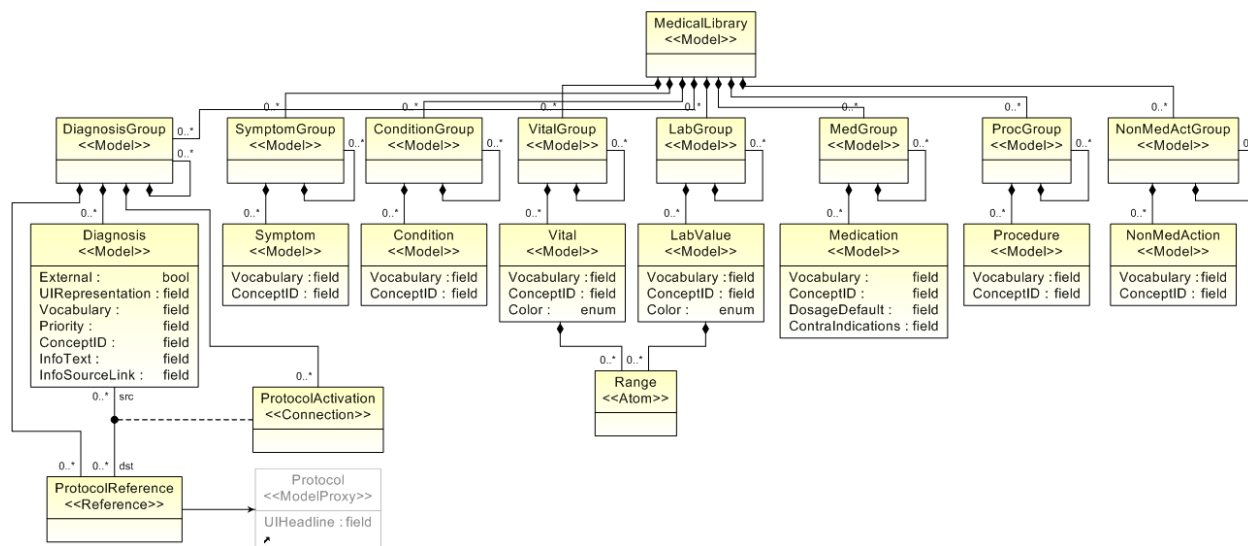


Figure 35 - Main concepts of the CPML's Medical Library (represented with MetaGME)

## Orderables

Orderables define executable (medical) actions that are specific to a healthcare organization; they provide means for building bundles that are available for healthcare professionals at a given HCO. Orderables are items that Activities of a Protocol refer to.

Finding the right abstraction layer defining orderables was a challenge in CPML. On one hand, the goal of orderables is to provide definitions for the simplest executable (i.e. atomic) actions. On the other hand, they need to allow the modelers to concentrate on the logic of the guideline, without having to deal with many of the complex details of a complete order on a per order basis, including dosing, timing, and compliance with safety and privacy policies. To resolve this dichotomy, we examined how current ordering systems approached this problem.

The *order sentence* is the simplest representation of intended plans in CPOE systems and the dominant information model used to communicate order specifications sufficient for downstream systems to execute an order. Medication order sentences for example, include information regarding the drug name, dose, units, route and frequency of administration.

*Order sets* are a more complex form of plan representation in CPOE systems. Order sets are preconfigured groupings of order sentences for plans of short duration (hours to days) related to a particular medical problem (e.g. chest pain admission order set) or treatment protocol (e.g. CHOP chemotherapy order set). Order sets provide decision support for implementing clinical practice



guidelines and institutional policies [207]. Order sets typically group order sentences by type (e.g. lab order, medication order), and may have simple temporal constraints and conditional logic. Order sets serve as a starting point template that is further customized to account for variance in patient state. While order sets are short duration plans, they may be reused for subsequent episodes of care where orders are repeated. This is a typical implementation for many CPOE systems supporting recurrence (e.g. repeating cycles of chemotherapy protocols). Even though order sets provide a richer form for representing orders than order sentences, they have limited support for the representation of complex plans especially ones with longer duration. Moreover, once the order sentence instances of an order set are released into the transaction systems of a CIS, they generally lose their connection to the original order set template and to each other<sup>84</sup>. Again, this makes it difficult for downstream systems and healthcare professionals to understand the context of the orders that comprise a complex plan. Finally, the inability to reuse knowledge in order set templates makes knowledge maintenance a significant challenge in the face of a large number of existing order sets (e.g. 1200 standard of care order sets used in chemotherapy [208][209]). Likewise, validation of the generated order sets is also limited to verbal feedback from expert clinician users, increasing the risk of errors for edge cases.

We resolved this dichotomy in CPML by allowing users to represent simplified order sets as orderables that provide the means for building bundles (i.e. sets) of order sentences. Orderables include bundles of procedures, medications, and lab tests (as seen on Figure 36). At the highest level, similarly to one found in the Medical Library, a grouping layer (e.g. LabBundleGroup) allows for the organization of built bundles into a taxonomy. Components of bundles, the order sentences are defined by creating a mapping to respective concepts in the medical library and extending them with identifiers (Vocabulary and ConceptID) from proprietary or standardized vocabularies (e.g. UMLS, SNOMED, FDB). The first mapping helps define the used concepts with the local ontology, the Medical Library, which could be used for example to find substitutes for an action. The purpose of the second one is to identify order sentences in the CPOE that are requested by STEEP. The same external vocabulary-based mapping is provided for order sets as well. This provides a flexible approach when integrating with various CISs, as it allows the highest level of action (or action group) available in the serving CPOE to be requested by implemented CIGs.

---

<sup>84</sup> See “black box” method in P2.3 point of “Open problems” section.

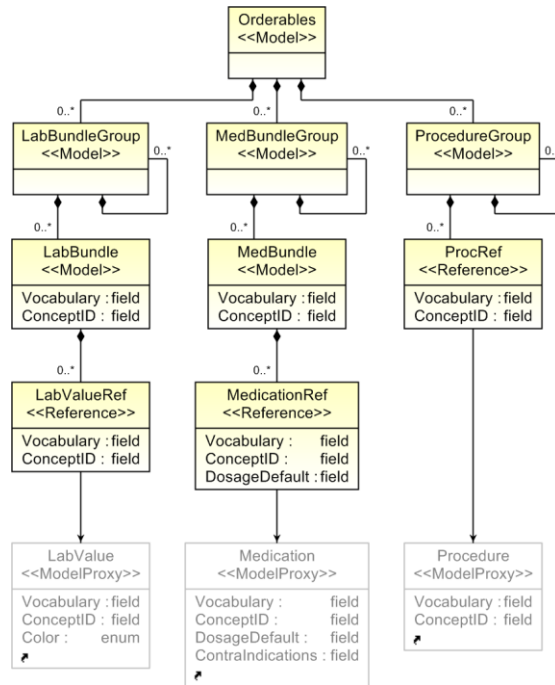


Figure 36 - Main concepts of the CPML's Orderables (represented with MetaGME)

### Protocol modeling

Protocols can be conceptualized as a coordination layer over medical activities. As such, the essential semantics of protocols is behavioral, while the essential semantics for medical knowledge was ontological. The connection between the coordination layer and medical knowledge layer is established by introducing the abstract concept of Activity in protocol models and providing language facilities for relating Activities with terms coming from Medical Library and Orderables.

Protocol models are representations of treatment processes (called Processes). These processes are concurrent, asynchronous, and interact with each other via events. To capture the decision logic concisely, we organized processes in a hierarchical manner. Processes can listen to events happening around them and start running only if their triggering conditions are satisfied. Processes are coordinated with the help of events and related messages. The main concepts in CPML for protocol modeling are summarized in Figure 37 and in the following description.

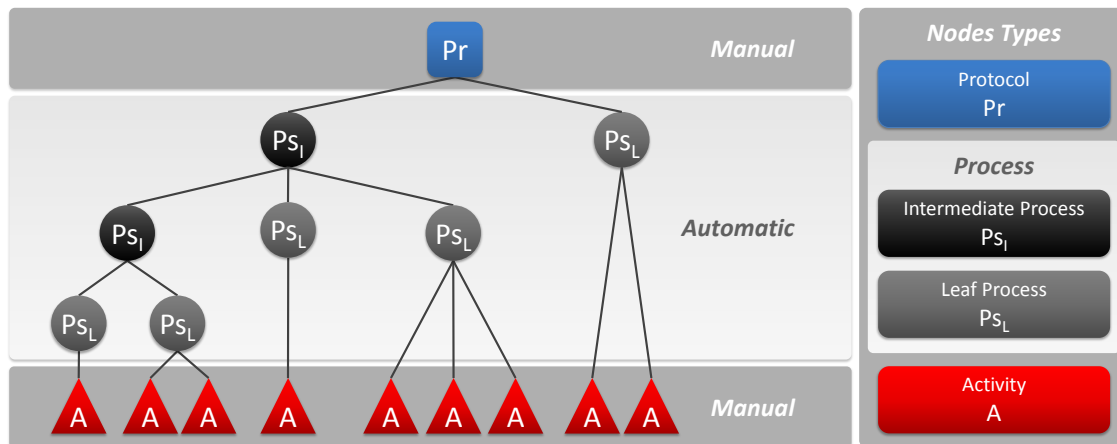


Figure 37 - Example Protocol hierarchy

- Protocol:** Protocol is the top-level concept in protocol modeling. It includes a complex group of coordinated activities required for managing a health problem. Protocols are considered “manual”, as they only start their execution if a user explicitly selects them.
- Process:** Processes represent a coordinated group of activities used in Protocol models. Processes help decompose the treatment protocol and organize the treatment steps. A Process can be considered as a container including other protocols, other processes, events and activities. Processes are concurrent and asynchronous, and they can interact with each other via Events. For the sake of clarity, we require processes to either only contain other processes or protocol invocations (Intermediate Process), or to only contain Activities (Leaf Process). Processes are considered “automatic”, as their execution is only controlled by the EE.
- Activity:** Activities are the lowest-level components of a Protocol. They are the representation of what medical actions must be performed at a given time as part of the treatment. Activities include ordering lab bundles, medication bundles, and procedures. They can also include two other basic actions: Inquiries and Notifications. An Inquiry defines an explicit data request, for data usually not available in EMRs (e.g. symptoms and case severity), which will be presented to the clinicians in the form of a pop-up question. Notifications represent explicit message requests to systems other than the CPOE.

Figure 38 visualizes the hierarchical structure formed by the Protocol, Process and Activity concepts for the Sepsis treatment protocol. The leaves of the tree are the medical Activities, the root of the tree is the Protocol and between them are the Processes that perform the overall coordination using the Coordination primitives described below (not shown in the figure).

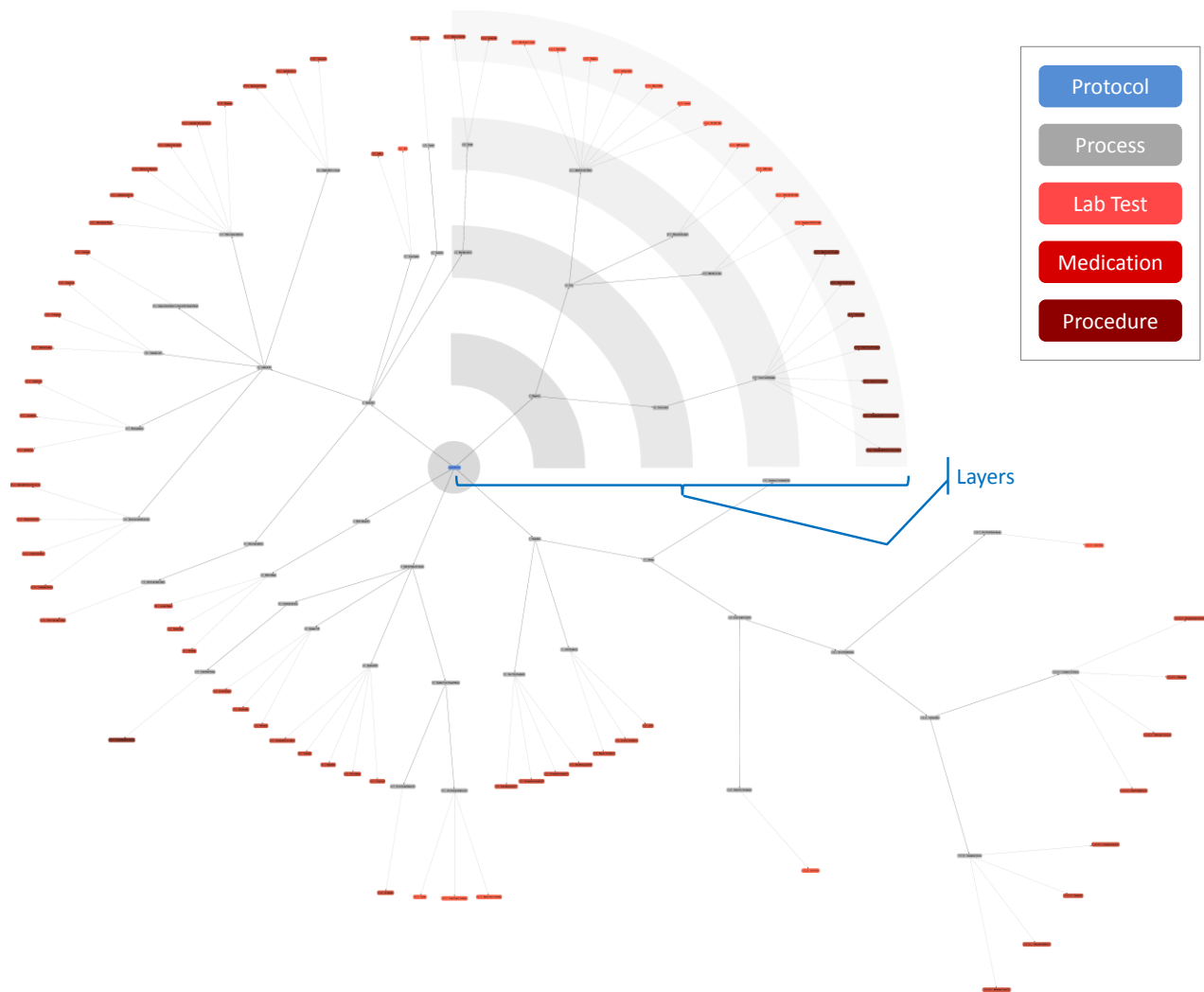


Figure 38 - Sepsis treatment CIG component hierarchy in CMPL

### Explicit coordination primitives

Modeling nondeterminism in treatment protocols is a highly desirable feature. It helps decrease the “recipe-style” appearance of the protocol recommendations and provides methods for adapting the treatment trajectory to unforeseen and unmodeled situations.

Coordination primitives help in expressing the desired behavior, for both deterministic and nondeterministic cases. As opposed to implicit coordination (e.g. constraints expressed with the help of the Expression Language), explicit coordination primitives presented in this section, define rules that will determine the flow of the execution among executable components (Activities, Protocols and Processes). Explicit coordination primitives include:

- **Step:** Step is a coordination primitive, captured as a connection that specifies the execution order (i.e. sequencing) of Activities within a Process.
  - **Operator:** Operators can be split to the following two groups:
    - **Control flow operators:** These operators used together with the Step connection and the executable components describe a TNM, which is essentially a workflow. We implemented the following three basic operators that proved to be sufficiently expressive for representing the sepsis guideline, however if needed, this list can be extended with minimal effort:
      - **Branch:** Branch implements *parallel split*, which is “the divergence of a branch into two or more parallel branches each of which execute concurrently”, as defined in [162].
      - **Fork:** Fork implements *exclusive choice*, which is “the divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on a mechanism that can select one of the outgoing branches”, as defined in [162].
      - **Merge:** Merge implements *synchronization*, which is “the convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled.”, as defined in [162].
    - **Logical operators:** The explicit logical operators we implemented in CPML are set operators **ActivitySelectionSet** and **LayeredSelectionSet**. These operators enable capturing complex choices by defining the admissible sets of treatment choices (i.e. solutions) using contained protocols, processes and activities in processes. In other words, solutions identify the acceptable permutations within the superset of contained executable components. For example, they can implement “pick at least two out of a predefined list of antibiotics”. If there are no logical operators defined in a Process, the default solution is the set containing all subcomponents.

- **Activation:** Activation is a coordination primitive, captured as a connection between events generated by executable components and Protocols or Processes. This association between events and executable components allow for an event-based coordination.

### **Model management, support and configuration**

This category includes the following three subgroups:

- **Metadata:** Metadata refers to meta information related to protocols and their building blocks. In the version of CPML discussed here, it only includes simple versioning. Later versions however were extended with capability to represent roles in the context of an organizational structure. Roles include actors participating in the plan creation process (e.g. author, reviewer), and in the plan execution process (e.g. nurse, attending, resident).
- **GUI Configuration:** The STEEP GUI in most part is configured implicitly, which means that certain model concepts (and their execution) will determine what will be shown to the users. Figure 39 illustrates how the top five Processes and a group of Activities is manifested in the GUI. On the other hand, with the help of the abstractions of GUI Configuration CPML provides dedicated constructs for explicitly controlling how certain components are represented on the STEEP GUI. They include elements such as ordering of the elements (Priorities), coloring options for graphed elements (normal and abnormal Ranges), and text-based descriptions regarding the use of certain components (Help and References that link to external sources).

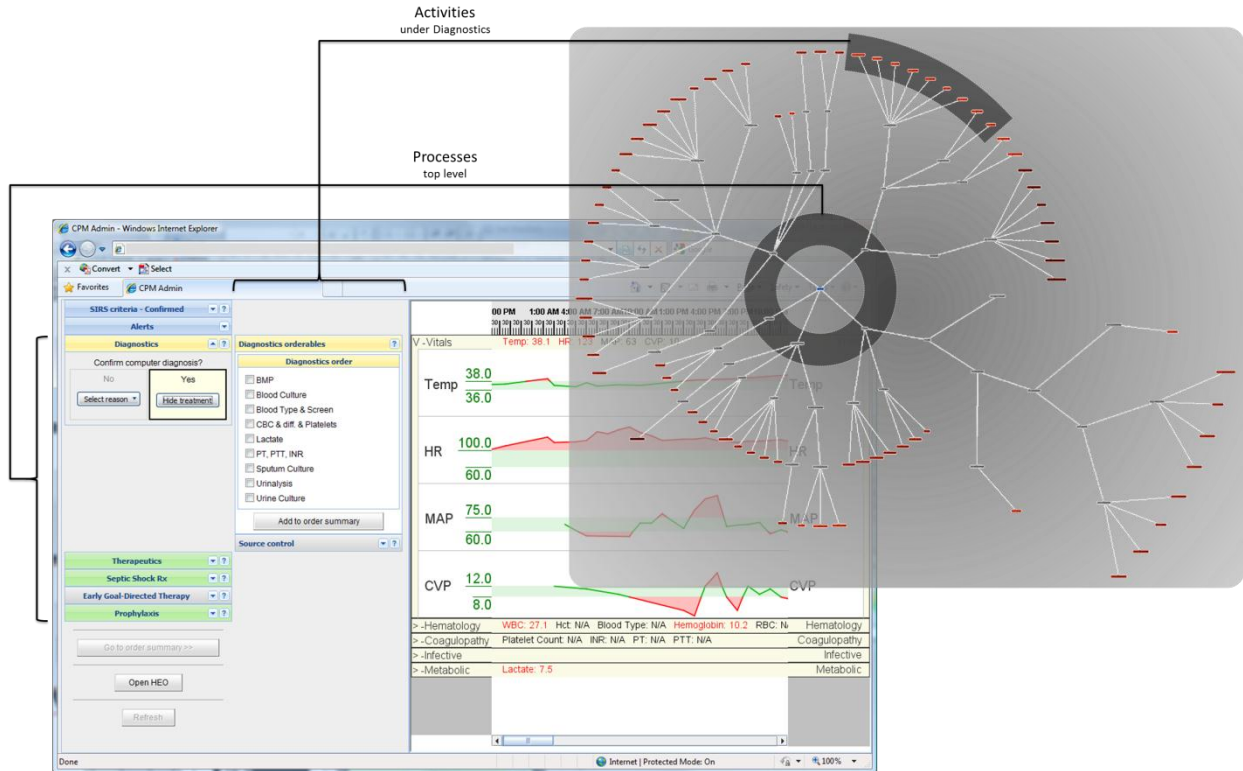


Figure 39 - Model-based configuration of the STEEP GUI

### CPML metamodels - Structural semantics

The second step in the design of CPML is the specification of the  $D_{CPML}(\gamma, C)$  domain for the modeling language, in other words, defining its structural semantics. In MIC there are two supported methods for defining structural semantics:

- **MetaGME-based metamodeling** uses the MetaGME metamodeling language [210]. MetaGME includes a variant of UML class diagrams as type language and the Object Constraint Language (OCL) as constraint language.
- **Logic-based metamodeling** uses the FORMULA (Formal Modeling Using Logic Programming and Analysis) [74,201] that used non-recursive Horn logic, for deciding well-formedness or mal-formedness of model instances. In this method, MetaGME models are translated into formal metamodels.

Since we intended to use the metaprogrammable components of the MIC tool suite (such as GME and UDM) we chose to use the MetaGME-based path.

In this section, we discuss the metamodel of the CPML sublanguage designed for Protocol Modeling.

## Relationship of Protocols, Processes and Activities

As described above, the primary concepts used for coordinating medical activities in CPML are Protocols, Processes and Activities. The metamodel segments linking these concepts are shown in Figure 40, Figure 41 and Figure 42. These figures contain many abstract<sup>85</sup> elements, which help simplify language design and automatic model interpretation by allowing the organization of concrete components.

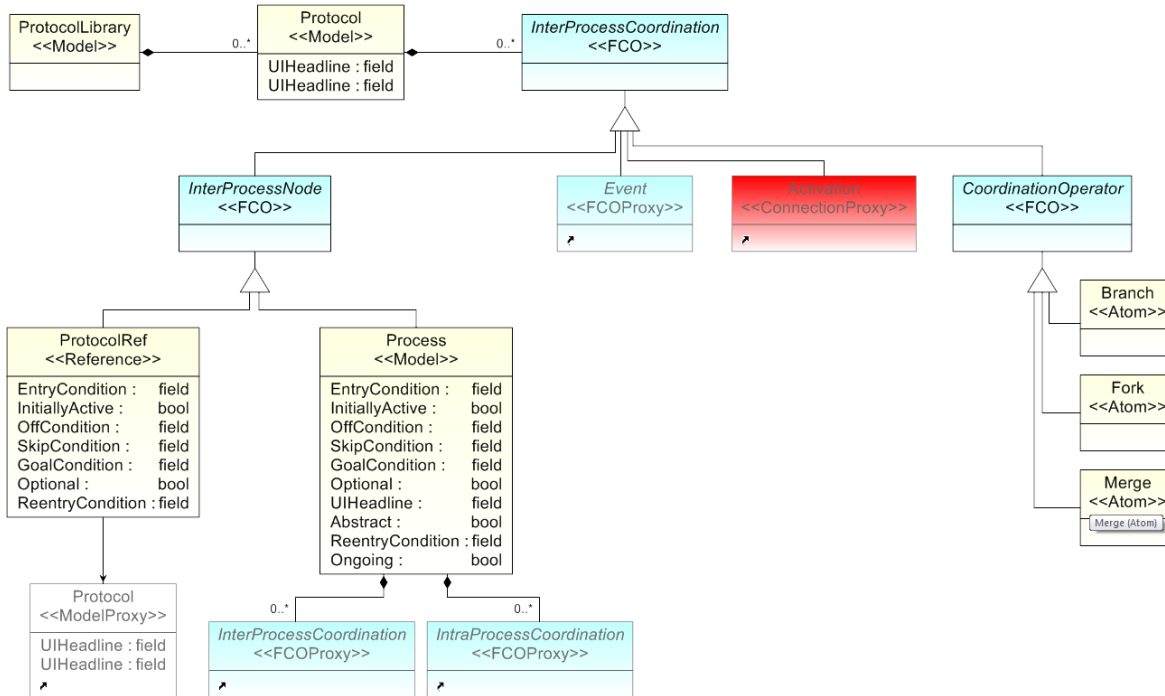


Figure 40 - Protocol Modeling in CPML: Protocol, Process (represented with MetaGME)

The metamodel in Figure 40 defines ProtocolLibrary as a container of Protocols. Protocols can be built up by a combination of references (i.e. pointers) to other Protocols, as well as Processes, Events, Activations and CoordinationOperators (i.e. control flow operators). Missing definition for the definition of Events and Activation connections are provided by Figure 41. As an example, it defines possible events for an activity as start, order and complete.

<sup>85</sup> Abstract components do not show up as available elements in the language. Typically, they are used for grouping concrete elements in the language definition that do. Abstract elements in MetaGME are represented with either italicized font, or with the <<FCO>> keyword.



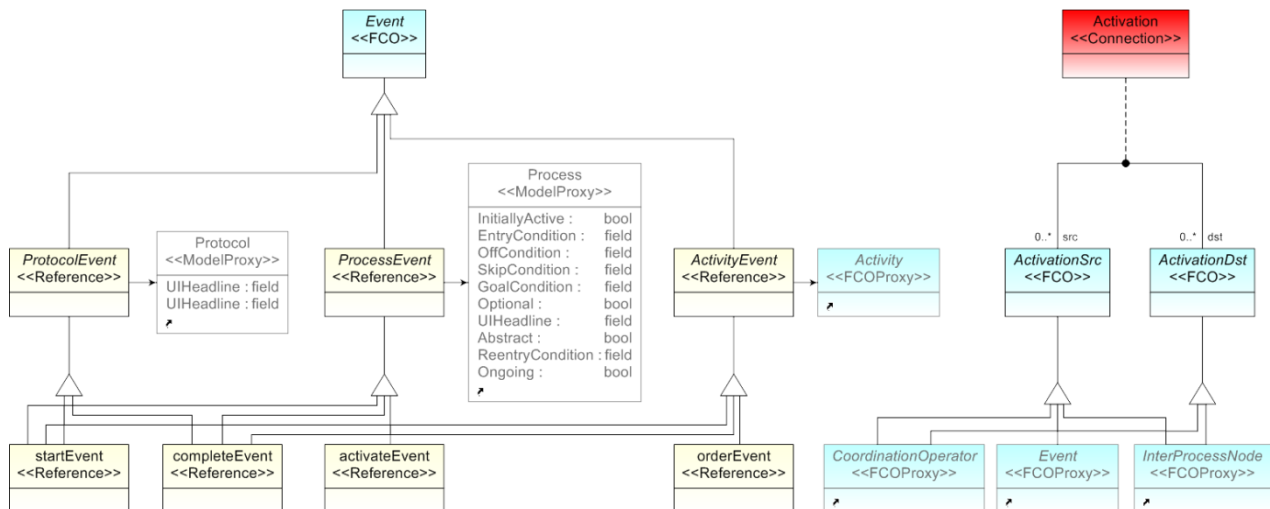


Figure 41 - Protocol Modeling in CPML: Event, Activation (represented with MetaGME)

Figure 42 describes Activities as Notifications, Inquiries, or references to Orderables, namely procedures (ProcedureRef), medication bundles (MedBundleRef), or laboratory test bundles (LabBundleRef). Furthermore, it also explains how solutions can be constructed out of Activities using ActivitySelectionSets and LayeredSelectionSets.

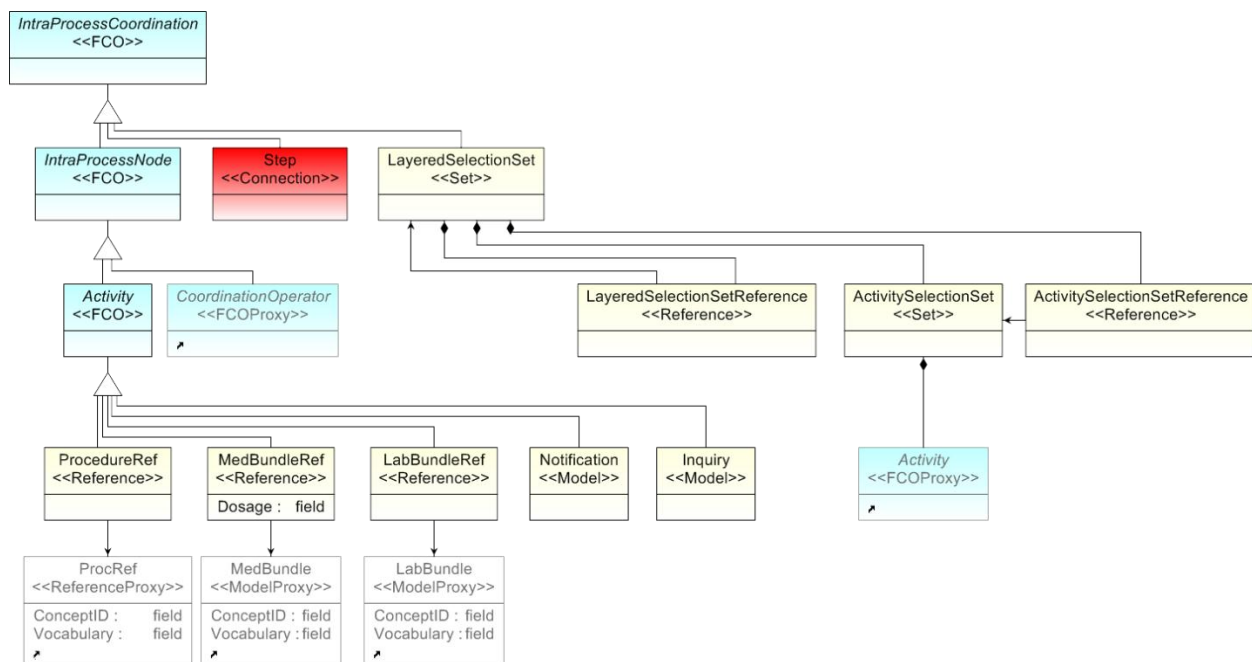


Figure 42 - Protocol Modeling in CPML: Activity, Step, Selection Set (represented with MetaGME)

In order to further specify the well-formedness rules defined by the previous diagrams, we extend the language definitions with OCL constraints. By tying these constraints to various events monitored by GME, models can be checked for correctness automatically. Three short examples are provided below:

### *Example 1*

This constraint, associated with connections, restricts users from creating connections, where the source and the destination are the same (i.e. only connections between two different items are allowed):

```
let src = self.connectionPoint("src").target() in
let dst = self.connectionPoint("dst").target() in
src <> dst
```

### *Example 2*

Language definitions often contain constructs that are references to objects, which allows users to creating a conceptual link to some already defined element in the model (e.g. Orderables are references to elements of the Medical Library). GME normally allows the creation of references in the model that do not point to anything (i.e. empty references). We, however, wanted users to be able to check for these empty references, as in CPML those are considered errors. This can be done with the help of the following code:

```
let RefSet = self.referenceParts() in
let NotEmptyRefSet = RefSet->notEmpty() in
if NotEmptyRefSet then RefSet->forall( not refersTo().isNull() ) else true
endif
```

### *Example 3*

In some cases, we require references to only point to “local” constructs, which means that the referrer and the referee has to be in the same container object:

```
not self.refersTo().isNull() implies self.refersTo().parent() = self.parent()
```

## **Coordination primitives**

In summary, the coordination of actions (Activities) in CPML is performed by layers of Processes inside of Protocols. These layers help in identifying and decomposing sub-problems. Once the proper decomposition is achieved, selection sets defined over the components of each Process help identify acceptable solutions. The solutions at the lowest level are TNMs built using the Step connection and the control flow operators that serialize the set of Activities contained in the Process. Event-based coordination of Processes is achieved by connecting monitored triggering events (Events) and Processes (or ProtocolRefs) with the Activation connection.

## Review of a model example

According to the well-formedness rules defined by the metamodel, we present a simple exert from the sepsis CIG (see Figure 43).

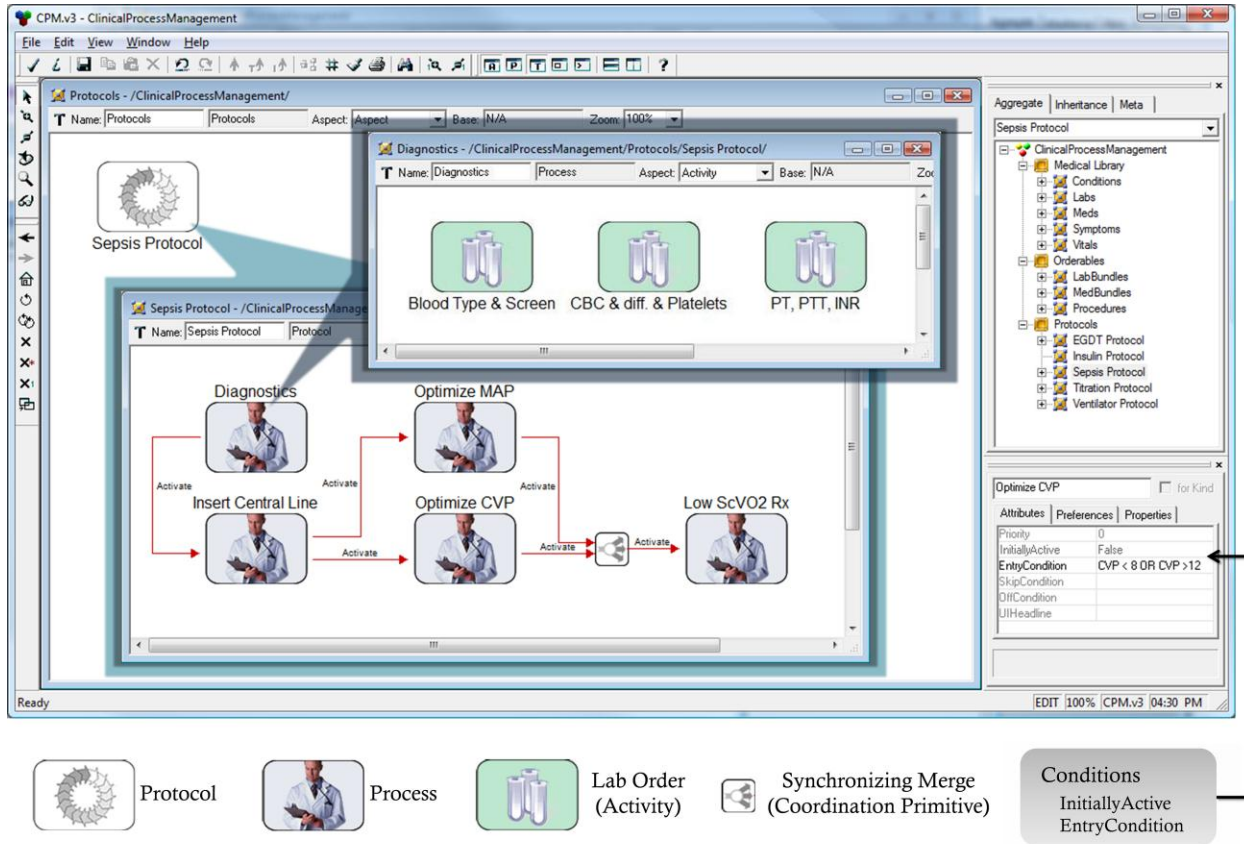


Figure 43 - Exert from the sepsis CIG (in CPML)

The example in Figure 43 explains how our domain experts implemented the initially<sup>86</sup> used MUST protocols (seen in Figure 33). In the figure, GME is configured with CPML. The main window, called *Protocols* shows one implemented Protocol, the *Sepsis Protocol*, which contains five processes, namely *Diagnostics*, *Insert Central Line*, *Optimize MAP*, *Optimize CVP* and *Low ScVO2 Rx*. These Processes are orchestrated with the help activations, which have no direct control over the execution order of the processes; it just constrains the order by specifying when the components start to listen. The execution

<sup>86</sup> While this is a good illustrative example, since the initial modeling, the VUMC sepsis CIG has changed in order to accommodate new requirements.

order is not determined until runtime, when STEEP can evaluate the entry conditions for the processes. The resulting execution behavior is as follows:

1. A user initiates the sepsis protocol.
2. Out of all the contained processes, all initially active, in this case only the *Diagnostics*, will become active. Because the skip condition of the *Diagnostics* process is not defined, it will evaluate its entry condition, which being empty, will allow the processes immediate execution.
3. The contents of *Diagnostics*, the three laboratory tests displayed on the figure, are in no particular order (i.e. no dependencies exist among them). This and the fact that there are no solution sets are included means that the set of all three tests define the only solution for the process. Accordingly, as the initial (and only) step, all of them will be initiated simultaneously and, as a result, will be recommended to the treating physician.
4. Assuming the physician accepts the recommendations and submits the orders to the CPOE, the *Diagnostics* process will complete its execution, and activate the next process, the central line insertion.
5. After the completion of the *Insert Central Line* process, two independent processes, the one for mean arterial pressure and the one for central venous pressure optimization become active. They start executing if their respective thresholds (defined by the entry condition) are met (the example shows bounding values for CVP). Otherwise processes are skipped.
6. Finally, after both problems have been addressed (expressed with the synchronizing merge), the *Low ScVO2 Rx* process is activated.
7. If all processes complete, the sepsis protocol finishes its execution.

## CHAPTER VII.

### DEFINING THE BEHAVIORAL SEMANTICS FOR CPML

The execution of CPML models determines the behavior of STEEP. The specification of the CPML language via its structural semantics, as described in the previous chapter, does not define how CPML models are translated into behavior. This is clearly an important missing point, since treatment management is manifested as a behavior: a coordinated sequence of actions and interactions. Understanding if the behavioral traces defined by the CPML models are safe, do not lead to deadlocks or nondeterministic behaviors, and satisfy invariants such as deconfliction of mutually exclusive treatments are very important. Answers cannot be found to these questions without understanding how CPML is translated into behavior.

As shown by the STEEP architecture (Figure 30), the STEEP Protocol Execution Engine interprets the CPML models and implements the model-to-behavior translation in the implemented system. While the source code of the engine provides a sufficient definition of the behavioral semantics, using this description for understanding and analyzing behavior is suboptimal. This is because the execution environment is a complex Java and web code that implements all the operationally required features of STEEP, including user interfaces, resource management, exception handling, server management and communication protocols with the connected systems. All of these activities result in a significant amount of “accidental complexities” that go well beyond the complexity of behaviors of the treatment protocol. What is needed is an abstract specification of the CPML behavioral semantics that is analyzable and captures the model-to-behavior translation without the accidental complexities of the implementation. The scientific challenge here is the development of a method that makes the specification of the behavioral semantics of CPML explicit and usable for analysis. Our selected approach is semantic anchoring, which defines behavioral semantics by specifying the transformation between CPML and a target language with a well-defined behavioral semantics.

In this chapter, we first discuss the concept of the transformational specification of semantics. We then describe a template-based specification of model transformations and demonstrate the semantic anchoring of CPML to the Stateflow modeling language.

### Specification of behavioral semantics

As described Chapter VI, CPML is defined as a 4-tuple, comprised of its domain, and a set of interpretations:

$$L = (\gamma, R_\gamma, C, (\llbracket \cdot \rrbracket_i)_{i \in I})$$
$$\llbracket \cdot \rrbracket_i: R_\gamma \rightarrow R_{\gamma'}$$

Up to this point, we have discussed the conceptualization yielding specification for  $\gamma$  and the specification of the CPML domain  $D_{CPML}(\gamma, C)$  (restricting our attention to the protocol modeling sublanguage). The last step in the modeling language development is the specification of behavior that can be described by the language constructs: specification of the behavioral semantics of CPML.

As shown in the functional and implementation architecture of STEEP (Figure 29 and Figure 30), the system operates in the context of the VUMC's CIS, continuously receives live data streams from patients, interacts with physicians by presenting care decision alternatives and receiving decisions, sends out instructions to execute medical activities (orderables) and receives status reports about their execution and results. All of these behaviors are guided by the CPML models that are continuously interpreted by the STEEP Execution Engine.

It is not surprising that, in light of this complexity, the semantics of CPML could be defined on many different levels of abstractions:

1. **Execution Semantics:** The specification should capture in full detail all behavioral details that the STEEP EE performs under the control of CPML models. The depth of these specifications may be sufficient for generating a full implementation for the EE on an implementation platform.
2. **Protocol Behavioral Semantics:** The specification should be rich enough to understand the abstract treatment processes and the semantics of coordinating medical activities. However, many platform related details (regarding details of messaging protocols, DB persistency interactions, etc.) are abstracted out.
3. **Mathematical Behavioral Semantics:** The goal of the specification is to map the modeling language into a mathematical domain that is rich enough to represent all behavioral categories the system can exhibit. Examples for such mathematical domains are Abstract State Machines [211] and Timed Automata [212], if the semantics is defined operationally, and Trace Algebra [213] and Abstract Algebra [214], if the semantics is defined denotationally.

## A pragmatic approach

Decision about the level of targeted behavioral abstraction is determined by the goal of the semantic specification. Our goal in this thesis has been practical: we define behavioral semantics for the protocol modeling sublanguage with the purpose of establishing a bridge toward existing tools that can be used in the validation and verification of coordination mechanism modeled in CPML. The selected tool is Mathwork's *Matlab Simulink Stateflow*<sup>87</sup> (SF) tool suite [216], because it includes a well-known modeling language with ample of publications on its formal behavioral semantics. It incorporates a well-developed and widely used simulator that can be effectively used for protocol validation, and it also includes a verification tool, *Simulink Design Verifier* (SLDV), which allows us to make steps toward formal verification of protocol properties.

More formally, our research goal is the development of an interpretation for CPML that maps protocol models in the CPML domain to Stateflow models in the SF domain:

$$\llbracket \cdot \rrbracket = R_{\mathcal{Y}_{CPML}} \rightarrow R_{\mathcal{Y}_{SF}}$$

The mapping will be defined and implemented as a model transformation, the semantics of the CPML protocol modeling sublanguage is defined by the transformation and the semantics of SF.

In the remaining part of the section, we discuss the design and implementation of the model transformation, present examples for its use and summarize the overall semantics specifications for CPML.

### Template-based specification of behavioral semantics

As discussed earlier, the protocol models define a coordination layer over medical activities. The primary active component types used in the modeling language are:

1. Protocol
2. Process
3. Activities

---

<sup>87</sup> Matlab Simulink Stateflow is an implementation of statecharts [215], which is an extension of finite state machines with hierarchy, concurrency and broadcasting. Stateflow further extends statecharts (with for example complex types).

These modeling entities exist concurrently, have internal states and interact with each other using coordination primitives. To implement such behavior, we facilitate a template-based specification for the transformation between CPML and SF models. The basic idea is shown in Figure 44, where A) the structure of a general CIG represented in CPML (from Figure 37) serves as a configuration for the transformation that uses predefined behavioral templates for each executable construct. The result of this transformation is B) a generated behavioral model that is defined using the constructs of the target domain.

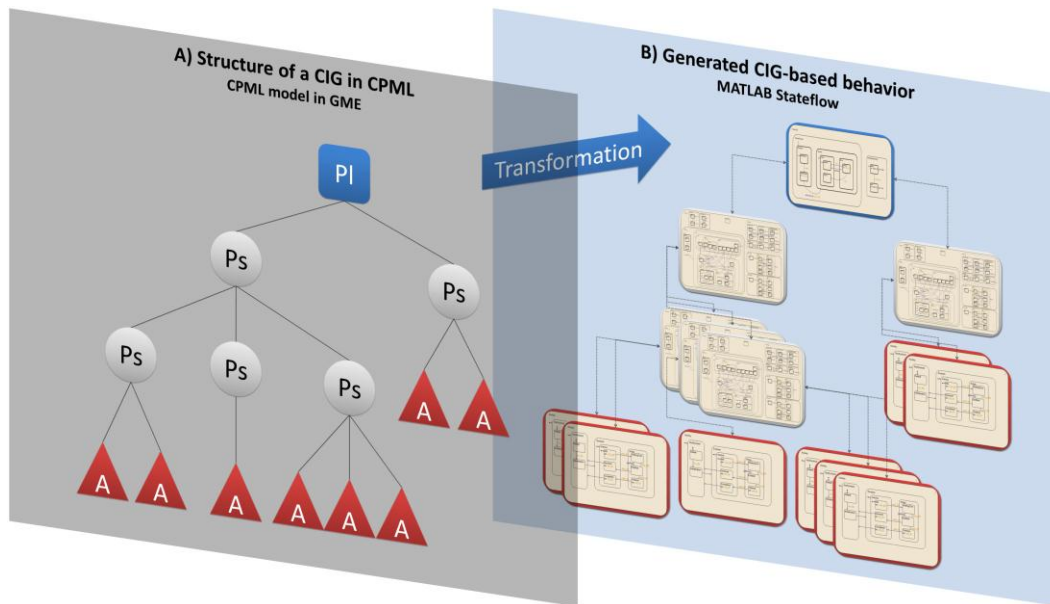


Figure 44 - Generation of SF models from CPML

Because our target domain is SF, Protocols, Processes and Activities are mapped into a hierarchically structured concurrent state machine structure. The state machines communicate via directed event broadcast messages (this restriction was introduced due to interoperability issues with the SLDV). The state transitions in the state machines are governed by complex guard conditions and triggering events that are generated by external and internal events.

### Behavioral templates

The individual state machines are created as instances of the three state machine templates, defined for Protocols, Processes and Activities respectively. The complete documentation of the state machine templates are not part of this thesis, here we explain only the behavior of the Activity template.





The three main parallel states of an Activity are:

- Life: This state represents the actions the Activity will perform during its existence. The first step is a CIG initialization phase (Initialization state), which allows the users to start up the CIG with an external event (UsrIni\_Ev), which propagates down in the containment hierarchy (and arrives at the Activity as Ini\_Ev\_Pald). Afterwards, as part of the Operation state, the Activity will start its life in the Idle state, as part of the Inactive state. If it recognizes that its parent is running (indicated by the Run\_Ev\_Pald event), it will transition to the WaitingTurn state, otherwise at the end of the protocol it will transition to the Finished/NeverActivated state. From being in WaitingTurn the Activity will start running (Running) and get recommend to the user (Recommended) if it is either a starting point of a workflow (i.e. there are no Step connections pointing to it: StrtPoi\_Co\_EnId), or if it receives an explicit message from one of its siblings that precedes it in the workflow to do so (Fin\_Ev\_Sild). Users can defer decision (Deferred state) for a certain time (DfrTO\_NumEnId) upon receiving a recommendation. While a decline (UrsDcl\_Ev\_EnId) puts an Activity into the Declined final state, a confirmation (UsrCfrm\_Ev\_EnId) will lead to adding the recommendation to a “shopping cart”. From here users can order the action (UsrOrd\_Ev\_EnId), which enables a handshake with the CPOE system (CisOrdAck\_Ev\_EnId). This original template contains numerous timeouts for ending the execution, including one for monitoring wait time for starting execution in WaitingTurn (MonTO\_Num\_EnId), for decision from the users upon a recommendation (NoDecTO\_Num\_EnId), for response from the CPOE upon order submission (CisOrdAckTO\_Num\_EnId), and for the activity to finish execution (RunTO\_Num\_EnId).
- Two monitoring states: These states run in parallel to the Life state to be able to react to events and messages independently from the state of Life:
  - ParentFinReqTracking: The goal of this orthogonal state is to monitor the Activity’s parent<sup>89</sup>. This is because when parents find a solution to the problem they are trying to solve, they will send out a Try2Fin\_Ev\_Pald event to indicate that they are trying to finish their execution. This message will inform all of the non-executing children that they are not needed anymore (i.e. not to start their execution).

---

<sup>89</sup> The words, “parent” and “child”, refer to the relationship of concepts in the containment hierarchy.

- OrderCompletionTracking is a monitoring state listening to external messages arriving from the CPOE system. With the help of this state, an Activity will be able to track whether the medical action it is supposed to recommend has been successfully performed within an acceptable time period<sup>90</sup> or not (CisOrdCplTO\_Num\_EnId).

### Template instantiation

At the time of template instantiation, a copy of the template is created. This is followed by the specialization of the template. During specialization, after a simple renaming of the template, both the internal behavior, and the interaction with the environment is set up according to the specification defined by the CIG model.

Our templates were designed to be static in terms of structure<sup>91</sup>, which means that there are no states and transitions added to, or removed from them at generation time. Instead, their specialization involves the use of the following techniques:

- **string replacement:** certain (text-based) components of the template are altered
  - Example 1: the name of the main state, “Activity\_EnId”, is replaced with “Ac\_<Name>\_<ID>”, where “<Name>” will be the name of the Activity in the CPML model, and the “<ID>” will be a unique, hierarchical id that is generated as part of the model transformation process)
  - Example 2: replacement of the triggering event placeholder “Fin\_Ev\_Sild” with the particular event’s id that the Activity needs to monitor before entering the “Running” state
  - Example 3: replacement of all occurrences of “EnId” and “Pald” to the id of the entity and its parent’s id respectively

---

<sup>90</sup> This is often referred to as *validity window*, which indicates the time range in which repeating the medical action is not needed, or allowed.

<sup>91</sup> The static design was possible with a simplification of the original semantic specification: Processes, as opposed to tracking many concurrent possible defined solutions, only track one trivial solution, which is the set of all included components. This limitation could be addressed by generating and embedding multiple solution tracking states instead of only one.

- **variable modification:** variables can be created, removed, or modified
  - Example 4: moving variables that are defined by the template, but implement information exchange between the environment (represented by Simulink) and the CIG logic (represented by SF), or are intended to be globally observable (by all states of the SF) from the level of the template’s main state to the level of the main SF model
- **value replacement:** constants and variables are initialized
  - Example 5: the time out value for being in the “Deferred” state, called “DefTO\_Num\_EnId”, is set to 4 hours

### Transformation process

To enable the template-based specification of behavioral semantics we implemented an analysis software tool chain as part of STEEP. This tool chain, which enables CIG analysis (as described in the next chapter), performs a transformation of the CIG models using the templates (described in the previous section), and an algorithm defined by Algorithm 1.

**Algorithm 1 - Simplified template-based model transformation process**

1:	<b>for all</b> <i>Protocols (contained in the model file)</i> <b>do</b>
2:	<i>record data from the models as template attributes</i>
3:	<i>traverse elements of the Medical Library</i>
4:	<b>for all</b> <i>Vitals</i> <b>do</b> <i>record parameters (as template attributes)</i>
5:	<b>for all</b> <i>Labs</i> <b>do</b> <i>record parameters (as template attributes)</i>
6:	<i>traverse Protocol</i>
7:	<b>for all</b> <i>contained Processes</i> <b>do</b>
8:	<i>traverse Process contents</i>
9:	<b>if contains Processes for all Processes</b> <b>do</b>
10:	<i>traverse Process contents</i>
11:	<b>else ( if contains Activities ) for all Activities</b> <b>do</b>
12:	<i>record Activity parameters (as template attributes)</i>

- 13: *record Process parameters (as template attributes)*
- 
- 14: *record Protocol parameters (as template attributes)*
- 
- 15: *invoke Matlab model generation string template group with attributes recorded at step 2:*
- 
- 16: *replace placeholders in Protocol template with appropriate attribute values*
- 
- 17: **for all** *Protocol inputs (i.e. Labs and Vitals) do replace placeholders in appropriate templates*
- 
- 18: **for all** *Protocol components (i.e. Processes and Activities) do replace placeholders in appropriate templates*
- 
- 19: *run Matlab model generation script (created in step 15:)*
- 
- 20: *create and initialize Matlab Simulink model (to act as the environment)*
- 
- 21: *create and initialize Matlab Stateflow model (to act as the protocol)*
- 
- 22: *add an instance of the Protocol behavioral template to the SF model and configure its parameters*
- 
- 23: **for all** *Vitals do create a data input channel for the SF model (i.e. interface between SF and Simulink)*
- 
- 24: **for all** *Labs do create a data input channel for the SF model (i.e. interface between SF and Simulink)*
- 
- 25: **for all** *Processes do add an instance of the Process behavioral template to the SF model and configure its parameters*
- 
- 26: **for all** *Activities do add an instance of the Activity behavioral template to the SF model and configure its parameters*

This algorithm is a simplified version of the one described in “Appendix C” (see Algorithm 2).

### **Analysis tool chain**

The previously described transformation process is implemented by the analysis tool chain, which translates CPML-based CIGs to Matlab Simulink/Stateflow to enable validation by simulation and verification through formal analysis. The layers and main components of the tool chain are shown in

Figure 46, which provides an implementation oriented view of the software components used in the model transformation process.

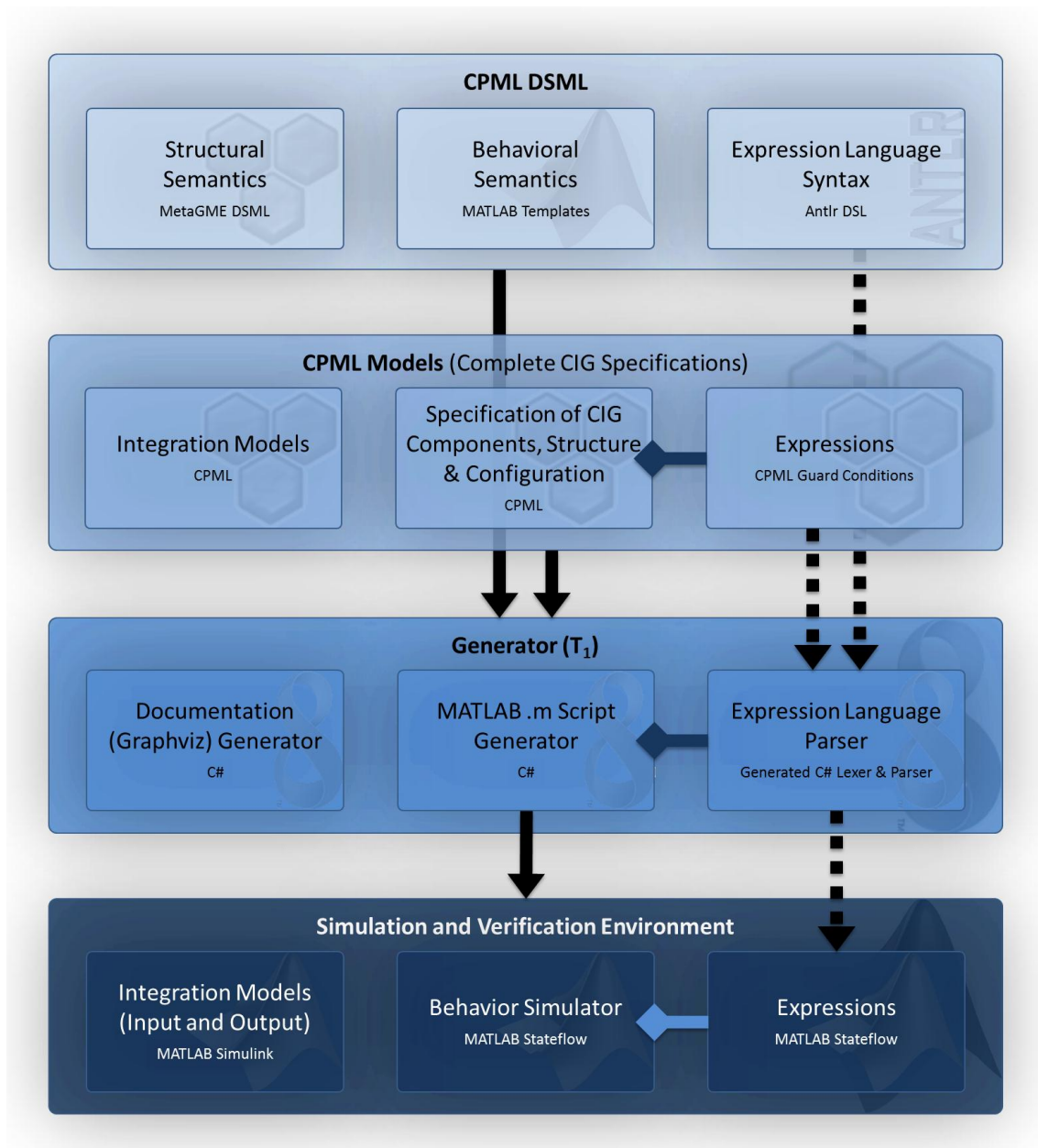


Figure 46 - CPML to Matlab Simulink/Stateflow transformation tool

The top layer is the meta layer, which shows the components used to design the CPML. The language includes a protocol modeling sublanguage for CIG design, an information modeling sublanguage for semantic integration of STEEP into the host CIS environment and language components for the model-

based, automated configuration of the Patient Management Console. The language components include an Expression Language as well, but we have not emphasized its details in the language specifications.

The CPML specification includes the specification of its structural semantics via metamodeling and the specification of its behavioral semantics via developing a translator to the SF language. CPML Models for STEEP consist of protocol models, integration models and expressions defining various decision conditions in the coordination process over medical activities.

The tool chain includes a suite of generators (Generator layer in Figure 46) that translate CPML models into other models and artifacts. As described by the algorithm presented in “Appendix C”, the first generator invoked is the one designed to generate the (XML) input files for the STEEP Execution Engine from the CPML models in GME. This is followed by two generators that use the XML file as input. The first one uses Graphviz [217] to generate a directed graph representation of protocol models for documentation purposes (seen in Figure 38). The second one generates a MATLAB .m script that, when invoked in Matlab, will create the executable Simulink/Stateflow models for simulation and verification purposes. This generator would ideally also be complemented by an Expression Language Parser (Figure 46), one similar to the parser the EE uses to translate expressions used in CPML models to Java. The goal of this future component, which is not yet implemented, is to translate CPML expressions directly to Matlab guard conditions, which is currently done manually. These generators are combined into one application written in C# (see Figure 47).

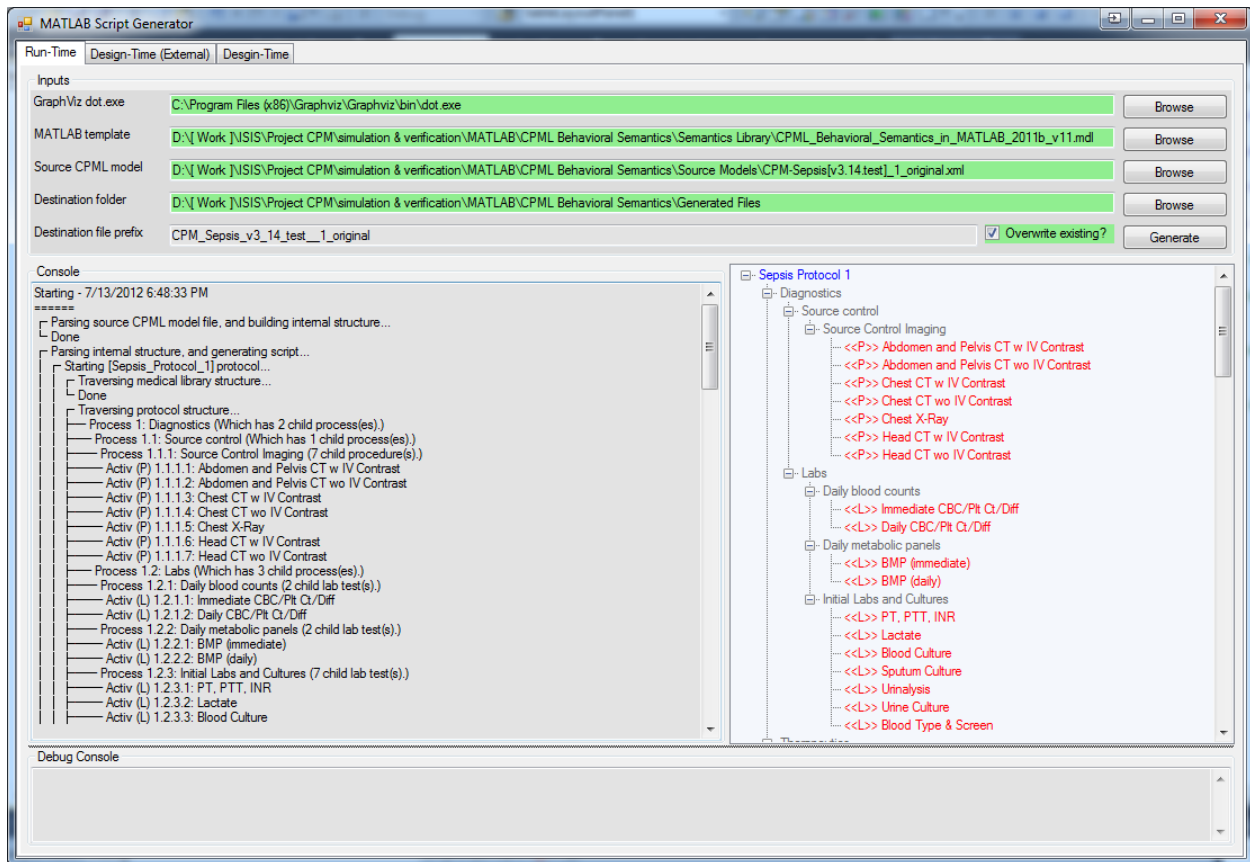


Figure 47 - Screenshot of the Matlab script generator tool

### Example

In the previous chapter, with the help of Figure 39, we showed how the components of the sepsis CIG configure the STEEP GUI. In this section, we use Figure 48 and Figure 49 to present a similar process: the transformation of the sepsis CIG into Matlab behavioral models.



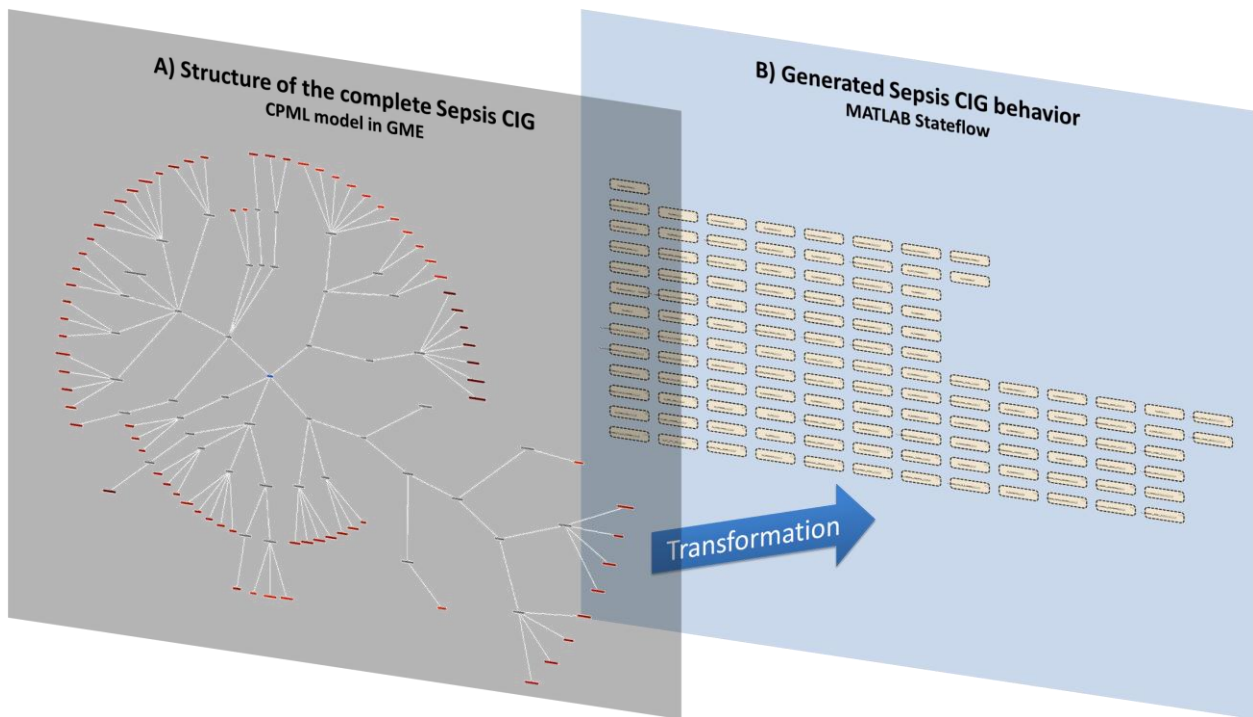


Figure 48 - Generation of SF models from CPML (Sepsis)

Figure 48 is a follow-up to Figure 44. It shows the model-based generation of SF models for the specific example of the sepsis protocol. As a result, there is one template instance created for each executable component (1 Protocol, 44 Processes and 77 Activities). More detail on the resulting SF model is provided by the next diagram.

While Figure 48 only shows the protocol logic implemented in SF, Figure 49 shows the encapsulating Simulink model as well. In Figure 49, the main model can be seen in the upper part of the diagram. All of its components are generated (including the layout), except the “External Events from File” element, which is a link to an Excel file that contains sample patient data for simulation. The SF implementation of the CIG logic is contained by “Protocol\_SF”. Its elements are the (previously mentioned) configured SF template instances that can be seen in the callout bubble<sup>92</sup> on the bottom of the figure, where each sub-charted state encapsulates the appropriate template for a given element of the Protocol-Process-Activity containment hierarchy. Even though it is not visible in this diagram, each state is configured to exchange information only with the appropriate states in SF and components of the environment.

<sup>92</sup> The callout illustration and the colored bands with labels (Protocol, Process and Activity) are not part of the Matlab model.

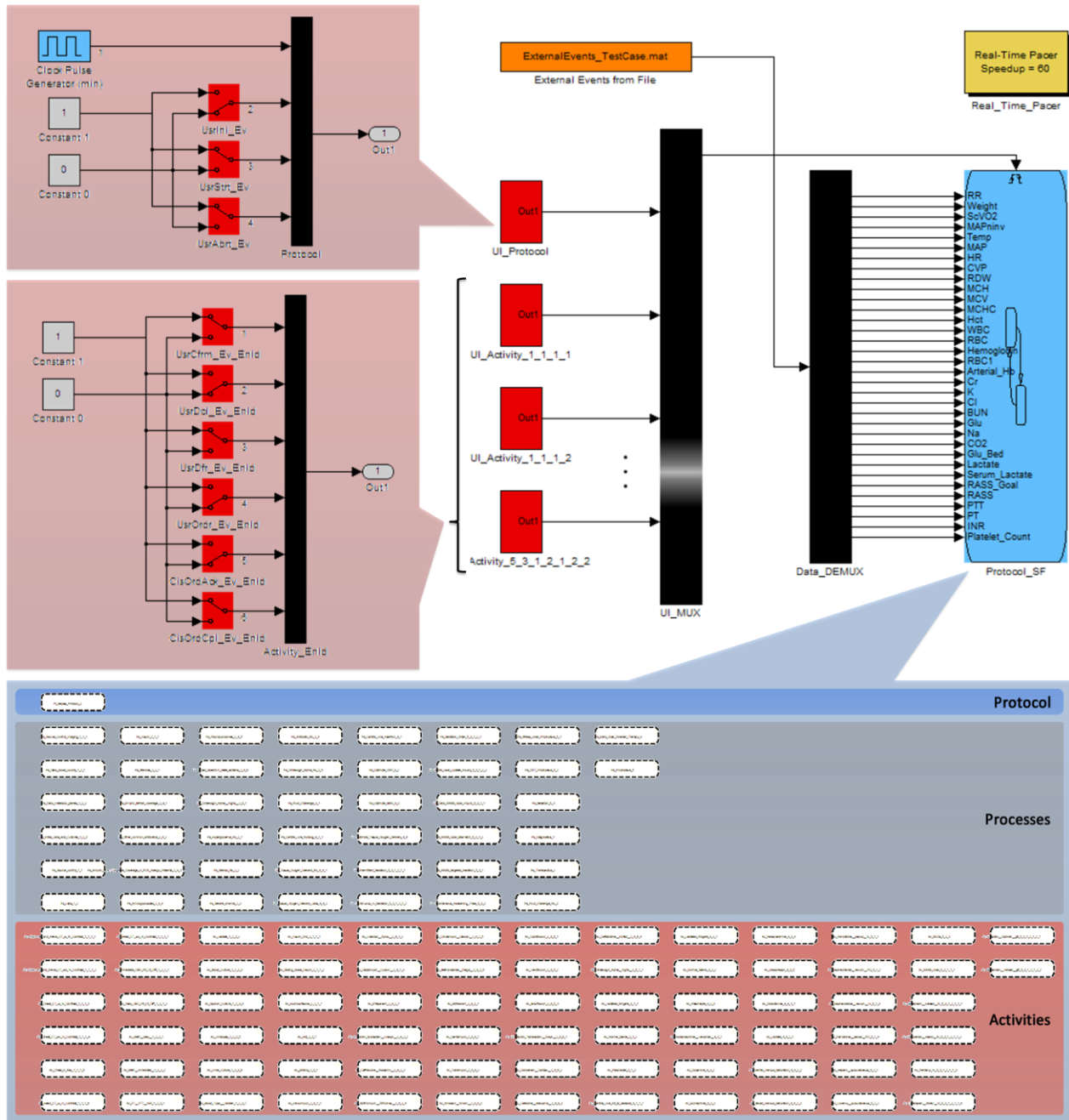


Figure 49 - Generated sepsis CIG in Matlab

The SF chart has both data and event inputs. Data inputs are created for each data item that the protocol relies on to make a decision. In the case of the sepsis protocol, there are 34 numeric data channels, which include, for example, respiratory rate (RR), weigh and central venous O<sub>2</sub> saturation (ScvO<sub>2</sub>). In this particular instance, the generator was configured to create a model for validation of the CIG using real-time user interaction together with sample patient data recorded a priori. Consequently, the generated simulation model, which otherwise includes only components defined by the template

library<sup>93</sup>, requires users to have an input file manually added to it<sup>94</sup>. The interaction between users and the simulation is facilitated by events. Events are triggered by the red switches (labeled as “UsrStrt\_Ev”, “UsrCfrm\_Ev\_EnId”, etc.) of the user controlled abstract GUI elements for the Protocol and the Activities (“UI\_Protocol”, “UI\_Activity\_1\_1\_1\_1”), which can be “pushed” during simulation to indicate a user action (to start the protocol, to confirm a recommended action, etc.). The last component on the diagram is the “Real\_Time\_Pacer”, which allows the simulation time in Matlab to be mapped to relative of real time. In this case, the simulation is sped up to 60 times real time, which results in minutes being mapped to seconds.

The other option for generation is the fully sample-based execution, where the simulation does not run relative to real-time. Instead, both patient data and clinician responses come from a sample scenario file. This is not illustrated here.

---

<sup>93</sup> The template library also defines the format for the input file and the generator creates an empty table with the appropriate header in an Excel file, in order to be able to be connected and provide input during simulation.

<sup>94</sup> By connecting the input to the SF chart through the “Data\_DEMUX” component.

## CHAPTER VIII.

### VALIDATION AND VERIFICATION

Development of STEEP was a lengthy and involved process including a large team of physicians, informatics personnel, HCI designers and computer scientists. One of the most difficult iterative development process was the validation of the CPML modeling language and models via walkthroughs, discussions, tests and later, simulations. Verification of well-formedness properties was provided by the explicit specification of structural semantics of CPML and the built-in constraint checking mechanism of the GME metaprogrammable modeling tool. The need for a simulation-based validation gradually emerged in the project when it was clear that the complexity mandates the use of sophisticated tools. The first validation experiments were completed in a dedicated simulation environment of VUMC and provided early justification for the clinical use of STEEP.

The need for automated verification came with the recognition that the models and the implied behaviors will be too complex for test-based verification. Our approach to automated verification was to anchor the semantics of CPML to Stateflow (see Chapter VII), a modeling language with formally defined semantics and tools supporting automated verifications of state invariants. Then, by relying on a verification toolbox for Stateflow, we showed the feasibility of verifying state invariants in CPML models.

In this Chapter, we provide an overview of the results regarding the established validation and verification methods and opportunities. Our goal has not included the development of a complete tool-based solution, or completing the validation and formal verification of the current release of the STEEP models. As it will be shown, the complexity of a real-life system is quite high and significant further research is needed to gain a scalable solution.

#### Validation through simulation

##### Clinician walkthroughs

Protocol validation tests whether the generated decision-support guidance corresponds to clinicians' expectations. The first step is to model walkthroughs with clinicians. The modeling language's expressiveness is helpful in this process and fully confirms the importance of using DSMLs that are highly customized to the clinical environment. The walkthroughs were performed by the same clinicians who

actively participated in CPML's iterative development (all together four physicians, and one nurse). Out of our regular hour-long weekly meetings, which in average included two physicians, we spent around one third of the time with this type of discussion in the first three years of development. In our experience with many different domains, domain expert involvement in DSML development is an absolute necessity.

During these walkthrough discussions, many components of the language evolved. The first, and probably the greatest, change was the decision to move from a workflow-based representation of the CIGs to a process-based one. Other major changes in terms of language constructs included the creation of solution sets, to create a more sophisticated selection layer over the components of processes, because initially, acceptable solutions only existed as textual recommendations, thus were not precisely enforced. Another major change was the decision to create purely diagnostic processes that allowed modelers to separate a patient condition-based diagnosis step from the associated recommendations. This separation caused clinicians to first manually confirm the diagnosis, before any automatic recommendations were given by the system.

Last, but not least, one of the most difficult problems that these walkthroughs allowed us to solve, was the discussion on how models affected the behavior of the GUI. We spent countless hours trying to figure out what were the necessary constructs we needed to include in the language to allow our experts to configure the user interface behavior without getting lost in the details.

### **Simulation environment**

The second validation step is simulation-based studies. The STEEP system architecture supports the generation of simulated execution through a supervisor console. The console helps the supervisor control the environment, including the simulated patient's response to treatment and the behaviors of other simulated players, such as physicians ordering drugs and procedures, nurses administering drugs, and laboratories delivering lab results. Sample data for simulated execution of protocols are stored in XML files that the execution engine accesses and the TMC displays just as they would with real data.

The simulation must be conducted in a realistic environment, where ICU personnel can face treatment management situations similar to real life and can interact with the system to make decisions. The validation process must be closely monitored and the results precisely evaluated. At one point in the development, we relied for this evaluation on the infrastructure provided by the Simulation Center of

the Center for Experiential Learning and Assessment at VUMC [218]. The Simulation Center not only helped validate the protocol models in terms of proper structuring (e.g. separation of actions into processes related to diagnosis, treatment and prophylaxis) and sort out timing issues (e.g. fluid challenge and early goal-directed therapy processes can happen at the same time), but also provided valuable training to the medical personnel in terms of how to operate the system when it was time for them to use it on actual patients after the deployment of the tool into the ICUs. In later phases of the project, but before the completion of the Matlab-based simulation environment, we however decided to only use the production version of the tool for simulation for the sake of simplicity, which did not have the ability to speed up or slow down the simulation time.

While the development of the simulation environment required many resources, it had a tremendous value when our team discovered logical flaws in the original sepsis CIG. The most obvious one is illustrated by the workflow diagram of Figure 34, where if “no” was given to the first and “yes” to the second question, then no antibiotic would be given to the patient, which is not the right recommendation.

### **Verification through model checking**

#### **Potential benefits and examples**

There are many potential benefits of using formal verification in CIG development. However, successful verification is greatly dependent on the problem definition, including the problem space and the property being analyzed, and the technique used (e.g. model checking, theorem proving, runtime verification, and statistical model checking)

We gathered a categorized set of examples where facilitating verification could greatly improve CIG-based CDSS (see Table 10):

Table 10 - CIG-based CDSS verification opportunities

Target of analysis	Description	Example
CIG language	Provide an analysis of the language in terms of soundness	<ul style="list-style-type: none"> <li>• Correctness of structural semantics: Does the model comply with the structural requirements defined by the language?</li> <li>• Compare structural and execution semantics: Is there a possibility to create models, which do not make sense (i.e. are not executable)?</li> <li>• Correctness of execution semantics: e.g. evaluate the inter-hierarchical information exchange of concepts in CPML (Protocol-Process, Process-Process, Process-Activity)</li> <li>• Analyze expressivity: e.g. overlap and interaction of the Expression Language and Processes in CPML</li> </ul>
Implementations of the EE	Provide an analysis of various execution semantic implementations	<ul style="list-style-type: none"> <li>• Compare (Java-based) real-life and (Matlab-based) simulation environments</li> </ul>
CIG models	Provide an analysis of CIG implementations against execution-specific requirements	<ul style="list-style-type: none"> <li>• General non-functional properties: e.g. deadlock freedom, determinism, division by zero, integer overflow</li> <li>• CIG language-specific properties: e.g. in CPML <ul style="list-style-type: none"> <li>○ A defined problem should always be solved</li> <li>○ All solutions should be considered</li> <li>○ At a given time only non-interacting solutions should be running</li> <li>○ At a given time there should be at least one solution running</li> <li>○ Never finish in "Complete" (i.e. never stop treatment with claiming success) if patient parameters are out of the specified range</li> </ul> </li> </ul>
CIG models	Provide an analysis of CIG implementations against domain-specific requirements	<ul style="list-style-type: none"> <li>• Dosing: e.g. no medication order should allow to surpass the maximum permissible dose per administration threshold</li> <li>• Negative outcomes: e.g. hypotension should not be explicitly allowed, which means that no process monitoring diastolic blood pressure should allow a minimum threshold to be set lower than 60 mm Hg</li> </ul>
CIG models	Provide an analysis of CIG implementations against guideline-specific requirements	<ul style="list-style-type: none"> <li>• Sepsis protocol requirement examples: <ul style="list-style-type: none"> <li>○ Lab test should happen before antibiotics</li> <li>○ There should be at least 2 different antibiotics ordered</li> <li>○ MAP issues should always be addressed</li> <li>○ Prophylaxis should always happen</li> </ul> </li> </ul>

---

CIG models

Provide an analysis of CIG implementations against a set of real-life treatment scenarios to gain knowledge on the quality of implemented CIGs

- Figuring out how good is the sepsis treatment requires
    - many actual test cases (that include the execution trace of the treatment steps) to be compared with the recommendations of the CIG (i.e. the "ideal treatment") to the same patient input
    - alternatively, working patient models (i.e. representation of how a patient reacts to a given input) could be feeding data to the system to see if the CIG could sufficiently treat the patient, or not
- 

### Our approach

The benefit of using DSMLs is that the domain models can be formally verified against established criteria. This is a significant step forward. In traditional approaches, where the system is manually coded, the model is not explicit and cannot be independently verified. Our models support verification on three levels.

The first level is static model verification, which the GME provides. Metamodels include well-formedness rules that separate syntactically correct models from incorrect ones. The constraints are expressed using OCL. During modeling, GME enforces these well-formedness rules. In CPML, the constraints include clinical limits for parameters as well as more sophisticated rules that would be difficult to check without automated verification (see examples in "Relationship of Protocols, Processes and Activities" section in Chapter VI).

The next level is verification of dynamic properties at design time. The execution engine transforms models into behaviors at runtime. In fact, protocols are instantiated into a complex, multithreaded program that interacts with ICU personnel, patient data, and events. Using well-defined, clean execution semantics (such as CSP) is crucial for verifiability of the models against a set of predefined behavioral properties such as determinacy, livelock, and deadlock. Concurrently, as previously described, we have developed a model translator to map the protocol models into an intermediate executable model using Matlab Simulink/Stateflow. The Stateflow models can drive a number of verification tools, such as model checkers, simulators, and reachability analysis tools. Our examples of how we use these tools in implementing a dynamic verification of the behavioral properties of CIGs are described later in this chapter.

At the final level, critical actions that are performed during the treatment need to be checked at runtime. Security and privacy policies determine access rights to data published through the TMC and to



the invocation of actions, such as initiating treatment processes and ordering medications. In the current implementation, we rely on general ICU access-control policies, but we intend to make this customizable in later versions. Decisions present in the protocol let healthcare professionals order various actions during treatment that must be not only logged but also matched against a set of legal regulations and the hospital's own policies. Systems interfaced to the execution engine perform several of these checks; for example, the CPOE system checks all medication-related actions against a large suite of rules that include dosing and timing constraints.

## **Tools**

### Simulink Design Verifier

For the verification of behavioral properties of CIGs, we chose Matlab SLDV because of its capability to analyze our generated SF models with only moderate changes to them. SLDV uses formal methods, provided by the Prover proof engine [219], to detect hard-to-find design errors in models without requiring an exhaustive testing. Detected design errors include dead logic (i.e. parts of the SF model considered unreachable), integer overflow, division by zero, and violations of design properties and assertions. After detection, erroneous blocks in the diagram are highlighted and SLDV calculates signal-range boundaries and generates a test vector (i.e. an example) that reproduces the error in simulation [220]. These examples can be analyzed with the help of a "test harness" generated by another toolbox SLDV relies on, called Simulink Verification and Validation [221].

### Alternative methods

We recognize that there are many alternative approaches we could have taken for the formal analysis of CIGs. As our primary goal was to facilitate the SF models already generated for simulation, we predominantly focused on using existing Stateflow/statechart analysis methods. Besides Matlab SLDV, the existing tools we have considered included HiVy [222], which translates SF models (with some restrictions) to the input language of the Spin model checker, HyLink [223], which transforms certain classes of SF models to UPPAAL and HyTech for verification, and Polyglot [224], which translates SF models and temporal properties specified with a pattern-based system [225] to Java in order to perform analysis with the help of Java Pathfinder / Symbolic Pathfinder.

Additionally, we have also considered the use of a simpler (i.e. less expressive) MoC as the basis for expressing the behavioral semantics of CPML. Such solution could have the added benefit of providing better analyzability. As an example, we have experimented with the creation of timed automata-based templates for the generation of timed automata-based behavioral models, which would have enabled the analysis of the behavioral semantics with the help of UPPAAL.

However, the problem with timed automata is spatial complexity. In practice, very large models [226], a combination of states, invariants, guards (conditions + clocks) and variable ranges, often require large amounts of memory, which during execution first results in a lot of page faults and in extreme cases, the system will run out of memory. Based on previous research experience [227–229], we determined that our models contained too many time-related properties for UPPAAL to handle. Furthermore, the lack of hierarchy made the number of the behavioral models grow exponentially.

### Verified examples

For testing purposes, we constructed a simple CIG using a small subset of the original sepsis CIG (Figure 48). The resulting simplified sepsis CIG can be seen on Figure 50. It contains only one Process, which has two Activities.

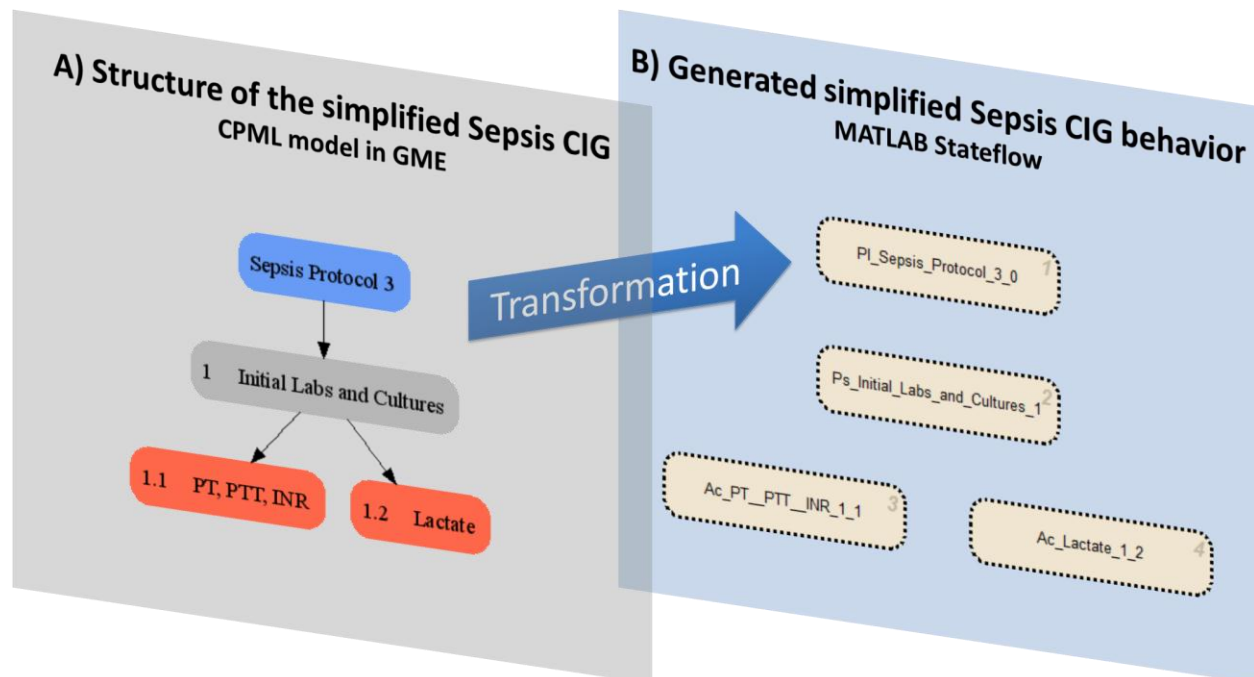


Figure 50 - Generation of SF models from CPML (Simplified sepsis)

In the following subsections, we show five examples, in which various invariants were tested against this simplified model. The creation and the analysis of each example were performed using the following steps:

1. If required by the example, alter the behavioral templates (e.g. UI\_Activity, Protocol state, etc.)
2. Invoke the generator of the analysis tool chain to get the Matlab model
3. Manually extend the model
  - a. Create an SLDV block implementing the property against which the model will be checked
  - b. Add data and events required by the analyzer (SLDV block) as outputs from the SF chart
  - c. Connect SF outputs to SLDV block
4. Run SLDV
5. Confirm results

#### *Example 1*

Figure 51 shows the contents of the first example (SP3\_20120316\_1706\_2\_A.mdl). In order to get rid of potential variations caused by human interaction (e.g. start of the protocol, confirming the treatment), in the following examples the Protocol and the Action controllers (UI\_Protocol and UI\_Activity) are replaced. The current example is configured so that the Protocol will start automatically at a time specified by a step function (see the upper left part of the figure). Similarly, Actions are confirmed (i.e. started) at a fixed time set to be after the time of the start of the Protocol (see the lower left part of the figure). The later step function, extended with appropriate delays (still acting as a user), will also order the medication, then (acting as the CPOE system) will acknowledge and finally confirm the completion of the Activity.

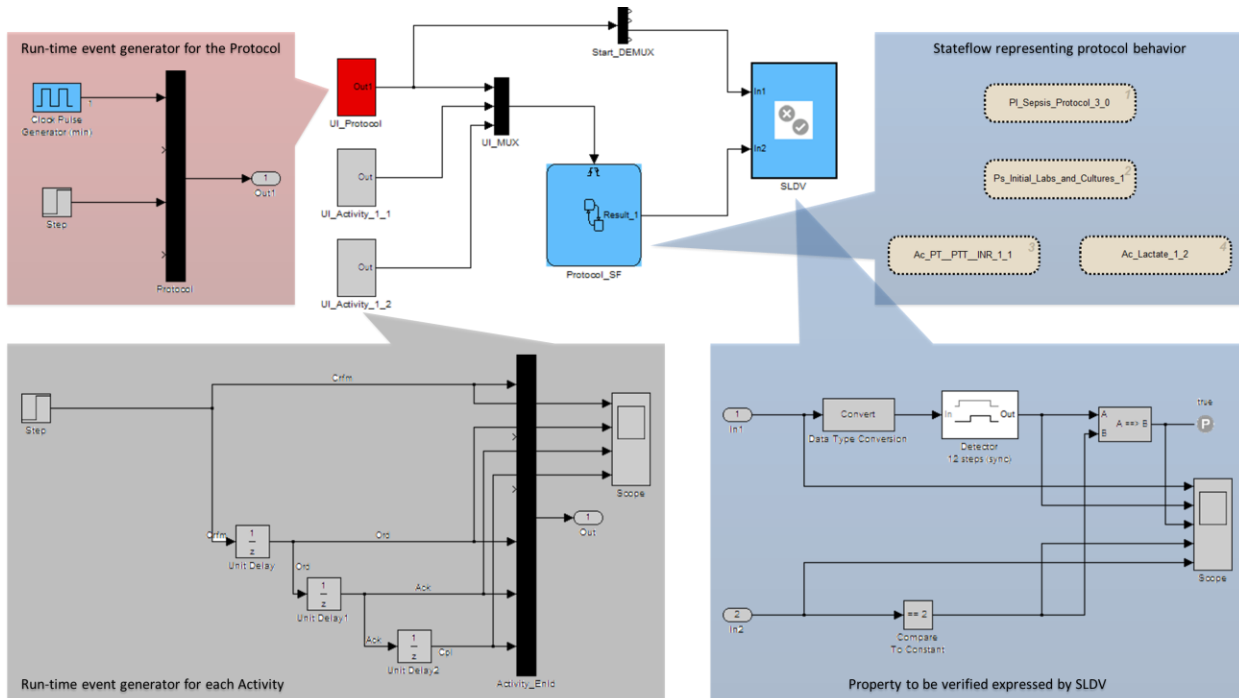


Figure 51 - Generated functional models in MATLAB (Example 1 and Example 2)

The property to be verified is expressed by the SLDV block (in the right lower part of the figure). With the help of a signal detector, a comparator and an implication, the property (“P”) defines a Boolean formula, which will only be true if, following the start of the Protocol (signaled through “In1”), after a fixed amount of time (12 steps) the Process completes (signaled with a value of 2 through “In2”). With the above-mentioned assumptions, the property can be expressed with the help of linear temporal logic (LTL) in the following way:

$$\varphi = G(In_1 \rightarrow X^{12}(In_2 = 2)), \text{ where } X^n(expr) = \underbrace{X(\dots X(expr))}_n$$

Symbols  $\varphi, G, X$  denote the property, all future and the following step respectively. By replacing the two input parameters ( $In_1$  and  $In_2$ ), this equation can be translated to be a function of the user triggering the start of the protocol ( $UstStart_{prt}$  emitted by the Step function of the UI\_Protocol) and the completion condition of the main process ( $Cpl_{prc}$ ):

$$\varphi = G(UstStart_{prt} \rightarrow X^{12}(Cpl_{prc}))$$

If we consider the assumptions encapsulated by the UI\_Protocol and the two UI\_Activities as part of the definition, the expression will change to the following:

$$UsrStart_{prt} \wedge XG(\neg UsrStart_{prt}) \wedge \left( \bigvee_{k=1}^5 X^k(\omega_1 \wedge \omega_2) \right) \rightarrow \varphi$$

where  $\omega_i = UsrCfrm_{Act_{1,i}} \wedge X(UsrOrd_{Act_{1,i}}) \wedge X^2(UsrAck_{Act_{1,i}}) \wedge X^3(UsrCpl_{Act_{1,i}})$ , and  $i = \{1, 2\}$

These expressions state that once the Protocol is started (permitted once per simulation) and are followed by the Activities with an appropriate delay the original  $\varphi$  property will hold.

Using an above than average laptop computer<sup>95</sup>, SLDV proved this objective to be valid in approximately 37 hours<sup>96</sup>. We found this result to be below our expectations, as it showed that using the same templates were not going to scale for larger models.

### Example 2

We suspected that some of performance degradation in Example 1 was caused by the fact that the models were designed to be “livelock proof”, in the sense that Activities and Processes would always reach a final (i.e. halting) state after a preset amount of time. This was achieved with the use of various timeouts (e.g. monitoring, running timeout). To evaluate our theory we created the next example (SP3\_20120316\_1706\_2\_B\_noTO.mdl), which used the templates from Example 1 without the timeout transitions from the inside of the Active states.

Using the same setup, including the properties of interest, the verification took 4 hours, supporting the hypothesis.

### Example 3

Both previous examples, Example 1 and Example 2, used a manually pre-coordinated user interaction mechanism (separately set up Step functions in UI\_Protocol and UI\_Activity). To improve on this design, we created a new UI\_Activity template, in which the user was guaranteed to respond automatically (with a fixed delay) to recommendations made by the protocol. We also further simplified the Process template by removing the goal tracking concurrent state, which was unnecessary, as a goal was not defined for this example. The example generated with the same CIG (SP3\_20120316\_1706\_2\_C\_autoDoc.mdl) is shown in Figure 52.

---

<sup>95</sup> Intel i7-2630QM quad core CPU with 12 GB of RAM running the 64-bit version of Windows 7.

<sup>96</sup> To complete the verification, SLDV approximated the floating-point arithmetic with rational number arithmetic.

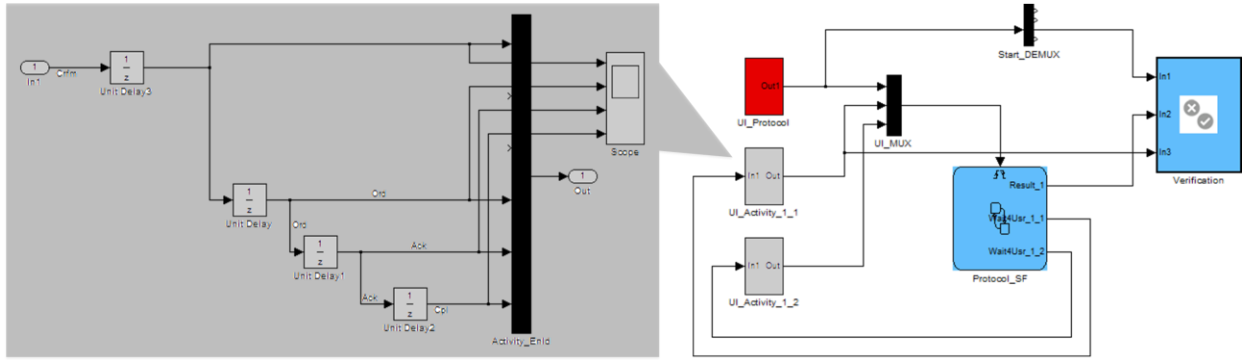


Figure 52 - Generated functional models in MATLAB (Example 3)

The new assumptions change our LTL expressions to the following:

$$UsrStart_{prt} \wedge XG(\neg UsrStart_{prt}) \wedge G\left(\bigwedge_{i=\{1,2\}} Rcmd_{Act_{1,i}} \wedge X(\omega_i)\right) \rightarrow \varphi$$

$$\text{where } \omega_i = UsrCfrm_{Act_{1,i}} \wedge X(UsrOrd_{Act_{1,i}}) \wedge X^2(UsrAck_{Act_{1,i}}) \wedge X^3(UsrCpl_{Act_{1,i}})$$

With this setup, the verification took 4.6 hours.

#### Example 4

Figure 53 shows our next example (SP3\_20120316\_1706\_2\_D\_XOR.mdl), in which we further experimented with the property expression language of SLDV. The property specified in this example requires all execution traces to allow the completion of one Activity only, which implements exclusive choice. In the model, the users are assumed to accept the first (“Doc\_Cfrm”) and decline the second (“Doc\_Dcl”) Activity recommendation. To this to happen, the generated model was manually reconfigured, as our generator allows the use of only one template per executable component.

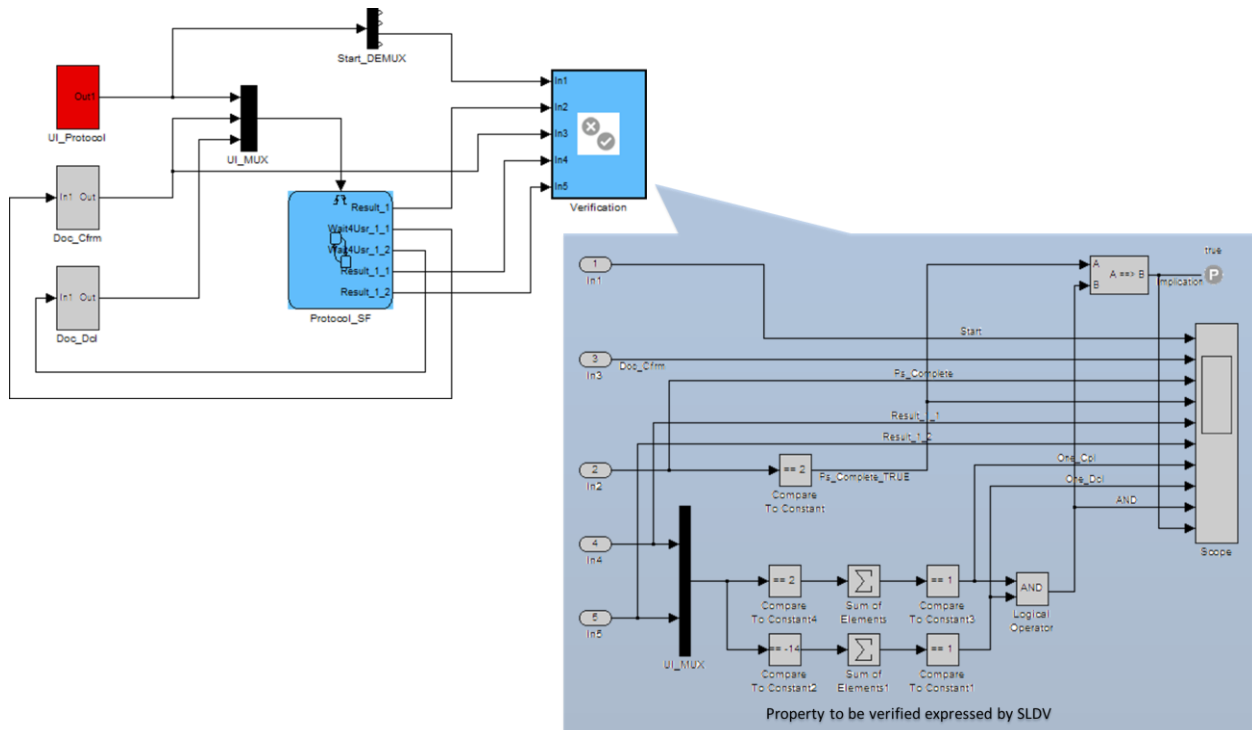


Figure 53 - Generated functional models in MATLAB (Example 4)

The property for this example can be written as:

$$\varphi = G \left( (In_2 = 2) \rightarrow \left( \left( \sum_{i=\{4,5\}} \{1 | In_i = 2\} \right) = 1 \right) \wedge \left( \left( \sum_{i=\{4,5\}} \{1 | In_i = -14\} \right) = 1 \right) \right)$$

Where the completion and decline events of an Activity are signaled through  $In$  with a value of 2 and  $-14$ . Alternatively, if completion and decline events are represented as  $Cpl_{Act_{1,i}}$  and  $Dcl_{Act_{1,i}}$  the expression changes to the following:

$$\varphi = G \left( Cpl_{PrC_1} \rightarrow \left( \left( \sum_{i=\{1,2\}} Cpl_{Act_{1,i}} \right) = 1 \right) \wedge \left( \left( \sum_{i=\{1,2\}} Dcl_{Act_{1,i}} \right) = 1 \right) \right)$$

The verification of this property took 3.7 hours.

#### Example 5

The final example (SP4\_20120319\_1834\_1\_E\_seqProcs.mdl) was set up to verify the sequential execution of the two Activities in a slightly different CIG model. In this model, the two Activities were

placed into two separate Processes in the main Process. The reason for this was to create a simple test, where an Activation event realized sequencing between the two lower Processes (and ultimately the two Activities as well). Figure 54 shows the SLDV property specification, according to which both of the lower Processes need to finish for the main Process to finish.

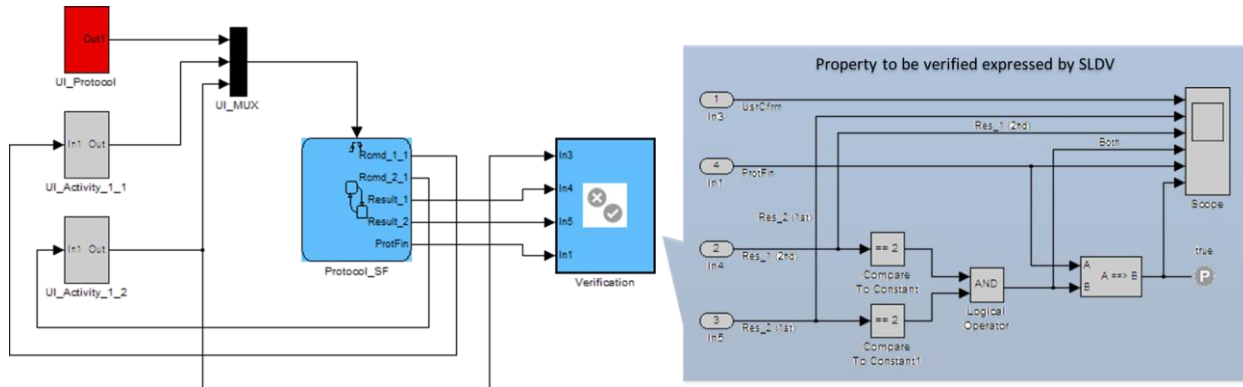


Figure 54 - Generated functional models in MATLAB (Example 5)

The property for this example can be written as:

$$\varphi = G(In_1 \rightarrow (In_4 = 2) \wedge (In_5 = 2))$$

or alternatively as

$$\varphi = G(Fin_{prt} \rightarrow Cpl_{prc_1} \wedge Cpl_{prc_2})$$

The verification of this property took 1.1 hours.

## Evaluation

The following list describes the chief limitation of the solution presented in this thesis for the verification of CIGs:

- Simulink does not support concurrency in terms of a nondeterministic interleaving of concurrent events. Thus, SLDV does not provide any means to check a model for any type of concurrency related properties (e.g. race conditions) [230].



- SLDV currently is only compliant with a subset of the Simulink/Stateflow language, which means that our behavioral templates had to be altered considerably. This included the need for using “directed message broadcasting” for the dissemination of SF events, instead of regular broadcasts. This limitation not only made the generated models far less readable for humans, but required changes to the behavioral templates that complicated the code generation. Furthermore, SLDV does not support the super step semantics [231] provided by SF, which fundamentally altered the meaning of our behavioral templates.
- Another disadvantage was that the generated behavioral models for the sepsis guidelines reached a high complexity, which Matlab and SLDV had a hard time dealing with, even after making numerous optimizations. For example, it became apparent that the performance of the copy/paste Matlab API command that we use in the instantiation of the templates in the generator is dependent on the size of the model to which the new instance is being added. While this dependency was only polynomial in time, in practice, it made the Matlab-based generation of the full sepsis CIG impossible with most machine/version configurations<sup>97</sup>. As another example, in order to produce results with SLDV in the magnitude of hours, we needed to simplify behavioral templates and treatment models to the point, where their clinical usability was diminished. The simplification was an experimental process, during which we tested out both alternative, but equivalent behavioral models, as well as, more abstract and simplified versions of our behavioral templates. The latter ones benefited during verification from reduced number of transitions (e.g. removal of time-outs) and reduced number of concurrent states, and stored information with the help of shared variables instead.

Such problems indicate that a considerable amount of energy needs to be spent on (1) further optimizing behavioral templates or (2) finding alternative Stateflow verification methods or (3) using alternative MoCs for defining the semantics of CPML.

However, we believe that after a thorough testing, systems, such as STEEP, should be able to operate in a clinical environment without a complete verification of all critical properties, as currently they are not applied as safety critical (i.e. hazardous) systems, in the sense that they are not completely autonomous.

---

<sup>97</sup> We tried several Matlab versions (v2011b, v2012a) on multiple Windows 7 machines. The only machine able to provide a solution was Linux-based.

This means that every patient influencing action will only be performed after a careful evaluation of users, who are professionals trained to solve problems without these systems in place.

## CHAPTER IX.

### CONCLUSION

#### Summary

#### Motivation

The use of evidence-based guidelines for managing complex clinical problems has become the standard of practice, but typically, clinical guidelines are defined and disseminated in a natural language, which inherently supports ambiguity. Furthermore, guidelines are not patient care plans, which means that to be truly effective, they must be deployed as customized and individualized clinical care plans, in other words, protocol instances. Our approach of using CIG models that formally capture medical knowledge and its interaction with contextual information in care delivery systems inherently supports this idea by allowing protocol models to be tailored and executed on a per-patient basis.

#### Existing approaches

Today there are many different frameworks for modeling, verifying and executing medical guidelines. Built by various groups, each of these CIG-based systems differ in their scope and implementation, but were developed with the intention that by using them non-programmers will be able to create, maintain and facilitate computerized clinical guidelines [36]. Current approaches can be split into two distinct groups: (1) ones that rely on custom-built execution environments and (2) ones that build on much general, but robust existing industrial solutions:

Proprietary systems offered by solutions of group (1) define and implement highly specific concepts and associated execution semantics, which are customized to the needs of the particular (clinical) domain. The disadvantages of these systems are that they often have modeling languages with limited readily available expressivity; their execution environment is not robust in terms of scalability and operational reliability; and their integration to various CISs is not addressed. Some of the most well-known projects and formalisms include the Arden Syntax, the Asgaard/Asbru project, EON, Gaston, GEM, the GLARE framework, GLIF, HELEN, SAGE, the SpEM framework, PRODIGY, and PROforma, out of which we closely examined four in this thesis.

The other types of CIG-based frameworks, members of group (2), rely on robust, standard-based industrial solutions. Resulting frameworks range from ones building on workflow-based standards and execution environments [232–235] to others facilitating rule-based systems [236,237]. This technique alleviates many of the issues presented by the solutions of group 1; however, they introduce ones of their own. The inherent problem of placing clinical applications on top of general business solutions is that resulting systems will be constrained in terms of expressivity of the chosen business platform. For example, special logical constructs, ones only seen in the clinical domain, are not implemented by any of the domain-independent platforms [82]. Another issue is that currently offered general modeling languages (e.g. BPEL, UML Activity Diagrams) do not use abstractions familiar to health care professionals.

There is a continuing interest in improving clinical decision support systems despite all listed efforts, most of which took several years to develop, as none of the offered solutions gained wide clinical adoption.

### **Current status**

Since the second half of 2007, as a result of the collaborative research projects shared between VUMC and ISIS, we have developed and implemented a model-based patient management system for sepsis [71,238–241]. Our research team of medical experts, computer scientists and system integrators created

1. a language, called CPML, together with a general modeling environment for representing clinical guidelines,
2. a computer interpretable sepsis treatment protocol
3. a general software architecture for the analysis and the execution of guidelines.

The resulting system is called STEEP. While STEEP is currently configured with the sepsis guideline, it is capable of running any CIGs created with CPML. It provides personalized decision support to assist in evidence-based patient management. By merging formal biomedical models with specific patient information, it supports both bedside decision-making and clinical research. It represents the status of the patient in workflows, management options as guided by clinical research protocols, and expected consequences.

The clinical usability and effectiveness of the STEEP system is currently being evaluated in a clinical trial at two ICUs, the medical and the surgical ICUs of VUMC. Our hypothesis is that STEEP decreases the time it takes to detect patients developing sepsis, and improves both physician compliance with evidence-based standards and clinical outcomes for patients.

### **List of contributions**

Our work contributed to the design of CIG-based CDSSs in the following areas:

1. Creating a detailed list of critical open problems that CIG-based CDSSs face today.
2. Leading the development of CPML, an integrated modeling language required for the model-based configuration of the STEEP system, which is a real-life functional experimental CIG-based CDSS configured with a CIG for the management of sepsis and running at VUMC [71].
3. Contribution to the GUI design and the selection of its configurable features in STEEP [71].
4. Contribution to the design of the execution engine that runs simultaneous instances of the protocol in STEEP [71].
5. Contribution to the design of the communication interface for the proprietary CPOE used in STEEP.
6. Contribution to the modeling of the sepsis CIG with the help of CPML [71].
7. Creating a methodology and a related software suite comprised of generators and Matlab Simulink/Stateflow to allow the template-based generation of behavioral models, as well as, their simulation and formal analysis.
8. Formal analysis of various properties over a set of simplified examples taken from the modeled sepsis CIG.

### **Lessons learned and future directions**

The STEEP project provided our interdisciplinary team an invaluable opportunity to investigate model-based treatment management problems, and analyze and understand related technology directions. The development of the STEEP architecture and making it suitable for a clinical trial have proved to be a significantly larger effort than we expected. Through the design and the development of our system we gathered a list of scientific challenges. The findings generic to CIG-based CDSS were summarized in the

“Open problems” section of Chapter III. In this section, we discuss the lessons learned related specifically to STEEP and provide insight into what the next steps of this research could be.

Table 11 shows the tasks in the STEEP project that took the most amounts of time and energy to complete. Explanation of each task is provided in the following sections.

**Table 11 - The most significant development task in the STEEP project**

Task	Effort	Reusability	Section containing the explanation
CPML development	High	High	“Knowledge representation and management: Language”
CPML EE development	Moderate	High	“System development and evolution”
Sepsis CIG development	Moderate	Moderate	“Knowledge representation and management: Protocols”
STEER GUI development	High	Moderate	“System integration”
STEER integration into the VUMC CIS	High	Limited	“System integration”
CPML CIG simulation and analysis environment	Moderate	High	“Validation and verification”

### **Knowledge representation and management: Language**

#### The drawbacks of language development

The development of CPML was a large effort. Some of the most significant reasons are listed in the following subsections. We, however, believe that our efforts were not in vain, as many components of the language are novel and highly reusable in other CIG-based CDSSs.

#### Workflow to rule-based representation

As described in the “Early attempts” section of Chapter VI, our team started with the assumption that some of the workflow-style standards or existing modeling languages will provide the required abstractions for modeling sepsis treatment protocols. After much experimentation, we decided to develop domain-specific abstractions that were built around the Process concept supporting concurrency and event-based communication. The result was an evolving modeling language, which provided adequate expressiveness for our clinical example.

#### Separation of knowledge layers

CPML integrates many different kinds of knowledge, including medical guidelines, execution semantics and interface abstractions of connected systems. Based on our experiences, we concluded that it is

critical to decompose the knowledge in problem domains with complexity similar to CIGs into sub-languages that represent separate and essential aspects of the problem space, and obtain complete models via model composition. In CPML, we progressively identified and separated multiple sub-languages that represent essential aspects of the problem space. As described in Chapter VI, sub-languages were organized into three major categories: Medical Knowledge Modeling, Protocol Modeling and Model Management and Support. We believe that the decoupling of the various sub-domains provides flexibility on multiple levels.

With the help of sub-languages, we separated the two most important types of knowledge represented in CPML models: (a) the medical knowledge, which represents a medical ontology and (b) the behavioral knowledge, which defines how certain tasks described in the guideline have to be executed. These two kinds of knowledge are significantly different and are traditionally handled by their respective communities: medicine and computer science. Both of these aspects are complex and have semantics of their own. Such separation allows for the independent construction and management of both knowledge bases by their respective community. Knowledge can be updated according to their own lifecycle. This is an important benefit, because after an initial design phase, the operational knowledge usually has a relatively slow evolution, while medical knowledge requires more frequent updates and extensions. In fact, the separation proved extremely useful when we were experimenting with various execution models, as we could quickly generate and test the complete behavior after changing the behavioral templates, while leaving the treatment models unchanged. Further decomposition in the medicine domain allowed the sepsis treatment protocol to continuously evolve, while order sets of medications, procedures and laboratory tests were stable as defined by the Vanderbilt's CPOE.

Another benefit of the separation is that medical knowledge can be expressed using common existing formalisms (e.g. OWL [242]) and existing knowledge sources (e.g. UMLS) can be employed. Decoupling also allows for the execution semantics to be expressed explicitly with a MoC, in our case Stateflow, which makes all the existing methods and tools for the chosen MoC available. For STEEP, this included verification techniques and simulation and execution environments provided by the Matlab tool suite.

### Learning CPML

Although the CPML environment provides health experts with the necessary domain specific concepts, CPML still has to be learned. As an alternative, knowledge experts can rely on and collaborate with an

experienced modeler to capture guidelines (similarly to the collaboration that existed in the Sepsis project), which requires time and energy spent at both sides.

## Limitations

### Adoption of CPML protocols

Due to the way the language was constructed, Protocols are expressed by using Orderables, and Orderables are constructed using elements of the Medical Library. This setup means that Protocols are always expressed with the help of building blocks specific to a HCO<sup>98</sup>. While this approach has the benefit of allowing modelers to create CIGs that are directly runnable at a given location, it also introduces an indirection, which makes the understanding of protocols created at one institution by an expert at another difficult, as they need to look at how each orderable is constructed before they can fully understand the models. Another disadvantage is that the adoption of protocols created at other institutions becomes non-trivial. For this process to be automated, a mapping between the orderables of the other HCO and elements of the Medical Library needs to be created.

### Information aggregation

As described by “P2.10” in Chapter III, the support for information aggregation in CPML is fairly limited. We are able to represent problems described in “Example 2: Defining composite indicators” (e.g. diagnosis), but for solving “Example 1: Aggregating different sources” and “Example 3: Identifying the source of indicators” we rely on our modelers and VUMC’s data cleaning and aggregating processes respectively.

### Representation of planned actions

One of our original goals was to develop a language and provide a modeling environment for capturing a large variety of clinical guidelines. However, when trying to adopt our solution to the modeling of other management protocols, we realized that the design of CPML was solely influenced by the problems present in ICUs. Thus, CPML was highly oriented towards representing solutions for acute care situations. This caused problems, as the solutions for acute problems are typically reactive in nature, which means that as soon as a problem is detected an associated solution needs to be identified and

---

<sup>98</sup> This issue is also discussed in the “P2.8” point of Chapter III.



executed. Because the management of chronic diseases introduced additional complications related to the long-term treatment of patients, we needed to extend CPML to allow a planning oriented modeling.

Our conclusion is that for future problem domains (such as the cancer management we are currently analyzing) we need to further decompose CPML into reusable sub-languages instead of trying to apply it as it is, or develop new domain specific modeling languages from scratch. Reusable model libraries built for these separate aspects will then be used to generate the integrated domain specific models.

#### Future applications

The next version of CPML is currently under development [243]. It is being developed as part of another collaborative project with VUMC. The project, called the Vanderbilt Oncology Information System (VOIS) program, was launched in January 2011, with the goal of creating a robust and integrated clinical information system to support the cancer care continuum. The program includes informatics methods to support longitudinal cancer treatment plan management, genome-directed cancer treatment prioritization, and team-based approaches to cancer diagnosis and treatment response assessment.

Longitudinal cancer treatment plan management is one of the more complex projects within the VOIS program. The goal is to build a model-driven system to represent, instantiate and manage patient chemotherapy plans integrated into the existing VUMC clinical information systems. This includes integration with a broader range of Vanderbilt's CIS, including the CPOE [207], the PIS, the nursing documentation system, the EMR system [244], and the outpatient whiteboard. To facilitate the cooperative work of the multi-disciplinary care team, the system must also represent and adapt to real time changes in the state of the patient, operational resources, and medical knowledge.

All of these components must be represented as part of the CIGs in CPML at multiple levels of abstraction. This problem is complicated by the fact that longitudinal patient management plans need to be continuously adapted to compensate for the changing environment. With this ambitious scope for modeling, a framework is needed to assist in the iterative development of these complex models to both verify that the models are computationally satisfiable and clinically valid.

In [243], we propose an architecture and preliminary results regarding the construction a new class of CDSSs for synthesizing and managing cancer treatment plans. In this work, our goal is to automatically synthesize – and based on the changing environment – continuously re-plan complex treatment plans. Our system will combine: (a) state-of-the-art formal modeling to capture treatment abstractions and

medical ontologies, (b) state-of-the-art finite model finding and constraint solving to synthesize treatment plans, and (c) state-of-the-art CISs developed by VUMC.

We anticipate the development and maintenance of over 1000 different chemotherapy protocols of varying complexity [209]. With such a high volume of CIGs, automated systems are needed to assist in data driven validation of these models to ensure patient safety. Although in this work, our technical solution focuses on cancer management, our approach can be applied to the plan management of other clinical problems as well.

### **Knowledge representation and management: Protocols**

#### Sepsis CIG development

##### Effort

The development of the sepsis CIG required a group of domain experts and computer scientists to coordinate their efforts. Furthermore, to create an actual runnable implementation, the original sepsis guideline needed to be extended and tailored to match the requirements (e.g. minimize the number of required interaction with the clinicians) and available resources (e.g. rely only on locally available data streams) of the VUMC ICUs. Lastly, as CPML evolved, we needed to make sure that the guideline was always migrated.

##### Reusability

The sepsis CIG is designed to manage sepsis, which entails solving multiple concurrent problems. Thus, reusability is supported, as independent solutions to sub-problems can be reused in other CIGs needing them.

##### Improvements

Through the CIGs, decision-making is influenced by population-based evidence. If the trial confirms the clinical suitability of STEEP, the decisions and the actions carried out with STEEP will be documented, monitored for effectiveness, and used to improve the existing support models.

### Importing CIG written in other languages

As we demonstrated, there has been a tremendous amount of energy put into the formalization of medical guidelines by other research groups using various formalisms. Consequently, these efforts resulted in a significant number of already modeled protocols. We believe that by defining mappings between each formalism and CPML we could import and facilitate them.

### CIG library

Because in the STEEP project there was only one CIG created, our team did not need to concentrate on solving otherwise critical problems related to the management of CIGs. For handling a large number of concurrently working guideline authors and a broad range of CIG models, GME's single-user and "one-file-for-all-CIGs" approach would have been inadequate. For addressing this potential issue, we have started to work on a solution involving libraries, which would allow users to import CIGs from a central repository and submit their changes.

### **Validation and verification**

In STEEP, the behavior of the sepsis treatment manager is the result of the execution engine interpreting the CPML models. Relying on the execution environment implementation extended with simulation controls, the correctness of the created sepsis CIG was validated by multiple VUMC ICU physicians as part of a clinical evaluation completed in 2009. The validation used a case-based test approach, where various scenarios were run to analyze the behavior of the implemented guidelines.

To be able to formally analyze the modeled treatment models, we created a simulation and analysis environment. This component of STEEP required the formal definition of the behavioral semantics of the concepts in CPML, which was done by translating CPML into Matlab's Stateflow. The resulting Stateflow models were analyzable with Matlab's verification tools, in our case, the SLDV package. In addition to providing coverage metrics for identifying unreachable sections of the state space, these tools allow the checking of functional requirements (e.g. patient safety) over the behavioral models.

### Formal behavioral models driving the execution

A future idea we would like to explore is using our generated Matlab-based behavioral models to serve as the basis for generating code for the EE. Currently, our EE and simulation environments are

developed independently of each other, which, as mentioned earlier in Chapter VIII, would require a thorough analysis to ensure that their behaviors do not contradict each other.

To solve this problem, the relationship between the SF-based specification of behavioral semantics and the execution semantics implemented by the STEEP Execution Engine need to be examined. The STEEP project schedule did not allow the establishment of a formal relationship between the two, because the STEEP Execution Engine, CPML and the STEEP models were developed in parallel, and the SF-based behavioral semantics was completed in the last phase of the project. While developing the SF-based specification of semantics, consistency with the EE was ensured via informal discussion and reviews. However, we have not completed the modification of the Java thread package-based concurrency model of the engine to match the “pseudo” concurrency model of SF<sup>99</sup>. This could have been possible, since the specification of SF behavioral semantics in Java has been recently completed in a parallel running project at ISIS, called Polyglot [224]. In future design flows, a more rigorous connection can be built by reusing and extending the implementations of the execution models from libraries of Polyglot.

#### Importing standardized dosing constraints as general properties for verification

As part of our future work, we will consider importing dosing constraints defined as part of standardized knowledge bases (e.g. FDB) to serve as general properties against which all implemented CIGs could be verified.

### **System integration**

The real power of CDSSs can only be harvested once integrated into a host CIS. However, we found the integration of the STEEP tools into VUMCs CIS to be a complex challenge, both in terms of (1) user experience and (2) interoperability. To improve the process, we applied model-based solutions to in the following ways:

---

<sup>99</sup> While SF has a notion of parallel states, it does not implement true concurrency, as there is always a deterministic order in execution. Polyglot uses the same model.

1. User experience: STEEP needed to fit into the existing workflow of clinicians. Because the GUI of STEEP is the only way to interact with an executing CIG, our domain experts needed to make sure that everything was conveyed in an appropriate manner. This generated two independent requirements: (a) the UI needed to be incorporated as a part of existing patient management solutions and (b) it needed to be easily configurable in terms of what and how it was showing. While (a) was strictly an implementation challenge, CPML addressed (b) by offering basic model-based constructs for CIG designers to control the elements of the UI. Because the GUI is only moderately tailored to VUMC and to the management of sepsis (e.g. it integrates information provided by the VUMC's sepsis surveillance tool) and otherwise as generic as CPML, one could argue that its reusability is high. However, a future adopter of the GUI would need to make sure that the use of the tool fits into the workflow of their users, which includes dealing with our customizations and the actual technology used to implement the GUI (e.g. GWT).
2. Interoperability: Information exchange between STEEP and the underlying CIS infrastructure also benefited from formal modeling. CPML offered constructs for describing types of data inputs (e.g. patient data and user actions) and outputs (e.g. medication orders) as part of the interface definition of STEEP. However, we struggled with complexity, which arose from the fact that many of VUMC's CISs were not designed to allow other tools to drive their execution. For these systems, new communication methods (including APIs, communication protocols and data elements) had to be defined. It also meant that we had to build interface models in CPML.

## **System development and evolution**

### Engine

The architecture we applied in the STEEP project allows CPML and its semantic implementations (the EE and the simulation environment) to evolve independently. While this is considered a benefit when experimenting with various implementations, their coordination is critical. We achieve this with a help of a set of generators.

### Execution semantic variations

In the model-based framework, the EE provides semantics for the protocol models. As part of the language design process, our interpretation of protocol semantics also shifted considerably from a

workflow-style view that defines behavior via observable treatment trajectories to a process view where treatment trajectories emerge from the interaction of event-driven treatment processes. The conclusion of our research has been that casting strongly different protocol execution models (e.g. reactive and plan-based models) into a single execution engine is ineffective and increases complexity. We propose that, similarly to the decomposition of languages, execution engines also need to be modularized and composed of reusable components.

#### Mission critical software development

The reusability level of the engine is dependent on the reusability of language and the quality of the implementation. Because the software was intended to be used in a clinical setting, robustness, including reliability and speed, were critical concerns that made the development difficult and the result, optimistically, highly usable. Once complete, the clinical trial should provide feedback on this matter.

We have found that while the model-based generation and reconfiguration of clinical applications provides the ability to rapidly create and modify the software, existing workflows for testing the clinical correctness and the integration of such applications at HCOs do not share the model-oriented view. This means that for purely model updates, where the software code is left unchanged, it does not matter how small the changes are, or how thorough analysis the models went through afterwards, the testing team will always consider the result as a separate version, and perform a full test of the software. Though considering how dangerous an overlooked mistake might be, this could be a good thing.

## APPENDIX A:

### LIST OF RELATED PUBLICATIONS

#### Peer-reviewed journal articles (first author)

1. Mathe, J., Martin, J., Miller, P., Ledeczi, A., Weavind, L., Nadas, A., Miller, A., Maron, D. and Sztipanovits, J. A Model-Integrated, Guideline-Driven, Clinical Decision-Support System. *IEEE Software, Special issue on Domain-Specific Languages & Modeling*. 26, 4 (Aug. 2009), 54-61. [71]
2. Mathe, J., Werner, J., Lee, Y., Malin, B. and Ledeczi, A. Model-Based Design of Clinical Information Systems. *Methods of Information in Medicine*. 47, 5 (2008), 399-408. [245]

#### Peer-reviewed journal articles (not first author)

3. Hooper, M.H., Weavind, L., Wheeler, A.P., Martin, J.B., Gowda S.S., Semler M.W., Hayes R.M., Albert D.W., Deane N.B., Nian H., Mathe J.L., Nadas A., Sztipanovits J., Miller A., Bernard G.R., Rice T.W. Randomized Trial of Automated, Electronic Monitoring to Facilitate Early Detection of Sepsis in the Intensive Care Unit. *Critical Care Medicine In press*, (2011). [240]

#### Book chapters

4. Mathe J, Werner J and Sztipanovits J. Model-Based Design of Trustworthy Health Information Systems. *Homeland Security Facets: Threats. Countermeasures, and the Privacy Issue*. Artech House; 2011. [241]

#### Conference papers

5. Mathe, J.L., Sztipanovits, J., Levy, M., Jackson, E.K., and Schulte, W. Cancer Treatment Planning: Formal Methods to the Rescue. *4rd International Workshop on Software Engineering in Health Care (SEHC 2012)*, (2012). [243]
6. Mathe, J.L. Towards an Adaptable Framework for Modeling, Verifying, and Executing Medical Guidelines. *Proceedings of the Doctoral Symposium at MODELS 2009* (Denver, CO, October 2009). [239]

7. Mathe, J., Miller, P., Ledeczi, A., Weavind, L., Miller, A., Maron, D., Nadas, A., Sztipanovits, J. and Martin, J. A Model-Integrated Approach to Implementing Individualized Patient Care Plans Based on Guideline-Driven Clinical Decision Support and Process Management - A Progress Report. *2nd International Workshop on Model-Based Design of Trustworthy Health Information Systems (MOTHIS 2008)* (Toulouse, France, October 2008). [238]
8. Duncavage, S., Mathe, J., Werner, J., Malin, B.A., Ledeczi, A. and Sztipanovits, J. A Modeling Environment for Patient Portals. *AMIA Annual Symposium Proceedings* (Chicago, IL, November 2007), 201-205. [246]
9. Mathe, J., Duncavage, S., Werner, J., Malin, B., Ledeczi, A. and Sztipanovits, J. Implementing a Model-Based Design Environment for Clinical Information Systems. *First International Workshop on Model-Based Design of Trustworthy Health Information Systems (MOTHIS 2007)* (Nashville, TN, September 2007). [247]
10. Werner, J., Mathe, J.L., Duncavage, S., Malin, B., Ledeczi, A., Jirjis, J.N. and Sztipanovits, J. Platform-Based Design for Clinical Information Systems. *Industrial Informatics, 2007 5th IEEE International Conference on (INDIN 2007)* (Vienna, Austria, July 2007), 749-754. [248]
11. Mathe, J.L., Duncavage, S., Werner, J., Malin, B.A., Ledeczi, A. and Sztipanovits, J. Towards the Security and Privacy Analysis of Patient Portals. *Special Interest Group on Embedded Systems Review (SIGBED Rev.)*. 4, 2 (2007), 5-9. [249]
12. Emerson, M., Mathe, J. and Duncavage, S. WiNeSim: A Wireless Network Simulation Tool. *Proceedings of the Sixth ACM International Conference on Embedded Software (EMSOFT'06 Workshop)* (Seoul, South Korea, October 2006). [250]
13. Werner, J., Eby, M., Mathe, J., Karsai, G., Xue, Y. and Sztipanovits, J. Integrating Security Modeling into Embedded System Design. *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006)* (April 2006). [251]



## APPENDIX B:

### CURRENT STATUS OF THE INTEGRATED STEEP ARCHITECTURE AT VUMC

Our team developed a STEEP prototype (see). Parallel development of the CPML language, the TMS-C and the associated EE allowed clinicians to see how changing the models (or the language) affected the system's behavior and allowed them to provide the development team with continuous feedback. Once complete, the STEEP prototype served for system validation, during which domain experts manually validated the system in terms of clinical correctness and applicability. Following the system validation, the team successfully integrated the prototype into VUMC's existing suite of CISs and performed integration testing and a complete clinical evaluation in a live environment. At the time of writing, the clinical trial of the STEEP system is in progress [240].

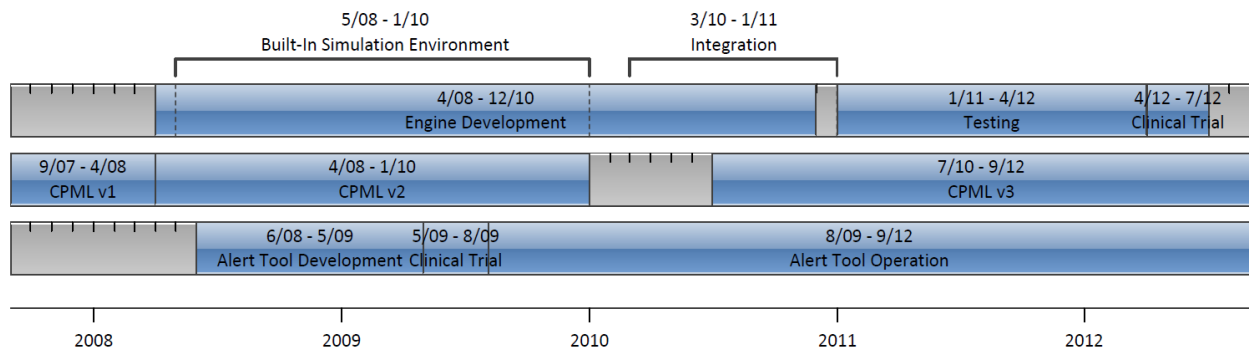


Figure 55 - STEEP project timeline

## APPENDIX C:

### ALGORITHM FOR TEMPLATE-BASED MODEL TRANSFORMATION

Algorithm 2 shows the detailed steps for transforming the GME-based CPML models (captured as MGA files) into executable Matlab models (MDL files) using the model transformation tool chain developed as part of STEEP:

Algorithm 2 - Detailed template-based model transformation process

ID	Step	Description
1.	Run the MGA2XML <sup>100</sup> transformation	Convert the CPML models captured by a GME model file (.MGA) to the input of the behavioral generator (which is the same configuration file (.XML) the EE uses to configure its behavior)
1.1.	Open MGA file	
1.2.	Traverse models and build a data structure from CPML's concepts	<ul style="list-style-type: none"> <li>• Medical Library</li> <li>• Orderables</li> <li>• Protocols</li> </ul>
1.3.	Generate XML	
2.	Generate the SF behavioral model	This will be a Matlab file (.MDL) based on the the input file created in the previous step
2.1.	Run the XML2M&DOT transformation	Generate documentation files and the Matlab script (.M), built from Matlab's API commands, that will create the SF-based behavioral model
2.1.1.	Get all inputs	<ul style="list-style-type: none"> <li>• Link to the XML version of the GME models</li> <li>• Link to the SL Matlab template file (.MDL) that contains definitions for Protocols, Processes and Activities, as well as other execution-specific components</li> <li>• Link to the Main M-script generator string template file (.ST)</li> <li>• Link to the Graphviz executable to generate a graphical representation (.PNG) from a generated document file (.DOT)</li> </ul>
2.1.2.	Traverse models	
2.1.2.1.	Convert XML	Parse source CPML model file using a generated schema definition <sup>101</sup> file in order to be able to travers models in memory

<sup>100</sup> This transformation was built by Andras Nadas from the STEEP project.

<sup>101</sup> Run the Microsoft XML Schema Definition tool (Xsd.exe) [252] to generate common language runtime classes that correspond to and are generated from the schema definition (.XSD) of the EE configuration files

2.1.2.2.	Generate output for each Protocol	
2.1.2.2.1.	Create new GraphvizContainmentHierarchy	Placeholder for DOT-style graph description
2.1.2.2.2.	Create new InstanceOfStringTemplate	Placeholder for string template and attributes that will create the M-script
2.1.2.2.3.	Record model-independent attributes of main template	String template parameters such as paths, creation time, etc.
2.1.2.2.4.	Record model-specific attributes	
2.1.2.2.4.1.	Traverse MedicalLibrary	Creation of data inputs for the protocol, specifically Labs and Vitals
2.1.2.2.4.2.	Traverse Protocol	
2.1.2.2.4.2.1.	Record child-independent parameters of Protocol	E.g. Protocol name
2.1.2.2.4.2.2.	Traverse Processes	
2.1.2.2.4.2.2.1.	Record child-independent parameters of Process	Including name, id, type, parent id, number of children, list of condition expressions (initially active, optional, repeatable, entry, reentry, skip, goal, fail, cancel, terminate)
2.1.2.2.4.2.2.2.	Traverse children of the selected (child) Process	
2.1.2.2.4.2.2.2.1.	Evaluate contents of Process	Make sure that this Process is either a (1) Process or an (2) Activity container (not both)
2.1.2.2.4.2.2.2.2.	IF (1) THEN Traverse child Processes of the selected Process	
2.1.2.2.4.2.2.2.3.	IF (2) THEN Traverse child Activities of the selected Process	
2.1.2.2.4.3.	Record child-dependent parameters of Protocol	I.e. number of children
2.1.2.2.5.	Write to destination files	
2.1.2.2.5.1.	Generate M-script by printing the InstanceOfStringTemplate into a file	The included StringTemplate [253] library will use the recorded attributes of the InstanceOfStringTemplate to recursively replace special placeholders in the template files
2.1.2.2.5.1.1.	Invoke template for a Protocol	Create file based on m_template.st using attributes for configuring the generation (e.g. verbose mode, overwrite existing, source file locations, real-time simulation, SF layout type) and attributes of Protocol (e.g. name)
2.1.2.2.5.1.1.1.	Invoke template for Vitals	Append to previous script file based on m_template_medGroup_vital.st using attributes of Vitals
2.1.2.2.5.1.1.2.	Invoke template for Labs	Append to previous script file based on m_template_medGroup_lab.st using attributes of Labs
2.1.2.2.5.1.1.3.	Invoke template for Processes	Append to previous script file based on m_template_process.st using attributes of Processes

2.1.2.2.5.1.1.4.	Invoke template for Activities	Append to previous script file based on m_template_activity.st using attributes of Activities
2.1.2.2.5.2.	Generate an alternative visualization using Graphviz	<ul style="list-style-type: none"> <li>• Generate DOT file based on GraphvizContainmentHierarchy</li> <li>• Generate PNG file using the DOT file and Graphviz</li> </ul>
2.2.	Perform the M2MDL transformation	Generate MDL by running M-script in Matlab
2.2.1.	Set script parameters	<ul style="list-style-type: none"> <li>• Generation configuration parameters (e.g. verbose mode, real-time simulation)</li> <li>• Execution-specific parameters that are not model-based (e.g. default shelf-life for medications)</li> <li>• Layout algorithm patterns (e.g. spacing, sizing of states in SF)</li> </ul>
2.2.2.	Get all inputs	<p>Load custom library elements:</p> <ul style="list-style-type: none"> <li>• Semantics template library (e.g. Protocol, Process, Activity, etc.)</li> <li>• Simulation controller library (Real-time pacer)</li> </ul>
2.2.3.	Create basic components	<ul style="list-style-type: none"> <li>• Simulink system (for simulation of the environment)</li> <li>• SF chart (for capturing the logic of the Protocol)</li> <li>• SF state (instance of the Protocol template)</li> </ul>
2.2.4.	Create data input channels for Protocol SF	<ul style="list-style-type: none"> <li>• Create data input channels for the list of vitals</li> <li>• Create data input channels for the list of lab values</li> <li>• Bundle created inputs with a multiplexer</li> <li>• Resize, arrange elements</li> </ul>
2.2.5.	Create data and event inputs in Simulink	
2.2.5.1.	IF real-time (simulation time-based) execution model THEN	<ul style="list-style-type: none"> <li>• Create a protocol controller for user (from library)</li> <li>• Add real-time pacer (from library)</li> </ul>
2.2.5.2.	IF predefined (scenario-based) execution THEN	Read scenario input from an Excel file
2.2.6.	Calculate estimated model creation time	Because the mode generation time is dependent on the size of the model (polynomial increase)
2.2.7.	Create Processes	Do this for all Processes
2.2.7.1.	Create a copy of the Process SF template	<ul style="list-style-type: none"> <li>• Establish ID</li> <li>• Rename template</li> <li>• Copy to main SF</li> <li>• Reposition (using the layout algorithm)</li> </ul>

2.2.7.2.	Create and modify events and variables	<ul style="list-style-type: none"> <li>• Replace string templates (e.g. all condition expressions<sup>102</sup>)</li> <li>• Add value to constants (e.g. number of children, optionality)</li> <li>• Rename protocol template variables (e.g. replace all occurrences of “EnId” to the unique id of the Process, and “Pald” to the unique id of the Process’s parent in the containment hierarchy)</li> <li>• Move events intended to be global but defined and contained locally by the template-based Process to the level of the container SF chart</li> </ul>
2.2.7.3.	IF verbose mode THEN append log	
2.2.8.	Create Activities	Do this for all Activities
2.2.8.1.	Create a copy of the Activity SF template	<ul style="list-style-type: none"> <li>• Establish ID</li> <li>• Rename template</li> <li>• Copy to main SF</li> <li>• Reposition (using the layout algorithm)</li> </ul>
2.2.8.2.	Create and modify events and variables	<ul style="list-style-type: none"> <li>• Replace string templates (e.g. starting point)</li> <li>• Add value to constants (e.g. ability to repeat the action)</li> <li>• Rename protocol template variables (e.g. replace all occurrences of “EnId” to the unique id of the Process, and “Pald” to the unique id of the Process’s parent in the containment hierarchy)</li> <li>• Move events intended to be global but defined and contained locally by the template-based Activity to the level of the container SF chart</li> </ul>
2.2.8.3.	IF verbose mode THEN append log	
2.2.8.4.	IF real-time (simulation time-based) execution model THEN	Create inputs for Activities (from library) Position them using the layout algorithm Connect them to the event input of the SF chart
2.2.9.	Set simulation parameters	E.g. stop time, solver type, etc.

---

<sup>102</sup> While these conditions are manually translated from our expression language to Matlab’s expression language, an automated translation would be possible.

## BIBLIOGRAPHY

1. Institute of Medicine (U.S.) Committee on Quality of Health Care in America. Crossing the quality chasm: a new health system for the 21st century. National Academies Press; 2001.
2. Kaushal R, Jha AK, Franz C, Glaser J, Shetty KD, Jaggi T, et al. Return on investment for a computerized physician order entry system. *J Am Med Inform Assoc*. 2006 Jun;13(3):261–266.
3. Shabo A. A global socio-economic-medico-legal model for the sustainability of longitudinal electronic health records. Part 1. *Methods Inf Med*. 2006;45(3):240–245.
4. Simon SJ, Simon SJ. An examination of the financial feasibility of Electronic Medical Records (EMRs): a case study of tangible and intangible benefits. *International Journal of Electronic Healthcare*. 2006;2(2):185 – 200.
5. Menachemi N, Brooks R. Reviewing the Benefits and Costs of Electronic Health Records and Associated Patient Safety Technologies. *Journal of Medical Systems*. 2006 Jun 1;30(3):159–168.
6. Lo HG, Newmark LP, Yoon C, Volk LA, Carlson VL, Kittler AF, et al. Electronic health records in specialty care: a time-motion study. *J Am Med Inform Assoc*. 2007 Oct;14(5):609–615.
7. Pizziferri L, Kittler AF, Volk LA, Honour MM, Gupta S, Wang S, et al. Primary care physician time utilization before and after implementation of an electronic health record: a time-motion study. *J Biomed Inform*. 2005 Jun;38(3):176–188.
8. Hunt DL, Haynes RB, Hanna SE, Smith K. Effects of computer-based clinical decision support systems on physician performance and patient outcomes: a systematic review. *JAMA*. 1998 Oct 21;280(15):1339–1346.
9. Silver MR, Lusk R. Patient safety: a tale of two systems. *Qual Manag Health Care*. 2002;10(2):12–22.
10. Frank L, Galanos H, Penn S, Wetz HF. Using BPI and emerging technology to improve patient safety. *J Healthc Inf Manag*. 2004;18(1):65–71.
11. Committee on Engaging the Computer Science Research Community in Health Care Informatics, National Research Council, Lin H, Stead WW. *Computational Technology for Effective Health Care: Immediate Steps and Strategic Directions*. National Academies Press; 2009.
12. The Free Dictionary. Cognitive function definition in the Medical dictionary [Internet]. 2010 [cited 2010 Dec 14]; Available from: <http://medical-dictionary.thefreedictionary.com/cognitive+function>
13. Arshad Ahmed. The Healthcare Battle for a Clinical Decision Support System (CDSS) Control Point Begins [Internet]. *Scientia Advisors*. 2010 Dec 16 [cited 2012 Apr 10]; Available from: <http://www.scientiaadv.com/blog/2010/12/16/the-healthcare-battle-for-a-clinical-decision-support-system-cdss-control-point-begins/>

14. Hayward R. Clinical decision support tools: Do they support clinicians? *Canadian Medical Association Journal*. 2004;170(10):66–85.
15. Best Price Computers. DSS - Decision Support Systems Explained and Defined [Internet]. 2010 [cited 2010 May 10]; Available from: <http://www.bestpricecomputers.co.uk/glossary/decision-support-systems.htm>
16. OpenClinical.org. CDSS (Clinical Decision Support System) [Internet]. 2010 [cited 2010 Mar 18]; Available from: <http://www.openclinical.org/dss>
17. Field MJ, Lohr KN, Institute of Medicine (U.S.) Committee on Clinical Practice Guidelines. *Guidelines for clinical practice: from development to use*. National Academies Press; 1992.
18. Committee on Clinical Practice Guidelines, Institute of Medicine. *Guidelines for Clinical Practice: From Development to Use*. Washington, D.C.: The National Academies Press; 1992.
19. Kelly DC, Manguno-Mire G. Commentary: Helling V. Carey, Caveat Medicus. *J Am Acad Psychiatry Law*. 2008 Sep 1;36(3):306–309.
20. OpenClinical.org. CPG (Clinical Practice Guidelines) [Internet]. 2009 [cited 2009 Dec 3]; Available from: <http://www.openclinical.org/guidelines.html>
21. Sackett DL, Rosenberg WMC, Gray JAM, Haynes RB, Richardson WS. Evidence based medicine: what it is and what it isn't. *BMJ*. 1996 Jan 13;312(7023):71–72.
22. Shapiro NI, Howell M, Talmor D. A blueprint for a sepsis protocol. *Acad Emerg Med*. 2005 Apr;12(4):352–359.
23. Berner MM, Rütther A, Stieglitz RD, Berger M. The concept of “evidence-based medicine” in psychiatry. A path to a more rational psychiatry? *Nervenarzt*. 2000 Mar;71(3):173–180.
24. Jessup M, Abraham WT, Casey DE, Feldman AM, Francis GS, Ganiats TG, et al. 2009 Focused Update: ACCF/AHA Guidelines for the Diagnosis and Management of Heart Failure in Adults: A Report of the American College of Cardiology Foundation/American Heart Association Task Force on Practice Guidelines Developed in Collaboration With the International Society for Heart and Lung Transplantation. *J Am Coll Cardiol*. 2009 Apr 14;53(15):1343–1382.
25. Shapiro NI, Howell MD, Talmor D, Lahey D, Ngo L, Buras J, et al. Implementation and outcomes of the Multiple Urgent Sepsis Therapies (MUST) protocol. *Crit. Care Med*. 2006 Apr;34(4):1025–1032.
26. Isern D, Moreno A. Computer-based execution of clinical guidelines: A review. *Int J Med Inform*. 2008 Dec;77(12):787–808.
27. National Guideline Clearinghouse (NGC) of U.S. Guideline submission [Internet]. 2009 Dec 2 [cited 2009 Dec 3]; Available from: <http://www.guideline.gov/>

28. National Guideline Clearinghouse (NGC) of U.S. How to Submit Guidelines - Template of Guideline Attributes [Internet]. 2009 Dec 2 [cited 2009 Dec 3]; Available from: <http://www.guideline.gov/submit/submit.aspx#kit>
29. Shiffman RN, Shekelle P, Overhage JM, Slutsky J, Grimshaw J, Deshpande AM. Standardized Reporting of Clinical Practice Guidelines: A Proposal from the Conference on Guideline Standardization. *Annals of Internal Medicine*. 2003;139(6):493–498.
30. Cabana MD, Rand CS, Powe NR, Wu AW, Wilson MH, Abboud PA, et al. Why don't physicians follow clinical practice guidelines? A framework for improvement. *JAMA*. 1999 Oct 20;282(15):1458–1465.
31. Asgaard Project. Asgaard/Asbru Project [Internet]. 2011 [cited 2011 Jan 25]; Available from: <http://www.asgaard.tuwien.ac.at/about/project.html>
32. Dube K, Wu B. A generic approach to computer-based Clinical Practice Guideline management using the ECA Rule paradigm and active databases. *International Journal of Technology Management*. 2009;47(1/2/3):75 – 95.
33. Shahar Y, Miksch S, Johnson P. The ASGAARD Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines [Internet]. In: *Artificial Intelligence in Medicine*. 1998. p. 29–51. Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.916>
34. Hammond WE. Clinical Decision Support [Internet]. 2010 [cited 2010 Dec 11]; Available from: [http://www.google.com/url?sa=t&source=web&cd=2&ved=0CB0QFjAB&url=http%3A%2F%2Fwww.hl7.org%2Fdocumentcenter%2Fpublic%2Fcalendarofevents%2Fhimss%2F2010%2Fpresentations%2FHIMSS%2520CDSS%2520Hammond.pdf&rct=j&q=arden%20syntax%202.7%20pdf&ei=PfYCTbHMHMH\\_lgfOy4CDCA&usg=AFQjCNFG5Sm99TBshrLI\\_KrJ2ZKEKRCKmw&sig2=symZtqs3WPKCO4W472wB8A](http://www.google.com/url?sa=t&source=web&cd=2&ved=0CB0QFjAB&url=http%3A%2F%2Fwww.hl7.org%2Fdocumentcenter%2Fpublic%2Fcalendarofevents%2Fhimss%2F2010%2Fpresentations%2FHIMSS%2520CDSS%2520Hammond.pdf&rct=j&q=arden%20syntax%202.7%20pdf&ei=PfYCTbHMHMH_lgfOy4CDCA&usg=AFQjCNFG5Sm99TBshrLI_KrJ2ZKEKRCKmw&sig2=symZtqs3WPKCO4W472wB8A)
35. MedlinePlus Medical Encyclopedia. Contraindications [Internet]. 2012 [cited 2012 Aug 7]; Available from: <http://www.nlm.nih.gov/medlineplus/ency/article/002314.htm>
36. Sutton DR, Taylor P, Earle K. Evaluation of PROforma as a language for implementing medical guidelines in a practical context. *BMC Med Inform Decis Mak*. 2006;6:20.
37. Rozenberg G, Salomaa A, editors. *Handbook of Formal Languages: Vols. 1 - 3*. 1st ed. Springer; 2004.
38. Chomsky N. Three models for the description of language. *Information Theory, IRE Transactions on*. 1956 Sep 6;2(3):113–124.
39. Object Management Group (OMG). UML (Unified Modeling Language) [Internet]. 2010 [cited 2010 Nov 19]; Available from: <http://www.uml.org/>
40. OASIS. WSBPEL (Web Services Business Process Execution Language) [Internet]. 2010 [cited 2010 Nov 19]; Available from: [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel)



41. Kelly S, Tolvanen J-P. Domain-Specific Modeling: Enabling Full Code Generation. Wiley-Interscience; 2008.
42. Sutton DR, Fox J. The syntax and semantics of the PROforma guideline modeling language. *J Am Med Inform Assoc.* 2003 Oct;10(5):433–443.
43. Miller RA. Medical Diagnostic Decision Support Systems—Past, Present, And Future. *J Am Med Inform Assoc.* 1994 Jan;1(1):8–27.
44. Stead WW, Hammond WE. Computer-based medical records: the centerpiece of TMR. *MD Comput.* 1988 Oct;5(5):48–62.
45. Perry CA. Knowledge bases in medicine: a review. *Bull Med Libr Assoc.* 1990 Jul;78(3):271–282.
46. Weiss S, Kulikowski CA, Safir A. Glaucoma consultation by computer. *Computers in Biology and Medicine.* 1978 Jan;8(1):25–40.
47. Pauker SG, Gorry GA, Kassirer JP, Schwartz WB. Towards the simulation of clinical cognition: Taking a present illness by computer. *The American Journal of Medicine.* 1976 Jun;60(7):981–996.
48. Yorkshire Centre for Health Informatics, University of Leeds. Welcome to AAPHelp - Decision Support in Acute Abdominal Pain [Internet]. 2010 [cited 2010 Nov 24];Available from: <http://www.aaphelp.leeds.ac.uk/aaphelp/>
49. de Dombal FT, Leaper DJ, Staniland JR, McCann AP, Horrocks JC. Computer-aided Diagnosis of Acute Abdominal Pain. *BMJ.* 1972 Apr 1;2(5804):9–13.
50. Miller RA. CV for Randolph A. Miller [Internet]. 2010 [cited 2010 Nov 24];Available from: <http://www.mc.vanderbilt.edu/dbmi/miller/>
51. Masarie Jr. FE, Miller RA, Myers JD. INTERNIST-I properties: Representing common sense and good medical practice in a computerized medical knowledge base. *Comput. Biomed. Res.* 1985 Oct;18(5):458–479.
52. Miller RA, Masarie FE. Use of the Quick Medical Reference (QMR) program as a tool for medical education. *Methods Inf Med.* 1989 Nov;28(4):340–345.
53. Davis R, King J. An Overview of Production Systems. Stanford AI Lab; 1975.
54. Papamarkos G, Poulouvassilis A, Poulouvassilis R, Wood PT. Event-Condition-Action Rule Languages for the Semantic Web. In: Workshop on Semantic Web and Databases. 2003. p. 309–327.
55. Fisher JR. MYCIN - Medical Infromatics Class [Internet]. 2006 [cited 2010 Nov 29];Available from: <http://neamh.cns.uni.edu/MedInfo/mycin.html>
56. Lazarevic Z. Mycin Expert System - A Ruby Implementation with Examples [Internet]. 2010 [cited 2010 Nov 29];Available from: <http://lazax.com/software/Mycin/mycin.html>

57. Buchanan BG. Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project. Addison-Wesley; 1984.
58. Van Melle WJ. A domain-independent system that aids in constructing knowledge-based consultation programs. 1980;
59. Novak GSJ. TMYCIN Expert System Tool. 1988.
60. Novak GSJ. TMYCIN Expert System Tool [Internet]. 2010 [cited 2010 Nov 29];Available from: <http://www.cs.utexas.edu/users/novak/tmycin.html>
61. Aikins JS, Kunz JC, Shortliffe EH, Fallat RJ. PUFF: An expert system for interpretation of pulmonary function data. *Comput. Biomed. Res.* 1983 Jun;16(3):199–208.
62. Patil RS, Szolovits P, Schwartz WB. Causal understanding of patient illness in medical diagnosis. In: *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2.* Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc.; 1981. p. 893–899.
63. Patil RS, Szolovits P, Schwartz WB. Modeling Knowledge of the Patient in Acid-Base and Electrolyte Disorders. In: *Artificial Intelligence in Medicine.* Westview Press, Boulder, Colorado: 1982.
64. Shortliffe EH, Scott AC, Bischoff MB, Campbell AB, Van Melle W, Jacobs CD. ONCOCIN: An expert system for oncology protocol management [Internet]. In: *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project.* 1984 [cited 2010 Nov 30]. Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.80.5446>
65. Kahn MG, Ferguson JC, Shortliffe EH, Fagan LM. Representation and Use of Temporal Information in ONCOCIN. *Proc Annu Symp Comput Appl Med Care.* 1985 Nov 13;:172–176.
66. Fagan LM. *Medical Informatics: Computer Applications in Health Care and Biomedicine.* 2nd ed. Springer; 2003.
67. MGH Laboratory of Computer Science, Barnett O. DXplain Project Website [Internet]. 2010 [cited 2010 Nov 30];Available from: <http://lcs.mgh.harvard.edu/projects/dxplain.html>
68. Miller R, Masarie FE, Myers JD. Quick medical reference (QMR) for diagnostic assistance. *MD Comput.* 1986 Oct;3(5):34–48.
69. Linton AM. QMR (Quick Medical Reference). *Bull Med Libr Assoc.* 1993 Jul;81(3):347–349.
70. Greenes RA, Peleg M, Boxwala A, Tu S, Patel V, Shortliffe EH. Sharable computer-based clinical practice guidelines: rationale, obstacles, approaches, and prospects. *Stud Health Technol Inform.* 2001;84(Pt 1):201–205.
71. Mathe J, Martin J, Miller P, Ledeczki A, Weavind L, Nadas A, et al. A Model-Integrated, Guideline-Driven, Clinical Decision-Support System. *IEEE Softw.* 2009 Aug;26(4):54–61.

72. Gregory V. G. O'Dowd. Building a Medical Vocabulary: A Guide for Medical Students [Internet]. 2008. Available from: [http://hikumano.hama-med.ac.jp/dspace/bitstream/10271/48/1/kiyo22\\_2.pdf](http://hikumano.hama-med.ac.jp/dspace/bitstream/10271/48/1/kiyo22_2.pdf)
73. United States Department of Defense. Technology Readiness Assessment (TRA) Guidance. 2011.
74. Jackson E, Thibodeaux R, Porter J, Sztipanovits J. Semantics of Domain Specific Modeling Languages. In: Model-Based Design of Heterogeneous Embedded Systems. CRC Press; 2009. p. 437–486.
75. Balasubramanian D. Behavioral Semantics of Modeling Languages: A Pragmatic Approach [Internet]. 2011; Available from: <http://etd.library.vanderbilt.edu/available/etd-04082011-145057/unrestricted/DanielDissertation.pdf>
76. Savage JE. Models of Computation: Exploring the Power of Computing. 2008.
77. Bondi AB. Characteristics of scalability and their impact on performance [Internet]. In: Proceedings of the 2nd international workshop on Software and performance. New York, NY, USA: ACM; 2000. p. 195–203. Available from: <http://doi.acm.org/10.1145/350391.350432>
78. Leong TY, Kaiser K, Miksch S. Free and open source enabling technologies for patient-centric, guideline-based clinical decision support: a survey. Yearb Med Inform. 2007;:74–86.
79. Pryor TA, Hripcsak G. The arden syntax for medical logic modules. J Clin Monit Comput. 1993 Nov;10(4):215–224.
80. Hripcsak G, Ludemann P, Pryor TA, Wigertz OB, Clayton PD. Rationale for the Arden Syntax. Comput. Biomed. Res. 1994 Aug;27(4):291–324.
81. HL7. Arden Syntax [Internet]. 2010 [cited 2010 Dec 21]; Available from: <http://www.hl7.org/implement/standards/ardensyntax.cfm>
82. Mulyar N, van der Aalst WMP, Peleg M. A pattern-based analysis of clinical computer-interpretable guideline modeling languages. J Am Med Inform Assoc. 2007 Dec;14(6):781–787.
83. ASTM.org. American Society for Testing and Materials (ASTM) [Internet]. 2010 [cited 2010 Dec 11]; Available from: <http://www.astm.org/>
84. Health Level Seven International. About Health Level Seven International [Internet]. 2012 [cited 2012 Apr 10]; Available from: <http://www.hl7.org/about/index.cfm>
85. ANSI.org. American National Standards Institute (ANSI) [Internet]. 2010 [cited 2010 Dec 11]; Available from: <http://www.ansi.org/>
86. OpenClinical.org. Arden Syntax [Internet]. 2005 Mar 30 [cited 2010 Dec 11]; Available from: [http://www.openclinical.org/gmm\\_ardensyntax.html](http://www.openclinical.org/gmm_ardensyntax.html)
87. HL7. ANSI Approved Standards [Internet]. 2012 [cited 2012 Apr 12]; Available from: <http://www.hl7.org/implement/standards/ansiapproved.cfm>

88. Scherpbier HJ. CORBAmed RFI 3 Response - Clinical Decision Support. 1997.
89. Kyprianides P. Arden Syntax [Internet]. Medical Informatics class by Kevin C. O’Kane. 2006 [cited 2010 Dec 12];Available from: [http://neamh.cns.uni.edu/MedInfo/arden\\_syntax.html](http://neamh.cns.uni.edu/MedInfo/arden_syntax.html)
90. Sailors RM. ArdenML: The Arden Syntax Markup Language (or Arden Syntax: It’s Not Just Text Any More!). AMIA Annu Symp Proc. 2001;:1016–1016.
91. Sukil Kim, Peter J. Haug, Roberto A. Rocha, Inyoung Choi. Modeling the Arden Syntax for medical decisions in XML. Int J Med Inform. 2008 Oct 1;77(10):650–656.
92. Saltz J, Niland J, Payne P, Shah H, Stahl D. Rules Engine Technologies Across caBIG Workspaces [Internet]. Strategic Planning Workspace; 2007 [cited 2012 Apr 12]. Available from: [https://cabig.nci.nih.gov/community/archive/caBIG\\_StrategicPlanning/caBIG\\_Rules\\_Engine\\_WP\\_v4.doc](https://cabig.nci.nih.gov/community/archive/caBIG_StrategicPlanning/caBIG_Rules_Engine_WP_v4.doc)
93. Kuhn RA, Reider RS. A C++ framework for developing medical logic modules and an Arden Syntax compiler. Computers in Biology and Medicine. 1994 Sep;24(5):365–370.
94. Gietzelt M, Goltz U, Grunwald D, Lochau M, Marschollek M, Song B, et al. Arden2ByteCode: A one-pass Arden Syntax compiler for service-oriented decision support systems based on the OSGi platform. Computer Methods and Programs in Biomedicine. 2012 May;106(2):114–125.
95. Sailors RM. MLM Builder: An Integrated Suite for Development and Maintenance of Arden Syntax Medical Logic Modules. AMIA Annu Fall Symp Proc. 1997;:996–996.
96. Wikipedia.org. Medical Logic Module (MLM) [Internet]. 2010 [cited 2010 Dec 11];Available from: [http://en.wikipedia.org/wiki/Medical\\_logic\\_module](http://en.wikipedia.org/wiki/Medical_logic_module)
97. OpenClinical.org. Syntax and Semantics of PROforma [Internet]. 2010 [cited 2010 May 26];Available from: [http://www.openclinical.org/gmm\\_proforma.html](http://www.openclinical.org/gmm_proforma.html)
98. Cossac.org. PROforma project website [Internet]. 2009 [cited 2009 Nov 10];Available from: <http://www.cossac.org/technologies/proforma>
99. Sutton D, Fox J. Syntax and Semantics of PROforma. 2003.
100. Fox J, Thomson R. Decision support and disease management: a logic engineering approach. IEEE Trans Inf Technol Biomed. 1998 Dec;2(4):217–228.
101. Fox J, Das S. Safe and Sound: Artificial Intelligence in Hazardous Applications [Internet]. AAAI Press; 2000 [cited 2011 Jan 14]. Available from: <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0262062119>
102. Steele R, Fox J. PROforma tutorial: Introduction to the PROforma language and application development using Tallis. 2002.
103. InferMed. Arezzo - Clinical Decision Support [Internet]. 2010 [cited 2010 May 26];Available from: [http://www.infermed.com/index.php/arezzo/arezzo\\_technology](http://www.infermed.com/index.php/arezzo/arezzo_technology)

104. Cossac.org. Tallis | COSSAC project website [Internet]. 2009 [cited 2009 Nov 10];Available from: <http://www.cossac.org/tallis>
105. Hederman L, Smutek D, Wade V, Knape T. Representing Clinical Guidelines in UML: A Comparative Study. Health Data in the Information Society: Proceedings of Mie2002 [Internet]. 2002 [cited 2009 Sep 14];Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.3683>
106. Ohno-Machado L, Gennari JH, Murphy SN, Jain NL, Tu SW, Oliver DE, et al. The GuideLine Interchange Format: A Model for Representing Guidelines. *J Am Med Inform Assoc.* 1998 Jul 1;5(4):357–372.
107. OpenClinical.org. GLIF [Internet]. 2010 [cited 2010 Apr 13];Available from: [http://www.openclinical.org/gmm\\_glif.html](http://www.openclinical.org/gmm_glif.html)
108. Peleg M, Boxwala AA, Ogunyemi O, Zeng Q, Tu S, Lacson R, et al. GLIF3: the evolution of a guideline representation format. *AMIA Annu Symp Proc.* 2000;:645–649.
109. Boxwala AA, Peleg M, Tu S, Ogunyemi O, Zeng QT, Wang D, et al. GLIF3: a representation format for sharable computer-interpretable clinical practice guidelines. *J Biomed Inform.* 2004 Jun;37(3):147–161.
110. Wang D, Shortliffe EH. GLEE - A model-driven execution system for computer-based implementation of clinical practice guidelines. *AMIA Annu Symp Proc.* 2002;:855–859.
111. Choi J, Currie LM, Wang D, Bakken S. Encoding a clinical practice guideline using guideline interchange format: a case study of a depression screening and management guideline. *Int J Med Inform.* 2007 Oct;76 Suppl 2:S302–307.
112. OpenClinical.org. GELLO [Internet]. 2011 [cited 2011 Jan 24];Available from: [http://www.openclinical.org/gmm\\_gello.html](http://www.openclinical.org/gmm_gello.html)
113. InterMed Collaboratory. GLIF.ORG [Internet]. 2009 [cited 2009 Nov 10];Available from: [http://www.glif.org/glif\\_main.html](http://www.glif.org/glif_main.html)
114. Stanford Center for Biomedical Informatics Research. Protégé website [Internet]. The Protégé Ontology Editor and Knowledge Acquisition System. 2011 [cited 2011 Jan 28];Available from: <http://protege.stanford.edu/>
115. Noy NF, Fergerson RW, Musen MA. The knowledge model of Protege-2000: combining interoperability and flexibility. *Proceedings EKAW '00 Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management.* 2001;:17–32.
116. Wang D, Peleg M, Tu SW, Boxwala AA, Ogunyemi O, Zeng Q, et al. Design and implementation of the GLIF3 guideline execution engine. *J Biomed Inform.* 2004 Oct;37(5):305–318.
117. Sordo M, Ogunyemi O, Boxwala AA, Greenes RA. GELLO: an object-oriented query and expression language for clinical decision support. *AMIA Annu Symp Proc.* 2003;:1012.

118. HL7. V3 Rules/GELLO [Internet]. 2011 [cited 2011 Jan 28];Available from: <http://www.hl7.org/implement/standards/v3gello.cfm>
119. Jenders RA. What I Did on my (Summer) Holiday: International Clinical Decision Support Standards [Internet]. 2004 Feb 16;Available from: [www.hl7.org.au/docs/Arden-Syntax-Workshop\\_MEL.pdf](http://www.hl7.org.au/docs/Arden-Syntax-Workshop_MEL.pdf)
120. Object Management Group (OMG). OCL (Object Constraint Language) [Internet]. 2011 [cited 2011 Jan 28];Available from: [http://www.omg.org/technology/documents/modeling\\_spec\\_catalog.htm#OCL](http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL)
121. OpenClinical.org. Asbru [Internet]. 2011 [cited 2011 Feb 4];Available from: [http://www.openclinical.org/gmm\\_asbru.html](http://www.openclinical.org/gmm_asbru.html)
122. Fuchsberger C, Hunter J, McCue P. Testing Asbru Guidelines and Protocols for Neonatal Intensive Care [Internet]. In: Artificial Intelligence in Medicine. 2005 [cited 2009 Aug 13]. p. 101–110.Available from: [http://dx.doi.org/10.1007/11527770\\_14](http://dx.doi.org/10.1007/11527770_14)
123. Asgaard Project. Example protocols written in Asbru [Internet]. 2011 [cited 2011 Feb 5];Available from: [http://www.asgaard.tuwien.ac.at/plan\\_representation/protocols.html](http://www.asgaard.tuwien.ac.at/plan_representation/protocols.html)
124. Seyfang A, Kosara R, Miksch S. Asbru Reference Manual [Internet]. 2002 Jan;Available from: [http://www.asgaard.tuwien.ac.at/asbru\\_7\\_3/asbru\\_7.3\\_reference.pdf](http://www.asgaard.tuwien.ac.at/asbru_7_3/asbru_7.3_reference.pdf)
125. Balsler M, Duelli C, Reif W, Schmitt J, Balsler M, Duelli C, et al. Formal Semantics of Asbru [Internet]. 2006 Jun [cited 2011 Feb 14];Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.147.1845>
126. Asgaard Project. AsbruView [Internet]. 2011 [cited 2011 Jan 25];Available from: <http://www.asgaard.tuwien.ac.at/asbruvie/>
127. Kosara R, Miksch S, Shahar Y, Johnson P. AsbruView: Capturing Complex, Time-oriented Plans - Beyond Flow-Charts [Internet]. In: 2nd Workshop on Thinking with Diagrams 1998 (TwD-98). University of Wales; 1998 [cited 2011 Feb 15]. p. 119–126.Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.3770>
128. OpenClinical.org. AsbruView: Integrated Visualization of Computerized Protocols and Temporal Patient Data [Internet]. 2011 [cited 2011 Jan 25];Available from: [http://www.openclinical.org/dld\\_asbruvie.html](http://www.openclinical.org/dld_asbruvie.html)
129. Votruba P. DELT/A (Document Exploration and Linking Tool (with Addons)) [Internet]. 2011 [cited 2011 Feb 15];Available from: <http://www.asgaard.tuwien.ac.at/~peter/DELTA/index.html>
130. Votruba P, Miksch S, Seyfang A, Kosara R. Tracing the formalization steps of textual guidelines. *Stud Health Technol Inform*. 2004;101:172–176.
131. Aigner W, Miksch S. CareVis: Integrated visualization of computerized protocols and temporal patient data. *Artif Intell Med*. 2006 Jul;37(3):203–218.

132. OpenClinical.org. CareVis [Internet]. 2011 [cited 2011 Feb 15];Available from: [http://www.openclinical.org/dld\\_carevis.html](http://www.openclinical.org/dld_carevis.html)
133. OpenClinical.org. DeGeL [Internet]. 2011 [cited 2011 Feb 15];Available from: [http://www.openclinical.org/gmm\\_degel.html](http://www.openclinical.org/gmm_degel.html)
134. Shahar Y, Shalom E, Mayaffit A, Young O, Galperin M, Martins S, et al. A distributed, collaborative, structuring model for a clinical-guideline digital-library. *AMIA Annu Symp Proc.* 2003;:589–593.
135. Shahar Y, Young O, Shalom E, Galperin M, Mayaffit A, Moskovitch R, et al. A framework for a distributed, hybrid, multiple-ontology clinical-guideline library, and automated guideline-support tools. *J Biomed Inform.* 2004 Oct;37(5):325–344.
136. Aigner W, Miksch S. Communicating the logic of a treatment plan formulated in Asbru to domain experts. *Stud Health Technol Inform.* 2004;101:1–15.
137. Oracle. Oracle BPEL Process Manager [Internet]. 2011 [cited 2011 Feb 17];Available from: <http://www.oracle.com/technetwork/middleware/bpel/overview/index.html>
138. Asgaard Project. Asbru Plan Execution [Internet]. 2011 [cited 2011 Feb 15];Available from: [http://www.asgaard.tuwien.ac.at/exec\\_unit/index.html](http://www.asgaard.tuwien.ac.at/exec_unit/index.html)
139. Fuchsberger C, Miksch S. Asbru’s Execution Engine: Utilizing Guidelines for Artificial Ventilation of Newborn Infants’, Workshop on Intelligent Data Analysis. Vienna University of Technology, Institute of Software Technology and Interactive Systems [Internet]. 2002 [cited 2011 Feb 9];Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.111.4961>
140. OpenClinical.org. Asbru Interpreter [Internet]. 2011 [cited 2011 Feb 4];Available from: [http://www.openclinical.org/dld\\_asbruInterpreter.html](http://www.openclinical.org/dld_asbruInterpreter.html)
141. Seyfang A, Miksch S. Advanced Temporal Data Abstraction for Guideline Execution. *Proceedings of the Symposium on Computer-based Support for Clinical Guidelines and Protocols.* 2004;:88–102.
142. Young O, Shahar Y. The Spock System: Developing a Runtime Application Engine for Hybrid-Asbru Guidelines [Internet]. In: *Artificial Intelligence in Medicine.* Springer Berlin / Heidelberg; 2005. p. 166–170.Available from: [http://dx.doi.org/10.1007/11527770\\_25](http://dx.doi.org/10.1007/11527770_25)
143. Young O, Shahar Y, Liel Y, Lunenfeld E, Bar G, Shalom E, et al. Runtime application of Hybrid-Asbru clinical guidelines. *J Biomed Inform.* 2007 Oct;40(5):507–526.
144. Bosse T. An interpreter for clinical guidelines in Asbru [Internet]. 2001 Aug;Available from: [www.cs.vu.nl/~protoc/old/Projects/TiborThesis.pdf](http://www.cs.vu.nl/~protoc/old/Projects/TiborThesis.pdf)
145. Eichelberg M, Aden T, Riesmeier J, Dogac A, Laleci GB. A survey and analysis of Electronic Healthcare Record standards. *ACM Comput. Surv.* 2005;37(4):277–315.
146. Peleg M, Tu S, Bury J, Ciccarese P, Fox J, Greenes RA, et al. Comparing Computer-interpretable Guideline Models: A Case-study Approach. *J Am Med Inform Assoc.* 2003 Jan 1;10(1):52–68.

147. Boone K. vMR Proposal [Internet]. 2010 Jul 29; Available from: [http://wiki.hl7.org/images/7/71/HL7vMR\\_Keith\\_Boone\\_Proposal\\_v2010-07-29.doc](http://wiki.hl7.org/images/7/71/HL7vMR_Keith_Boone_Proposal_v2010-07-29.doc)
148. HL7book. HL7 version 3 [Internet]. 2012 [cited 2012 May 23]; Available from: [http://hl7book.net/index.php?title=HL7\\_version\\_3](http://hl7book.net/index.php?title=HL7_version_3)
149. American Medical Association (AMA). CPT (Current Procedural Terminology) [Internet]. 2012 [cited 2012 May 23]; Available from: <http://www.ama-assn.org/ama/pub/physician-resources/solutions-managing-your-practice/coding-billing-insurance/cpt.page>
150. World Health Organization (WHO). ICD (International Classification of Diseases) [Internet]. 2012 [cited 2012 May 23]; Available from: <http://www.who.int/classifications/icd/en/>
151. Regenstrief Institute, Inc., Indiana University. LOINC (Logical Observation Identifiers Names and Codes) [Internet]. 2012 [cited 2012 May 23]; Available from: <http://loinc.org/>
152. International Health Terminology Standards Development Organisation (IHTSDO). SNOMED CT (Systematized Nomenclature of Medicine - Clinical Terms) [Internet]. 2012 [cited 2012 May 23]; Available from: <http://www.ihtsdo.org/snomed-ct/>
153. National Library of Medicine (U.S.). UMLS (Unified Medical Language System) [Internet]. 2010 [cited 2010 Mar 17]; Available from: <http://www.nlm.nih.gov/research/umls/>
154. HL7. HL7 Standards [Internet]. 2010 [cited 2010 Dec 10]; Available from: <http://www.hl7.org/implement/standards/index.cfm?ref=nav>
155. California HealthCare Foundation. CCDP Project Overview [Internet]. 2012. Available from: <http://www.chcf.org/documents/CCDPPProjectOverview.pdf>
156. Berner ES, Webster GD, Shugerman AA, Jackson JR, Algina J, Baker AL, et al. Performance of Four Computer-Based Diagnostic Systems. *New England Journal of Medicine*. 1994;330(25):1792–1796.
157. Elkin PL, Peleg M, Lacson R, Bernstam E, Tu S, Boxwala A, et al. Toward Standardization of Electronic Guideline Representation. *Md Computing*. 2000;17(6):39–44.
158. de Clercq PA, Blom JA, Korsten HHM, Hasman A. Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artif Intell Med*. 2004 May;31(1):1–27.
159. Peleg M. Guideline representation methods: A comparison study [Internet]. 2002 [cited 2011 Mar 18]; Available from: <http://www.openclinical.org/gmmcomparison.html>
160. Tu SW, Musen MA. Representation Formalisms and Computational Methods for Modeling Guideline-Based Patient Care. In: 1st European Workshop on Computer-based Support for Clinical Guidelines and Protocols. Leipzig, Germany: 2001.
161. Wang D, Peleg M, Tu SW, Boxwala AA, Greenes RA, Patel VL, et al. Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines: a



- literature review of guideline representation models. *Int J Med Inform.* 2002 Dec 18;68(1-3):59–70.
162. Russell N, Hofstede AHM ter, Aalst WMP van der, Mulyar N. Workflow Control-Flow Patterns: A Revised View [Internet]. BPM Center; 2007. Available from: <http://workflowpatterns.com/documentation/documents/BPM-06-22.pdf>
  163. Grando MA, Glasspool DW, Fox J. Petri Nets as a Formalism for Comparing Expressiveness of Workflow-Based Clinical Guideline Languages [Internet]. In: *Business Process Management Workshops*. Springer Berlin Heidelberg; 2009 [cited 2009 Nov 10]. p. 348–360. Available from: [http://dx.doi.org/10.1007/978-3-642-00328-8\\_35](http://dx.doi.org/10.1007/978-3-642-00328-8_35)
  164. Purves IN. PRODIGY: implementing clinical guidance using computers. *Br J Gen Pract.* 1998 Sep;48(434):1552–1553.
  165. Purves IN, Sugden B, Booth N, Sowerby M. The PRODIGY project - the iterative development of the release one model. *AMIA Annu Symp Proc.* 1999;:359–363.
  166. Mark ST, Musen M.D. Ph.D MA. A Flexible Approach to Guideline Modeling. *AMIA Annu Symp Proc.* 1999;:420–424.
  167. Tu SW, Musen MA. Modeling data and knowledge in the EON guideline architecture. *Stud Health Technol Inform.* 2001;84(Pt 1):280–284.
  168. Shiffman RN, Karras BT, Agrawal A, Chen R, Marenco L, Nath S. GEM: a proposal for a more comprehensive guideline document model using XML. *J Am Med Inform Assoc.* 2000 Oct;7(5):488–498.
  169. Hajizadeh N, Kashyap N, Michel G, Shiffman RN. GEM at 10: A Decade’s Experience with the Guideline Elements Model. *AMIA Annu Symp Proc.* 2011;2011:520–528.
  170. Haschler, Skonetzki S, Gausepohl HJ, Linderkamp O, Wetter T. Evolution of the HELEN representation for managing clinical practice guidelines. *Methods Inf Med.* 2004;43(4):413–426.
  171. Tu SW, Campbell J, Musen MA. The SAGE guideline modeling: motivation and methodology. *Stud Health Technol Inform.* 2004;101:167–171.
  172. Tu SW, Campbell JR, Glasgow J, Nyman MA, McClure R, McClay J, et al. The SAGE Guideline Model: achievements and overview. *J Am Med Inform Assoc.* 2007 Oct;14(5):589–598.
  173. Guarnero A, Marzuoli M, Molino G, Terenziani P, Torchio M, Vanni K. Contextual and temporal clinical guidelines. *AMIA Annu Symp Proc.* 1998;:683–687.
  174. Terenziani P, Montani S, Torchio M, Molino G, Anselma L. Temporal Consistency Checking in Clinical Guidelines Acquisition and Execution: the GLARE’s Approach. *AMIA Annu Symp Proc.* 2003;2003:659–663.

175. Mansour E, Wu B, Dube K, Li JX. An Event-Driven Approach to Computerizing Clinical Guidelines Using XML [Internet]. In: IEEE Services Computing Workshops, 2006. SCW '06. IEEE; 2006. p. 13–20. Available from: <http://arrow.dit.ie/ahfrcon/22>
176. De Clercq PA, Blom JA, Hasman A, Korsten HH. GASTON: an architecture for the acquisition and execution of clinical guideline-application tasks. *Med Inform Internet Med.* 2000 Dec;25(4):247–263.
177. de Clercq P, Hasman A. Experiences with the development, implementation and evaluation of automated decision support systems. *Stud Health Technol Inform.* 2004;107(Pt 2):1033–1037.
178. Allen J. Maintaining knowledge about temporal intervals. *Commun. ACM.* 1983;26(11):832–843.
179. Chomicki J. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.* 1995 Jun;20:149–186.
180. Basin D, Klaedtke F, Müller S. Monitoring security policies with metric first-order temporal logic. In: *Proceedings of the 15th ACM symposium on Access control models and technologies.* New York, NY, USA: ACM; 2010. p. 23–34.
181. Niekamp R. Software Component Architecture [Internet]. 2011 Jul 29; Available from: <http://congress.cimne.upc.es/cfsi/frontal/doc/ppt/11.pdf>
182. Conradi R, Westfechtel B. Version models for software configuration management. *ACM Comput. Surv.* 1998 Jun;30(2):232–282.
183. Stuckenholz A. Component evolution and versioning state of the art. *SIGSOFT Softw. Eng. Notes.* 2005 Jan;30(1):7–.
184. Driessen V. A successful Git branching model [Internet]. 2010 Jan 5 [cited 2012 Apr 25]; Available from: <http://nvie.com/posts/a-successful-git-branching-model/>
185. Hughes C. The representation of uncertainty in medical expert systems. *Med Inform (Lond).* 1989 Dec;14(4):269–279.
186. Wellbery C. Uncertainty in Medicine. *The Lancet.* 2010 May;375(9727):1666.
187. eHow.com. Definition of Objective Data for RNs [Internet]. 2010 May 4 [cited 2012 May 4]; Available from: [http://www.ehow.com/facts\\_6980435\\_definition-objective-data-rns.html](http://www.ehow.com/facts_6980435_definition-objective-data-rns.html)
188. Angelilli A, Ritch R. Directed therapy: An approach to the improved treatment of exfoliation syndrome [Internet]. 2009. Available from: <http://www.meajo.org/article.asp?issn=0974-9233;year=2009;volume=16;issue=1;spage=35;epage=40;aulast=Angelilli>
189. Öztürk A. Embedding the Evidence Information in Computer-Supported Guidelines into the Decision-Making Process. 2007;
190. Levy MA, Giuse DA, Eck C, Holder G, Lippard G, Cartwright J, et al. Integrated information systems for electronic chemotherapy medication administration. *J Oncol Pract.* 2011 Jul;7(4):226–230.

191. FDB (First Databank). FDB MedKnowledge (NDDF) [Internet]. 2012 May 1 [cited 2012 May 1]; Available from: <http://www.fdbhealth.com/solutions/fdb-medknowledge/>
192. Klabunde RE. Measurement of Cardiac Output [Internet]. 2009 Jul 1 [cited 2012 Apr 12]; Available from: <http://www.cvphysiology.com/Cardiac%20Function/CF021.htm>
193. Official Healthcare. Normal Body Temperature [Internet]. 2012 Mar 15 [cited 2012 Apr 12]; Available from: <http://www.healthcare-online.org/Normal-Body-Temperature.html>
194. Martin GS, Mannino DM, Eaton S, Moss M. The Epidemiology of Sepsis in the United States from 1979 through 2000. *New England Journal of Medicine*. 2003;348(16):1546–1554.
195. Bernard GR, Vincent JL, Laterre PF, LaRosa SP, Dhainaut JF, Lopez-Rodriguez A, et al. Efficacy and safety of recombinant human activated protein C for severe sepsis. *N. Engl. J. Med*. 2001 Mar 8;344(10):699–709.
196. Angus DC, Linde-Zwirble WT, Lidicker J, Clermont G, Carcillo J, Pinsky MR. Epidemiology of severe sepsis in the United States: analysis of incidence, outcome, and associated costs of care. *Crit. Care Med*. 2001 Jul;29(7):1303–1310.
197. Dellinger R, Levy M, Carlet J, Bion J, Parker M, Jaeschke R, et al. Surviving Sepsis Campaign: International guidelines for management of severe sepsis and septic shock: 2008. *Intensive Care Medicine*. 2008 Jan 1;34(1):17–60.
198. Poole JD. Model-Driven Architecture: Vision, Standards And Emerging Technologies. Workshop on Metamodeling and Adaptive Object Models, ECOOP [Internet]. 2001; Available from: [http://www.omg.org/mda/mda\\_files/Model-Driven\\_Architecture.pdf](http://www.omg.org/mda/mda_files/Model-Driven_Architecture.pdf)
199. Pinto A, Bonivento A, Sangiovanni-Vincentelli AL, Passerone R, SgROI M. System level design paradigms: Platform-based design and communication synthesis. *ACM Trans. Des. Autom. Electron. Syst*. 2004 Jun;11(3):537–563.
200. Sztipanovits J, Karsai G. Model-Integrated Computing. *Computer*. 1997;30(4):110–111.
201. Jackson E, Sztipanovits J. Formalizing the structural semantics of domain-specific modeling languages. *Software & Systems Modeling (SoSYM)*. 2008 Dec 17;8(4):451–478.
202. Resar R, Griffin AF, Haraden C, Nolan TW. Using Care Bundles to Improve Health Care Quality [Internet]. Cambridge, Massachusetts: Institute for Healthcare Improvement; 2012 [cited 2012 Jul 6]. Available from: <http://www.ihl.org/knowledge/Pages/IHIWhitePapers/UsingCareBundles.aspx>
203. Brookes SD, Hoare CAR, Roscoe AW. A Theory of Communicating Sequential Processes. *J. ACM*. 1984;31(3):560–599.
204. Gruber TR. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*. 1993;5:199–220.
205. Parr T. ANTLR (ANother Tool for Language Recognition) [Internet]. 2012 [cited 2012 Jul 8]; Available from: <http://www.antlr.org/>

206. Garshol LM. BNF and EBNF: What are they and how do they work? [Internet]. 2008 [cited 2012 Jul 8]. Available from: <http://www.garshol.priv.no/download/text/bnf.html>
207. Miller RA, Waitman LR, Chen S, Rosenbloom ST. The anatomy of decision support during inpatient care provider order entry (CPOE): empirical observations from a decade of CPOE experience at Vanderbilt. *J Biomed Inform.* 2005 Dec;38(6):469–485.
208. Sklarin NT, Granovsky S, O'Reilly EM, Zelenetz AD. Electronic Chemotherapy Order Entry: A Major Cancer Center's Implementation. *J Oncol Pract.* 2011 Jul;7(4):213–218.
209. Busby LT, Sheth S, Garey J, Ginsburg A, Flynn T, Willen MA, et al. Creating a process to standardize regimen order sets within an electronic health record. *J Oncol Pract.* 2011 Jul;7(4):e8–e14.
210. Karsai G, Sztipanovits J, Ledeczi A, Bapty T. Model-integrated development of embedded software. *IEEE Proc.* 2003;91:145–164.
211. Börger E. Abstract State Machines: A Method for High-Level System Design and Analysis [Internet]. 2003 [cited 2010 Feb 24]. Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.1254>
212. Alur R, Dill DL. A theory of timed automata. *Theoretical Computer Science.* 1994 Apr 25;126(2):183–235.
213. Hoge T. A presentation of the trace algebra of three 3x3 matrices. arXiv:1104.0904 [Internet]. 2011 Apr 5 [cited 2012 Jul 11]; Available from: <http://arxiv.org/abs/1104.0904>
214. Allenby R. Rings, Fields, and Groups: An Introduction to Abstract Algebra [Internet]. Arnold; 1991. Available from: <http://www.getcited.org/pub/103042052>
215. Harel D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.* 1987 Jun;8(3):231–274.
216. MathWorks. Stateflow - State chart design and development environment [Internet]. 2012 [cited 2012 May 24]; Available from: <http://www.mathworks.com/products/stateflow/>
217. Graphviz. Graphviz - Graph Visualization Software [Internet]. 2012 [cited 2012 Jul 19]; Available from: <http://www.graphviz.org/>
218. Vanderbilt University. Center for Experiential Learning & Assessment (CELA) [Internet]. 2012 [cited 2012 Jul 24]; Available from: <https://medschool.vanderbilt.edu/cela/>
219. Andersson G, Bjesse P, Cook B, Hanna Z. A proof engine approach to solving combinational design automation problems [Internet]. In: Design Automation Conference, 2002. Proceedings. 39th. 2002. p. 725 – 730. Available from: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=1012718&contentType=Conference+Publications>

220. MathWorks. Simulink Design Verifier Documentation [Internet]. R2011b Documentation - Simulink Design Verifier. 2012 [cited 2012 Feb 20]; Available from: <http://www.mathworks.com/help/toolbox/sldv/>
221. MathWorks. Simulink Verification and Validation [Internet]. 2012 [cited 2012 Jul 27]; Available from: <http://www.mathworks.com/products/simverification/index.html>
222. Pingree PJ, Mikk E. The HiVy Tool Set [Internet]. In: Alur R, Peled DA, editors. Computer Aided Verification. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004 [cited 2011 Nov 21]. p. 466–469. Available from: <http://www.springerlink.com/content/n7g30gfex2d1u26j/>
223. Manamcheri K, Mitra S, Bak S, Caccamo M. A step towards verification and synthesis from simulink/stateflow models [Internet]. In: Proceedings of the 14th international conference on Hybrid systems: computation and control. New York, NY, USA: ACM; 2011 [cited 2011 Nov 21]. p. 317–318. Available from: <http://doi.acm.org/10.1145/1967701.1967749>
224. Balasubramanian D, Păsăreanu CS, Whalen MW, Karsai G, Lowry M. Polyglot: modeling and analysis for multiple Statechart formalisms [Internet]. In: Proceedings of the 2011 International Symposium on Software Testing and Analysis. New York, NY, USA: ACM; 2011. p. 45–55. Available from: <http://doi.acm.org/10.1145/2001420.2001427>
225. Dwyer MB, Avrunin GS, Corbett JC. Patterns in property specifications for finite-state verification [Internet]. In: Proceedings of the 21st international conference on Software engineering. New York, NY, USA: ACM; 1999. p. 411–420. Available from: <http://doi.acm.org/10.1145/302405.302672>
226. Department of Information Technology at Uppsala University, Sweden, Department of Computer Science at Aalborg University in Denmark. UPPAAL Benchmarks [Internet]. 2012 [cited 2012 Oct 3]; Available from: <http://www.it.uu.se/research/group/darts/uppaal/benchmarks/>
227. Dubey A, Nordstrom S, Keskinpala T, Neema S, Bapty T, Karsai G. Towards a verifiable real-time, autonomic, fault mitigation framework for large scale real-time systems. Information Security Solutions Europe. 2007;;33–52.
228. Dubey A, Riley D, Abdelwahed S, Bapty T. Modeling and Analysis of Probabilistic Timed Systems [Internet]. In: Engineering of Computer Based Systems, 2009. ECBS 2009. 16th Annual IEEE International Conference and Workshop on the. 2009. p. 69 –78. Available from: [http://www.isis.vanderbilt.edu/sites/default/files/PTAVerification\\_0.pdf](http://www.isis.vanderbilt.edu/sites/default/files/PTAVerification_0.pdf)
229. Xing J, Theelen BD, Langerak R, de JP van, Tretmans J, Voeten JPM. UPPAAL in Practice: Quantitative Verification of a RapidIO Network [Internet]. In: Margaria T, Steffen B, editors. Leveraging Applications of Formal Methods, Verification, and Validation. London: Springer Verlag; 2010. p. 160–174. Available from: <http://doc.utwente.nl/72602/>
230. Leitner F, Leue S. Simulink Design Verifier vs. SPIN - A Comparative Case Study [Internet]. In: Proceedings of FMICS 2008, ERCIM Working Group on Formal Methods for Industrial Critical Systems. 2008 [cited 2012 Feb 15]. Available from: [http://www.inf.uni-konstanz.de/soft/publications\\_en.php](http://www.inf.uni-konstanz.de/soft/publications_en.php)

231. MathWorks. Types of Chart Execution - Super step semantics [Internet]. Types of Chart Execution - MATLAB & Simulink. 2012 [cited 2012 Oct 3]; Available from: <http://www.mathworks.com/help/stateflow/ug/types-of-chart-execution.html#brccxny>
232. Knappe T, Hederman L, Wade VP, Gargan M, Harris C, Rahman Y. A UML approach to process modelling of clinical practice guidelines for enactment. *Stud Health Technol Inform.* 2003;95:635–640.
233. Heard KM, Huang C, Noirot LA, Reichley RM, Bailey TC. Using BPEL to Define an Executable CDS Rule Process. *AMIA Annu Symp Proc.* 2006;2006:947.
234. Strasser M, Pfeifer F, Helm E, Schuler A, Altmann J. Defining and reconstructing clinical processes based on IHE and BPMN 2.0. *Stud Health Technol Inform.* 2011;169:482–486.
235. Huser V, Rasmussen LV, Oberg R, Starren JB. Implementation of workflow engine technology to deliver basic clinical decision support functionality. *BMC Med Res Methodol.* 2011 Apr 10;11:43.
236. Greenes RA, Sordo M, Zaccagnini D, Meyer M, Kuperman GJ. Design of a standards-based external rules engine for decision support in a variety of application contexts: report of a feasibility study at Partners HealthCare System. *Stud Health Technol Inform.* 2004;107(Pt 1):611–615.
237. Goldberg HS, Vashevko M, Postilnik A, Smith K, Plaks N, Blumenfeld BM. Evaluation of a Commercial Rule Engine as a Basis for a Clinical Decision Support Service. *AMIA Annu Symp Proc.* 2006;2006:294–298.
238. Mathe J, Miller P, Ledeczi A, Weavind L, Miller A, Maron D, et al. A Model-Integrated Approach to Implementing Individualized Patient Care Plans Based on Guideline-Driven Clinical Decision Support and Process Management - A Progress Report. In: 2nd International Workshop on Model-Based Design of Trustworthy Health Information Systems (MOTHIS 2008). Toulouse, France: 2008.
239. Mathe JL. Towards an Adaptable Framework for Modeling, Verifying, and Executing Medical Guidelines. In: Proceedings of the Doctoral Symposium at MODELS 2009. Denver, CO: 2009.
240. Hooper MH, Weavind L, Wheeler AP, Martin JB, Gowda SS, Semler MW, et al. Randomized Trial of Automated, Electronic Monitoring to Facilitate Early Detection of Sepsis in the Intensive Care Unit. *Crit. Care Med.* 2011 Aug;In press.
241. Mathe J, Werner J, Sztipanovits J. Model-Based Design of Trustworthy Health Information Systems. In: Homeland Security Facets: Threats, Countermeasures, and the Privacy Issue. London, UK: Artech House; 2011.
242. W3C. OWL 2 Web Ontology Language Document Overview [Internet]. 2009 Oct 27 [cited 2010 Feb 26]; Available from: <http://www.w3.org/TR/owl2-overview/>
243. Mathe JL, Sztipanovits J, Levy M, Jackson EK, Schulte W. Cancer Treatment Planning: Formal Methods to the Rescue. In: 4rd International Workshop on Software Engineering in Health Care (SEHC 2012). Zurich, Switzerland: 2012.

244. Giuse DA, Mickish A. Increasing the availability of the computerized patient record. AMIA Annu Fall Symp Proc. 1996;:633–637.
245. Mathe J, Werner J, Lee Y, Malin B, Ledeczi A. Model-Based Design of Clinical Information Systems. *Methods Inf Med.* 2008;47(5):399–408.
246. Duncavage S, Mathe J, Werner J, Malin BA, Ledeczi A, Sztipanovits J. A Modeling Environment for Patient Portals [Internet]. In: *Proceedings of the AMIA Annual Symposium*. Chicago, IL: 2007 [cited 2009 Aug 31]. p. 201–205. Available from: <http://www.ncbi.nlm.nih.gov/pubmed/18693826>
247. Mathe J, Duncavage S, Werner J, Malin B, Ledeczi A, Sztipanovits J. Implementing a Model-Based Design Environment for Clinical Information Systems. In: *First International Workshop on Model-Based Design of Trustworthy Health Information Systems (MOTHIS 2007)*. Nashville, TN: 2007.
248. Werner J, Mathe JL, Duncavage S, Malin B, Ledeczi A, Jirjis JN, et al. Platform-Based Design for Clinical Information Systems. In: *Industrial Informatics, 2007 5th IEEE International Conference on (INDIN 2007)*. Vienna, Austria: 2007. p. 749–754.
249. Mathe JL, Duncavage S, Werner J, Malin BA, Ledeczi A, Sztipanovits J. Towards the Security and Privacy Analysis of Patient Portals. *ACM SIGBED Review.* 2007;4(2):5–9.
250. Emerson M, Mathe J, Duncavage S. WiNeSim: A Wireless Network Simulation Tool. In: *Proceedings of the Sixth ACM International Conference on Embedded Software (EMSOFT'06 Workshop)*. Seoul, South Korea: 2006.
251. Werner J, Eby M, Mathe J, Karsai G, Xue Y, Sztipanovits J. Integrating Security Modeling into Embedded System Design. In: *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006)*. IEEE Computer Society; 2006.
252. Microsoft. XML Schema Definition Tool [Internet]. 2012 [cited 2012 Jul 14]; Available from: [http://msdn.microsoft.com/en-us/library/x6c1kb0s\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/x6c1kb0s(v=vs.80).aspx)
253. Parr T. StringTemplate Template Engine [Internet]. 2012 [cited 2012 Jul 16]; Available from: <http://www.stringtemplate.org/>